


PREFACE

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

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



Signed: Professor Nelishia Pillay

Date: 20 November 2018

Signed: Tatenda Chareka

Date: 28 November 2018

DECLARATION 1: PLAGIARISM

I, Tatenda Chareka (student number: 211506553), declare that:

- (i) the research reported in this dissertation, except where otherwise indicated or acknowledged, is my original work;
- (ii) this dissertation has not been submitted in full or in part for any degree or examination to any other university;
- (iii) this dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons;
- (iv) this dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
 - a) their words have been re-written but the general information attributed to them has been referenced;
 - b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced;
- (v) where I have used material for which publications followed, I have indicated in detail my role in the work;
- (vi) this dissertation is primarily a collection of material, prepared by myself, published as journal articles or presented as a poster and oral presentations at conferences. In some cases, additional material has been included;
- (vii) this dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References sections.

Signed: Tatenda Chareka

Date: 28 November 2018

DECLARATION 2: PUBLICATIONS

DETAILS OF CONTRIBUTION TO PUBLICATIONS that form part and/or include research presented in this thesis

Publication 1:

CHAREKA, T. & PILLAY, N. A study of fitness functions for data classification using grammatical evolution. Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), 2016, 2016. IEEE, 1-4.

Abstract

Network intrusion detection is a real-world problem that involves detecting intrusions on a computer network. Detecting whether a network connection is intrusive or non-intrusive is essentially a binary classification problem. However, the type of intrusive connections can be categorised into a number of network attack classes and the task of associating an intrusion to a particular network type is multiclass classification.

A number of artificial intelligence techniques have been used for network intrusion detection including Evolutionary Algorithms. This thesis investigates the application of evolutionary algorithms namely, Genetic Programming (GP), Grammatical Evolution (GE) and Multi-Expression Programming (MEP) in the network intrusion detection domain. Grammatical evolution and multi-expression programming are considered to be variants of GP. In this thesis, a comparison of the effectiveness of classifiers evolved by the three EAs within the network intrusion detection domain is performed. The comparison is performed on the publicly available KDD99 dataset. Furthermore, the effectiveness of a number of fitness functions is evaluated.

From the results obtained, standard genetic programming performs better than grammatical evolution and multi-expression programming. The findings indicate that binary classifiers evolved using standard genetic programming outperformed classifiers evolved using grammatical evolution and multi-expression programming. For evolving multiclass classifiers different fitness functions used produced classifiers with different characteristics resulting in some classifiers achieving higher detection rates for specific network intrusion attacks as compared to other intrusion attacks. The findings indicate that classifiers evolved using multi-expression programming and genetic programming achieved high detection rates as compared to classifiers evolved using grammatical evolution.

Acknowledgements

I would like to thank the National Research Foundation (NRF) for the financial assistance. The opinions raised and conclusions reached, are those of the author and are not attributed to the NRF. The Centre of High-Performance Computing (CHPC) for granting access to their resources.

I would like to thank my supervisor, Professor Nelishia Pillay for her constant support and guidance throughout the compilation of this thesis. I would also like to thank my family members, friends and colleagues for their continuous support and motivation.

Contents

PREFACE	i
DECLARATION 1: PLAGIARISM	ii
DECLARATION 2: PUBLICATIONS	iii
Abstract	iv
Acknowledgements	v
Contents	vi
List of Figures	xii
List of Tables	xiv
List of Algorithms	xvi
List of Abbreviations	xvii
1 Introduction	1
1.1 Purpose of the Study	1
1.2 Aims and Objectives	1
1.3 Contributions	2
1.4 Dissertation Layout	2
2 Genetic Programming	5
2.1 Introduction	5
2.2 Introduction to Genetic Programming	6
2.3 Overview of the GP Algorithm	6
2.4 Representation	7
2.4.1 Tree Based GP	7
2.4.2 Function Set	7
2.4.3 Terminal Set	8
2.5 Initial Population Generation	8
2.5.1 Full Method	9
2.5.2 Grow Method	9
2.5.3 Ramped Half and Half Method	10
2.6 Evaluation	10
2.6.1 Fitness Cases	11

2.6.2	Fitness Functions	11
2.7	Selection Methods	12
2.7.1	Tournament Selection.....	12
2.8	Genetic Operators	13
2.8.1	Reproduction	13
2.8.2	Mutation	14
2.8.3	Crossover.....	14
2.9	GP Control Models.....	15
2.9.1	Generational Model.....	16
2.9.2	Steady State Model.....	16
2.10	Termination	17
2.11	Introns and Bloat	17
2.12	Modularisation.....	18
2.13	Strengths and Weaknesses of GP	18
2.13.1	Strengths	18
2.13.2	Weaknesses	18
2.14	Introduction to Grammatical Evolution	19
2.15	Overview of the Generational GE Algorithm	19
2.16	Representation	20
2.16.1	BNF Grammar	20
2.16.2	Mapping Process	21
2.17	Initial Population Generation and Evaluation	23
2.18	Genetic Operators.....	23
2.18.1	Crossover.....	24
2.18.2	Mutation	25
2.19	Introns and Bloat	26
2.20	Strengths and Weakness of GE	27
2.20.1	Strengths	27
2.20.2	Weaknesses	27
2.21	Introduction to Multi-Expression Programming.....	28
2.22	Overview of the Steady State MEP Algorithm	28
2.23	Representation	29
2.24	Initial Population Generation and Evaluation	29

2.25	Genetic Operators.....	31
2.25.1	Crossover.....	31
2.25.2	Mutation.....	33
2.26	Introns and Modularisation.....	34
2.27	Strengths and Weakness of MEP.....	34
2.27.1	Strengths.....	34
2.27.2	Weaknesses.....	34
2.28	Chapter Summary.....	34
3	Network Intrusion Detection.....	36
3.1	Introduction.....	36
3.2	Network Intrusion Detection.....	37
3.3	Datasets for Network Intrusion Detection.....	38
3.3.1	DARPA 1998 and 1999.....	39
3.3.2	KDD Cup 99.....	40
3.3.3	NSL-KDD dataset.....	41
3.3.4	Network Attack Categories.....	41
3.4	Performance Measures.....	42
3.4.1	Confusion matrix.....	43
3.4.2	Accuracy and False Positive Rate.....	43
3.4.3	Sensitivity and Specificity.....	44
3.4.4	Precision and F-measure.....	44
3.4.5	Receiver operating characteristics.....	44
3.5	Feature Selection.....	45
3.6	Previous Work on Network Intrusion Detection.....	46
3.6.1	Evolutionary Algorithms.....	46
3.6.2	Neural Networks.....	46
3.6.3	Bayesian Networks.....	47
3.6.4	Decision Trees.....	47
3.7	Chapter Summary.....	48
4	GP and Network Intrusion Detection.....	49
4.1	Introduction.....	49
4.2	Using genetic programming for network intrusion detection.....	49
4.3	Binary Classification for NID using GP.....	50

	4.3.1	Genetic Programming.....	50
	4.3.2	Grammatical Evolution	52
	4.3.3	Linear Genetic Programming.....	53
4.4		Multiclass Classification for NID using GP.....	53
	4.4.1	Genetic Programming.....	53
	4.4.2	Grammatical Evolution	54
	4.4.3	Multi-expression Programming.....	54
	4.4.4	Linear genetic programming	54
4.5		Strengths and Weaknesses of GP in NID.....	55
	4.5.1	Strengths	55
	4.5.2	Weaknesses	55
4.6		Analysis of genetic programming in network intrusion detection	55
4.7		Chapter Summary.....	57
5		Methodology	58
	5.1	Introduction	58
	5.2	Research Methodology.....	58
	5.2.1	Aims and Objectives	58
	5.3	Proof by Demonstration Methodology.....	59
	5.3.1	Evaluation of approach.....	60
	5.3.2	Refinement of approach	60
	5.3.3	Termination Criterion.....	61
	5.4	Statistical Tests.....	61
	5.4.1	Statistical Testing	61
	5.5	Dataset.....	62
	5.5.1	Dataset description	62
	5.5.2	Dataset Pre-processing	62
	5.5.3	Binary classification dataset.....	63
	5.5.4	Multi-class classification dataset.....	64
	5.6	Distributed Architecture for Proposed Approaches	64
	5.7	Technical Specifications.....	65
	5.8	Chapter Summary.....	65
6		Genetic Programming for Network Intrusion Detection.....	66
	6.1	Introduction	66

6.2	GP Algorithm	66
6.3	Representation and initial population generation	67
6.4	Evaluation.....	68
6.5	Selection Method and Genetic Operators.....	71
6.6	Parameters	73
6.7	Chapter Summary.....	74
7	Grammatical Evolution for Network Intrusion Detection.....	75
7.1	Introduction	75
7.2	Representation	75
7.3	Initial Population Generation and Evaluation	77
7.4	Selection Method and Genetic Operators.....	78
7.5	Parameters	79
7.6	Chapter Summary.....	80
8	Multi-Expression Programming for Network Intrusion Detection	81
8.1	Introduction	81
8.2	MEP Algorithm	81
8.3	Representation	82
8.4	Initial Population Generation and Evaluation	83
8.5	Selection Method and Genetic Operators.....	83
8.6	Parameters	85
8.7	Chapter Summary.....	86
9	Results and Discussion.....	87
9.1	Introduction	87
9.2	Grammatical Evolution	88
9.2.1	Binary Classification	88
9.2.2	Multi-class classification.....	88
9.2.3	Analysis of multi-class classification for GE approach	93
9.3	Multi-Expression Programming.....	94
9.3.1	Binary Classification	94
9.3.2	Multi-class classification.....	94
9.3.3	Analysis of multi-class classification for MEP approach	99
9.4	Genetic Programming.....	100
9.4.1	Binary Classification	100

9.4.2	Multi-class classification.....	101
9.4.3	Analysis of multi-class classification for the GP approach.....	105
9.5	Comparison of GP, GE and MEP	106
9.5.1	Binary classification	106
9.5.2	Multi-class classification.....	108
9.6	Comparison with state of the art.....	109
9.6.1	Binary Classification	109
9.6.2	Multi-class classification.....	110
9.7	Chapter Summary.....	111
10	Conclusion and Future Work	113
10.1	Introduction	113
10.2	Objectives and Conclusion.....	113
	Bibliography.....	116
A.	User Manual	124
	Program requirements	124
	Initialising the Program.....	124
	Overview of the program	124
	Experiment Configurations	125

List of Figures

Figure 2.1: Tree depth.....	8
Figure 2.2: Full and Grow Tree Generation.....	9
Figure 2.3: Ramped half-and-half.....	10
Figure 2.4: Mutation operation	14
Figure 2.5: Crossover operation.....	15
Figure 2.6: GE fixed length one-point crossover.....	24
Figure 2.7: GE two-point crossover.....	24
Figure 2.8: Homologous Crossover	25
Figure 2.9: Mutation Operator Variations	26
Figure 2.10: MEP chromosome genes represented as trees.....	31
Figure 2.11: MEP one-point crossover	32
Figure 2.12: MEP two-point crossover.....	32
Figure 2.13: MEP uniform crossover.....	33
Figure 2.14: MEP Mutation	33
Figure 3.1: The NID process.....	37
Figure 3.2: DARPA, KDD99 and NSL-KDD relation.	39
Figure 3.3: ROC graph.....	45
Figure 5.1: Binary classification distribution.....	64
Figure 5.2: Multi-class Classification distribution.....	64
Figure 6.1: Example of an individual	68
Figure 6.2: Evaluation process.....	70

Figure 6.3: GP Crossover.....	72
Figure 6.4: GP Mutation	73
Figure 7.1: Binary to denary conversion.....	77
Figure 7.2: GE Individual	77
Figure 7.3: GE uniform crossover	78
Figure 8.1: MEP Individual	83
Figure 8.2: MEP Uniform Crossover.....	85
Figure 9.1: GE comparison of fitness function performance.....	93
Figure 9.2: MEP comparison of fitness function performance.....	100
Figure 9.3: GP comparison of fitness function performance.....	105
Figure 9.4: Binary classification comparison	106
Figure 9.5: Multi-class classification comparison	108
Figure A.1: Network Intrusion Detection System Main Menu.....	124
Figure A.2: Train and Test using NID System	125
Figure A.3: End of run Message	126
Figure A.4: Selecting best classifier	126
Figure A.5: Results of evaluation	127

List of Tables

Table 2.1: Fitness Cases.....	11
Table 2.2: The number of choices for each production rule.....	22
Table 3.1: KDD Cup 99 Sample Distribution.....	40
Table 3.2: KDD Cup Class distribution.....	41
Table 3.3: Network intrusion detection categories	42
Table 3.4: Binary confusion Matrix.....	43
Table 5.1: Z-hypothesis test table	61
Table 5.2: NSL-KDD sample distribution.....	62
Table 5.3: Data transformation	63
Table 6.1: Function descriptions.....	68
Table 6.2: Confusion Matrix.....	71
Table 6.3: GP Parameters for binary classifiers.....	73
Table 6.4: GP Parameters for multi-class classifiers	74
Table 7.1: GE Parameters for binary classification	79
Table 7.2: GE Parameters for multi-class classification.....	80
Table 8.1: MEP Parameters for binary classification	86
Table 8.2: MEP Parameters for multi-class classification.....	86
Table 9.1: Grammatical Evolution binary classification results.....	88
Table 9.2: Grammatical Evolution accuracy multi-classification results	89
Table 9.3: Grammatical Evolution MCC multi-classification results	90
Table 9.4: Grammatical Evolution f-score multi-classification results	90

Table 9.5: Grammatical Evolution TPR multi-classification results	91
Table 9.6: Grammatical Evolution precision multi-classification results.....	92
Table 9.7: Grammatical Evolution FPR multi-classification results	93
Table 9.8: Multi-Expression programming binary classification results.....	94
Table 9.9: MEP accuracy multi-classification results.....	95
Table 9.10: MEP Matthews's coefficient correlation multi-classification results.....	96
Table 9.11: MEP f-score multi-classification results.....	96
Table 9.12: MEP true positive rate multi-classification results.....	97
Table 9.13: MEP precision multi-classification results	98
Table 9.14: MEP false positive rate multi-classification results.....	98
Table 9.15: Genetic programming binary classification results	100
Table 9.16: GP accuracy multi-classification results.....	102
Table 9.17: GP Matthews's coefficient correlation multi-classification results.....	102
Table 9.18: GP f-score multi-classification results.....	103
Table 9.19: GP true positive rate multi-classification results	104
Table 9.20: GP precision multi-classification results	104
Table 9.21: GP false positive rate multi-classification results.....	105
Table 9.22: Statistical test results for binary classification.....	107
Table 9.23: Statistical test results for multi-class classification	109
Table 9.24: State of the art for binary classification.....	110
Table 9.25: State of the art for multi-class classification.....	111

List of Algorithms

Algorithm 2.1: Generational GP	16
Algorithm 2.2: Steady State GP	17
Algorithm 2.3: Generational GE Algorithm	19
Algorithm 2.4: Steady-State MEP Algorithm.....	28
Algorithm 5.1: Proof by demonstration	59
Algorithm 6.1: GP Algorithm.....	67
Algorithm 6.2: Tournament selection	71
Algorithm 8.1: MEP algorithm	82

List of Abbreviations

Abbreviation	Definition
NID	Network Intrusion Detection
GP	Genetic Programming
MEP	Multi-Expression Programming
GE	Grammatical Evolution
EA	Evolutionary Algorithms
GA	Genetic Algorithm
ADF	Automatically Defined Functions
IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
DARPA	Defence Advanced Research Projects Agents
AFRL	Air Force Research Laboratory
TCP	Transmission Control Protocol
DOS	Denial of Service
R2L	Remote to Local
L2R	Local to Root
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
Acc	Accuracy
FPR	False Positive rate
TPR	True Positive Rate
TNR	True Negative Rate
PPV	Precision
ROC	Receiver Operating Characteristics
SVM	Support Vector Machine
RBF	Radial Basis Function
SOM	Self-Organization Map
WEKA	Waikato Environment for Knowledge Analysis

UADR	Unknown attack detection rate
LERAD	Learning Rules for Anomaly Detection
LBNL	Lawrence Berkeley National Laboratory
MANET	Mobile Ad hoc Networks
LGP	Linear Genetic Programming
DT	Decision Trees
MCC	Matthews's coefficient correlation

1 Introduction

1.1 Purpose of the Study

Network intrusion detection is a real-world problem that involves detecting intrusions on a computer network. Detecting whether a network connection is intrusive or non-intrusive is essentially a binary classification problem. However, the type of intrusive connections can be categorized into a number of network attack classes and the task of associating an intrusion to a particular network type is multiclass classification.

Various techniques have been used for network intrusion detection including Naive Bayes classification, decision tree classification, neural networks and evolutionary algorithms, amongst others. Evolutionary algorithms such as genetic programming and its variants have been widely applied for network intrusion detection but a comparison of the performance of each variant within network intrusion detection has not been addressed. This dissertation also seeks to conduct a thorough analysis of related literature on the application of genetic programming and its variants for network intrusion detection.

1.2 Aims and Objectives

The primary objective of this dissertation is to develop, and evaluate the classification performance of genetic programming and variants of genetic programming, grammatical evolution and multi-expression programming for network intrusion detection. The objectives of this dissertation are:

Objective 1: Development and evaluation of applying grammatical evolution (GE) for generating intrusion detection classifiers

To propose and implement binary and multi-class classifiers for network intrusion detection (NID) and evaluate the performance of applying GE for evolving NID classifiers.

Objective 2: Development and evaluation of applying Multi-expression programming (MEP) for generating binary and multi-class classifiers for network intrusion detection.

To propose, implement and evaluate the performance of evolving classifiers using MEP for intrusion detection.

Objective 3: Development and evaluation of applying genetic programming (GP) for generating binary and multiclass classifiers for network intrusion detection.

Investigate the performance of binary and multi-class classifiers evolved using GP and compare the performance of the classifiers to state-of-the-art approaches.

Objective 4: Investigate the effectiveness of fitness functions for multi-class network intrusion detection

To investigate the effects of applying different fitness functions for the generation of intrusion detection classifiers and if there a correlation between the detection rate achieved by the classifier and the fitness function used.

Objective 5: Comparative analysis of GE, MEP and GP for network intrusion detection.

A comparative analysis of binary and multi-class classifiers evolved using grammatical evolution, multi-expression programming and genetic programming will be performed to evaluate which of the approaches generates the most effective classifiers.

1.3 Contributions

This dissertation makes the following contributions:

- Design and evaluation of generating effective classifiers using genetic programming, grammatical evolution and multi-expression programming.
- Comparative analysis of the effects of fitness functions when evolving binary and multi-class intrusion detection classifiers.
- Comparative analysis of the performance of intrusion detection classifiers generated using different variants of genetic programming.

1.4 Dissertation Layout

This section provides a summary of the chapters in this dissertation.

Chapter 2 – Genetic Programming

This chapter provides an introduction to genetic programming and its variants, grammatical evolution and multi-expression programming. A thorough description of each process within the variants is provided.

Chapter 3 – Network intrusion detection

Network intrusion detection (NID) is introduced in this chapter. The datasets and performance measures used within NID are also described as well as current and previous research work done within the network intrusion detection domain.

Chapter 4 – Genetic Programming and Network Intrusion Detection

This chapter reviews studies conducted within the network intrusion detection domain as well as an analysis of using genetic programming and its variants for network intrusion detection.

Chapter 5 – Methodology

The methodology used to achieve the aims and objectives outlined in Section 1.2 is discussed in this chapter. The statistical tests used to evaluate the performance of algorithms is provided in this chapter as well as a detailed description of the datasets used for this thesis.

Chapter 6 – Genetic Programming for Network Intrusion Detection

This chapter details the proposed genetic programming approach for binary and multi-class network intrusion detection.

Chapter 7 – Grammatical Evolution for Network Intrusion Detection

The grammatical evolution approach for binary and multi-class network intrusion detection is presented in this chapter.

Chapter 8 – Multi-Expression Programming for Network Intrusion Detection

The multi-expression programming approach used for binary and multi-class classification is presented in this chapter.

Chapter 9 – Results and Discussion

This chapter presents the results of each of the proposed approaches discussed from chapter six to eight. A comparison of the performance between each of the genetic programming variants is performed.

Chapter 10 – Conclusion and Future Work

This chapter summarizes the findings presented in this thesis and a conclusion to each of the objectives outlined in Chapter 1. The chapter also discusses future work which will be investigated.

2 Genetic Programming

2.1 Introduction

This chapter introduces genetic programming and its variants as well as provides details of the different aspects of the algorithm.

Sections 2.2 introduces genetic programming, followed by an overview of genetic programming in section 2.3. Representation in genetic programming is discussed in section 2.4, initial population generation methods are discussed in section 2.5. Each individual within a genetic programming population is evaluated for performance and these evaluation methods are discussed in section 2.6. Selection methods are provided in section 2.7 and genetic operators are discussed in section 2.8. Control models are discussed in section 2.9 and termination criteria used in GP are described in section 2.10. Introns and bloat are discussed in section 2.11, details of modularisation are provided in section 2.12 and the strengths and weaknesses of genetic programming are provided in section 2.13.

Section 2.14 introduces grammatical evolution, one of the variants of genetic programming. Section 2.15 provides an overview of grammatical evolution, followed by the representation in section 2.16. Initial population generation in grammatical evolution is discussed in section 2.17 and the genetic operators used in grammatical evolution are described in section 2.18. Introns and bloat in the context of grammatical evolution is discussed in section 2.19 followed by an overview of the strengths and weakness of grammatical evolution in section 2.20.

Multi-expression programming is introduced in section 2.21. Multi-expression programming is one of the variants of genetic programming. The overview of multi-expression programming is provided in section 2.22, followed by the representation in section 2.23. Section 2.24 discusses the initial population generation for multi-expression programming and genetic operators are discussed in section 2.25. Introns and modularisation within multi-expression programming is discussed in section

2.26 followed by the strengths and weaknesses of multi-expression programming in section 2.27. Section 2.28 presents a summary of the critical aspects of genetic programming and its variants.

2.2 Introduction to Genetic Programming

Evolutionary algorithms (EA) are a class of optimization algorithms from artificial intelligence which take an analogy from evolution to solve computer science problems. The user defines a goal in the form of a quality criterion and the EA uses the defined goal to measure and compare solutions through a number of iterations until an optimal or near optimal solution is found [3]. Solutions are found in the search space. The search space is a search area which contains potential solutions to a problem. Most evolutionary algorithms adopt processes such as reproduction, selection and mutation from Darwin's theory of natural selection and evolution in order to efficiently find solutions within the search space [81]. The theory of natural selection states that individuals with certain characteristics (stored in the genes) are more likely to survive and replicate their characteristics to the offspring and gradually improve the characteristics of the population created [25]. Different EAs differ in the way in which solutions are represented and the way in which new solutions are derived from existing solutions. A genetic algorithm (GA) is an evolutionary population-based algorithm that was inspired by John Holland in the early 1970s to model Darwin's theory of natural selection. A GA evolves a population of individuals towards better solutions. Each individual within the population is encoded as a string which represents a potential solution to a given problem. The GA searches for solutions to problems within the solution space [53, 81].

Genetic programming (GP) pioneered by Koza [40], is a problem solving EA in which programs are evolved to find solutions to problems. GP conducts search for a solution program to a problem in the program space. GP mimics the theory of natural selection and it is closely related to GAs. GP searches a program space and a GA searches a solution space resulting in the structure of the representations being different. GP is stochastic in nature; it is not guaranteed to find the global optimum, but a good enough solution defined by the researcher.

2.3 Overview of the GP Algorithm

A suitable representation is initially required before the GP algorithm is executed. The GP algorithm begins by generating a population of individuals made up from a combination of functions and terminals suitable for the domain. The population of individuals initially created is termed the initial population. Each individual in the initial population is assigned a value to determine how fit the individual is. The assigned value is termed as the fitness. Based on the fitness of the individual, the

algorithm can terminate execution. If a solution is found in the initial population, the algorithm terminates and returns the solution individual.

If a solution is not found in the initial population, the individuals go through transformations using genetic operators to create better individuals. Each transformation creates a generation. A generation is a population of individuals which are created using genetic operators. A selection method is used to select individuals from the population. Genetic operators are applied to the individuals selected. Individuals created from the application of genetic operators are referred to as offspring. After each generation, each offspring created is evaluated for quality. The generation of offspring iteratively continues until a solution is found or termination criterion is reached. The iterative process can either replace the whole population of individuals or specific individuals with a low fitness. The process from initial population generation, genetic operator applications, generations until a termination criterion is met is defined as a run. Each of the fundamental aspects of a GP run are discussed further in the following sections.

2.4 Representation

Elements of the GP population are programs, commonly represented as parse trees [74]. Other program representations include linear and graph representations [3]. A number of factors are considered when selecting the representation to use, these factors include efficiency, ease of implementation and information to be represented by the individuals [74]. A parse tree is comprised of elements of the function and terminal sets. The elements of the function and terminal sets are collectively called primitives. Genetic programming using a parse tree representation is also known as tree-based GP. Tree-based GP, functions and terminals are discussed below.

2.4.1 Tree Based GP

Each individual is represented as a parse tree for tree-based GP. Koza [40] represented programs in LISP (S-expression) which is equivalent to a parse tree representing a computer program. Pre-order notation is usually used to express parse trees for easy interpretation. Each parse tree is made up of one or several nodes. The first node within the parse tree is referred to as the root and the nodes that are found at the bottom of the parse tree are the leaves.

2.4.2 Function Set

The function set contains domain dependent functions. Mathematical functions, conditional statements, logical operators are examples of some of the functions. User defined functions can also be included in the function set. Each function has an arity. Arity is the number of arguments which a function takes [3].

2.4.3 Terminal Set

The terminal set is comprised of variables that make up the trees used to solve the GP problem. The variables can be of type string, real, integer or character. Constants such as ephemeral constants can be included in the trees used to solve the problem. Random ephemeral constants are values that fall within a specific range and remain unchanged during the entire duration of the run. For example, a random ephemeral constant with a range of integer values [1, 10] can be used, during a GP run if the ephemeral constant is selected, a random integer will be selected from the range [1, 10] and remain fixed for the duration of the GP run. Multiple ephemeral constants with different values can be used and the range is problem dependent. Elements of the terminal set have an arity of zero [3].

2.5 Initial Population Generation

The initial population is made up of randomly created individuals. Three methods exist for the generation of the initial population namely full, grow, and ramped half and half [40]. The generation of each individual begins by randomly selecting a function from the function set to represent the root node of the individual. The root node of the tree is selected from the function set in order to eliminate the creation of trivial trees (trees with a terminal element as the root node). Based on the arity of the root node selected, children are randomly chosen from the function and terminal sets and these are expanded iteratively in a depth first manner until a complete tree is created. The maximum depth of a tree is the distance from the root node to the bottom-most leaf node. In Figure 2.1, the root node of the individual is located at depth 1, whilst the child nodes of the root are located at depth 2 and the maximum depth of the tree is depth 4. The maximum depth of a tree is specified when creating the initial population in order to limit the size of the tree during initial population generation.

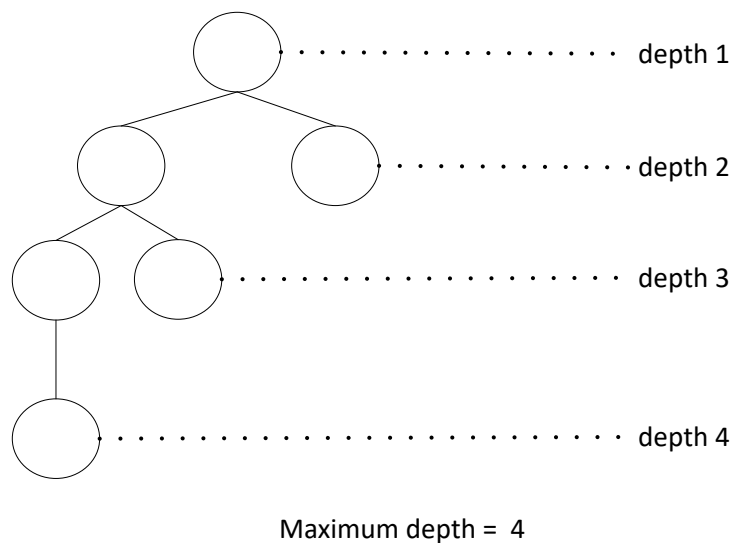


Figure 2.1: Tree depth

If the search space is not sufficiently represented during initial population generation, it may lead to premature convergence to a local optimum of the GP algorithm. The search space has to be sufficiently represented in order to increase the chances of finding a global optimum. The number of individuals created is controlled by the population size which is specified as one of the parameters of a GP algorithm.

2.5.1 Full Method

The full method creates individuals which have a balanced tree. Balanced trees have all the leaf nodes at the same depth. The internal nodes for the trees are randomly selected from the function set only until the maximum tree depth is reached. At the maximum tree depth, only nodes from the terminal set are selected. Figure 2.2a illustrates a tree created using the full method. Trees created using full might not have the same number of nodes due to different functions possessing different arity values. The method promotes less variety within the population due to the similarity in the structure of the individuals created.

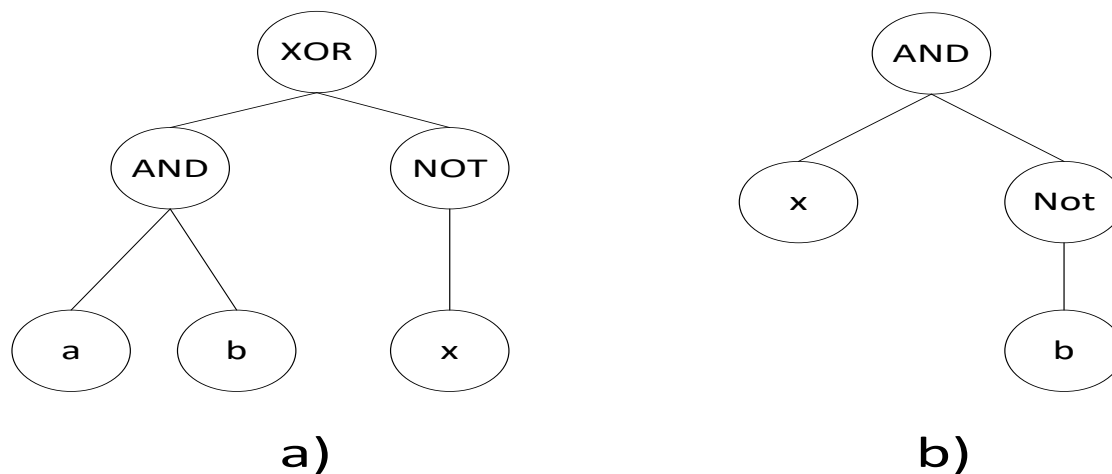


Figure 2.2: Tree individuals created using a) Full method and b) Grow method

2.5.2 Grow Method

The grow method creates individuals with irregular shapes and sizes [40]. The root of the individual is randomly selected from the function set. The rest of the nodes are randomly selected from either the function or terminal set until the tree depth limit is reached. Once the tree depth limit is reached only elements from the terminal set are selected. Figure 2.2b illustrates an individual created by the grow method. The grow method promotes greater variety within the population due to individuals possessing different shapes and sizes.

2.5.3 Ramped Half and Half Method

The ramped half and half method combine the full and grow methods discussed above. Half the population is created using the full method and the other half of the population is created using the grow method. An equal number of trees of each depth are also created. Koza [40] introduced this method of generation to provide a wide variety of trees created with different sizes and shapes.

For example, given a population size of 16 with a maximum tree depth of 5, at each tree depth half the population is created using the full method and the other half using the grow method. This means that at depth of 2, two individuals are created using the grow method and another two using the full method, at depth 3, two individuals using grow and two individuals using full, this continues until the maximum tree depth of 5 is reached. Figure 2.3 illustrates individuals created using the ramped half-and-half method.

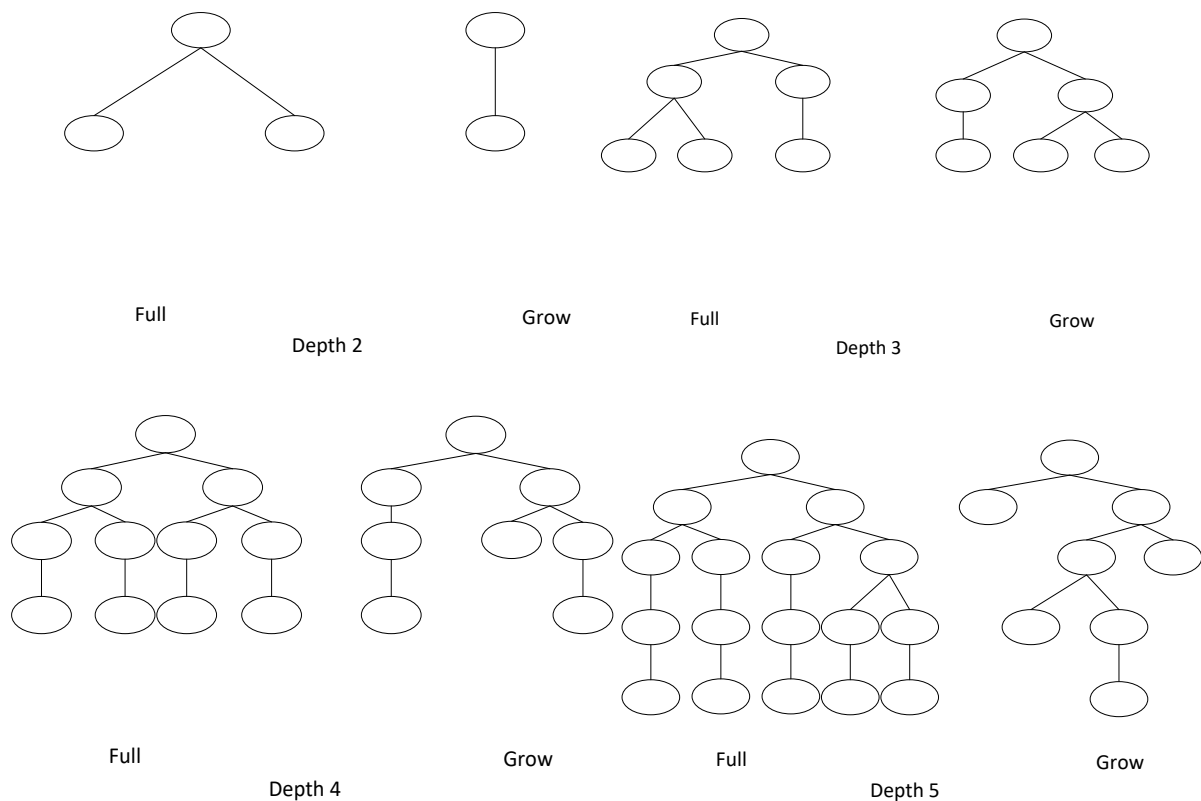


Figure 2.3: Ramped half-and-half

2.6 Evaluation

Each individual within a population is evaluated in terms of how well it solves a problem. Fitness provides a measure to the GP algorithm regarding which individuals should be given a higher

probability of being removed from the population as well as which individuals should be allowed to reproduce and recombine with other individuals within the population [3]. Evaluating the fitness of an individual is problem dependent and literature provides a vast amount of methods to use. Fitness cases and fitness functions are used to calculate fitness.

2.6.1 Fitness Cases

Programs within the population are executed over a set of different training cases. These training cases are referred to as fitness cases. Fitness cases are input-output pairs which describe the output to be produced by individuals given particular input values [3]. The success of a GP algorithm is to an extent dependent on the choice of fitness cases. Fitness cases should provide a good ratio of representing the problem domain to ensure generalization over the solutions produced by a GP run. The fitness of an individual is a function of the output produced by the individual and the target value for each fitness case. Table 2.1 provides an illustration of fitness cases.

Input Values		Output
X	Y	
4	10	116
5	7	74
6	3	45
7	9	130

Table 2.1: Fitness Cases

2.6.2 Fitness Functions

A numerical measure of how well an individual represents a solution is calculated using a fitness function. They can be used to evaluate how well an individual expresses the fitness cases [3]. The fitness function is a fitness measure that is used to compare different individuals within the population with respect to how far or close an individual is from the desired output. Different fitness functions are used for different problem domains. Fitness functions play an important role in driving the GP algorithm towards the global optimum and they should be designed carefully in order to prevent them from driving the algorithm towards local optima [42]. Raw fitness is one of the simplest and most commonly used fitness measures. It measures how promising an individual is at solving the problem. The error function is another commonly used fitness function. In some domains a high raw fitness represents a better individual whereas in some domains, it represents a weak individual [40]. Another fitness measure commonly used is the number of hits. The number of hits is the number of fitness cases for which the value produced by an individual is the same for each fitness case. It is used to

determine whether a solution has been found. For example, if a GP individual represents an expression $x^2 + y^2$ based on the fitness cases provided in Table 2.1, the number of hits would be 4 signifying a solution to the problem since there are four fitness cases. The greater the number of hits, the better the individual. In certain problem domains, the raw fitness is equivalent to the hits ratio.

Fitness functions have either a single objective or multiple objectives where two or more different measures are combined to solve a problem. These fitness functions are referred to as multi-objective fitness functions [3].

2.7 Selection Methods

Selection methods are methods used to select individuals responsible for offspring generation. The selected individuals are referred to as parents. Selection methods use fitness measures to select parents. Commonly employed selection methods are tournament selection and fitness-proportionate selection [40, 74]. Other selection methods used include truncation, ranking, linear and exponential selection [3].

Selection methods offer different effects on evolution and offspring generation. One of the effects is referred to as selection pressure. Selection pressure is the degree to which fitter individuals are favoured. Algorithms with high selection pressure favour fitter individuals as compared to algorithms with lower selection pressure. Selection pressure also controls the convergence rate of a GP approach; very high selection pressure may lead to premature convergence whilst a low selection pressure leads to a slower rate of convergence [52].

2.7.1 Tournament Selection

A random number of individuals are selected from the population to perform in a tournament. Comparison of each of the individuals within the tournament using the fitness value is performed. Based on the fitness the best individual is returned. The number of individuals randomly selected is referred to as the tournament size. A small tournament size promotes lower selection pressure and a high tournament size promotes higher selection pressure [3]. Tournament selection is commonly used within GP.

Tournament selection can also be applied inversely. Inverse tournament selection is applied in the same manner with tournament selection but instead of returning the best individual from the tournament, the worst individual is returned. Inverse tournament selection is used within the steady state GP control method discussed later in this chapter.

2.8 Genetic Operators

Genetic operators are search operators used to create individuals within a population. Genetic operators alter existing individuals in the hope of generating better offspring which solve the problem [40]. The offspring created are of different sizes and shapes as compared to their parents. Genetic operators are used to explore different areas of the program space through exploitation and exploration. Exploration is used to visit entirely new regions of the program space and genetic operators which favour exploration are termed as global search operators. Exploitation on the other hand is used to visit regions of the program space within the neighbourhood of previously visited areas. Local search operators is the term associated with genetic operators which make use of exploitation [14]. A good ratio between exploitation and exploration needs to be maintained to ensure the search converges to a global optimum. During the evolution process, exploration is recommended at the early stages rather than exploitation to ensure the best area of the program space is explored and as the evolution progresses exploitation is more favoured to ensure that the algorithm converges.

Various genetic operators have been used during the evolutionary process of GP. The three most commonly used genetic operators, reproduction, mutation and crossover, are discussed in detail below. Other genetic operators include permutation, decimation, encapsulation, hoist, create [40]. The choice of genetic operators to use is usually probabilistic and the probability of application is referred to as operator application rates [74]. The operator rates are used to determine the number of offspring created by each of the genetic operators. The rates can be represented as percentage values, for example, a population of 200 individuals and a 50%, 30% and 20% application rate for crossover, mutation and reproduction respectively results in 100 individuals being created using crossover, 60 offspring created using mutation and 40 individuals created using the reproduction operator. Operator application rates are specified at the beginning of a GP run.

Genetic operators can create very large offspring and pruning can be used to ensure that the individuals do not grow beyond a certain size. This is achieved by replacing all the function nodes at a specified tree depth (offspring depth) with randomly selected terminal nodes.

Some genetic operators have been criticized for being destructive. One of the destructive effects is breaking good building blocks that could be used to form part of a solution [57].

2.8.1 Reproduction

During reproduction, an individual is selected using one of the selection methods and copied to the next generation without any alterations [3, 40, 74].

2.8.2 Mutation

Mutation is a global search operator which creates an offspring by changing components of a single parent selected using one of the selection methods. Different variations of mutation exist such as shrink mutation, point mutation and subtree mutation. Subtree mutation is the most widely used form of mutation. Subtree mutation randomly selects a point within the selected individual (referred to as the mutation point) and replaces the subtree rooted at the mutation point with a newly randomly created subtree [40]. The grow method of population generation is generally used to create the subtree and mutation depth controls the depth of the subtree. Figure 2.4 illustrates the subtree mutation operator. Mutation promotes diversity within the population.

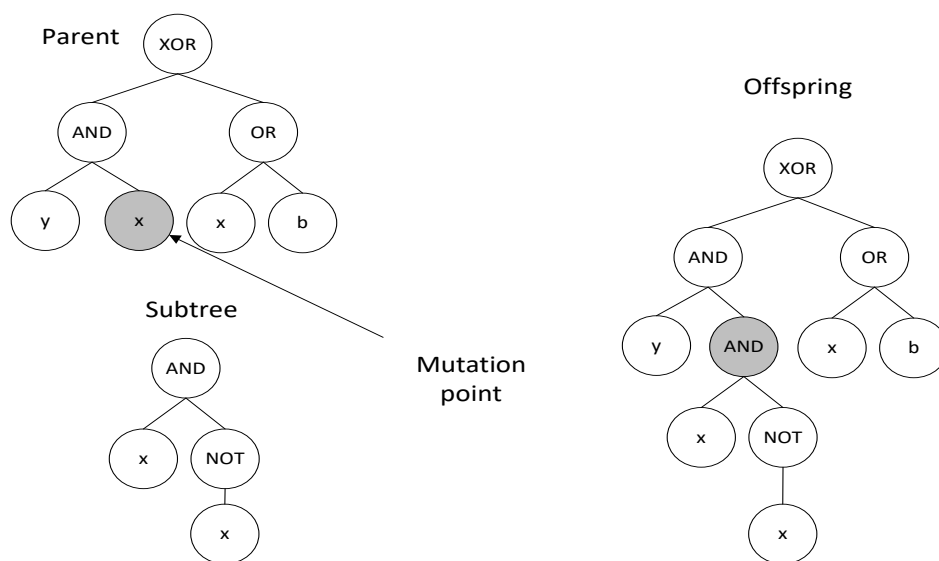


Figure 2.4: Mutation operation

2.8.3 Crossover

The crossover operator is a local search operator which generates two offspring by exchanging different components (genetic material) between two parents. Two parents are selected using a selection method. A crossover point is randomly selected in each of the two parents. The subtrees at the selected points are exchanged between the two parents to create two offspring [3]. Figure 2.5 illustrates subtree crossover. Crossover promotes convergence within the population.

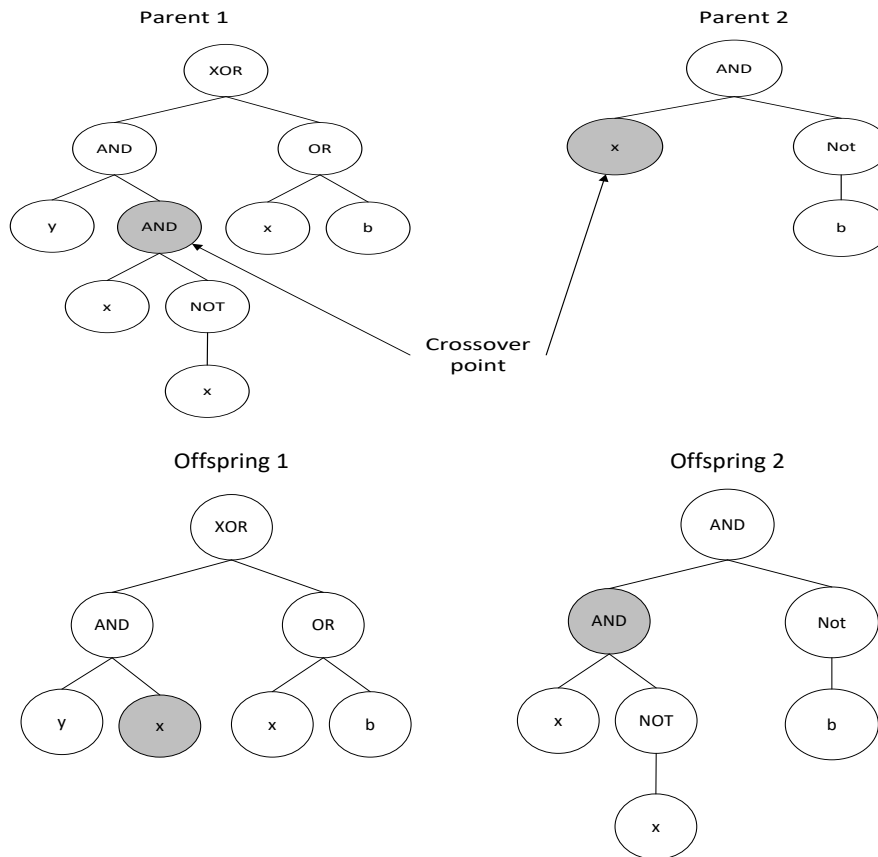


Figure 2.5: Crossover operation

Crossover has been criticised for being a destructive genetic operator. It has the ability to insert a good building block into an individual that does not make proper use for it. Some authors have argued that the closer a tree is to a solution the more susceptible it is to the destructive effect of crossover [57].

2.9 GP Control Models

There are two major models used to control the implementation of GP, the generational model and the steady-state model [3]. In generational GP, individuals in the population are replaced by new individuals after each iteration (termed as a generation). In steady-state GP, the weaker individuals are replaced as the evolutionary process continues. The two models are discussed further below.

2.9.1 Generational Model

The generational control model illustrated by Algorithm 2.1 creates a new population from the

Generational GP Algorithm

Begin

- Randomly initialize the population
- Repeat
 - Evaluate the individual programs in the existing population.
 - Select an individual or individuals in the population using selection methods.
 - Perform genetic operators on the selected individual or individuals.
 - Insert the results of genetic operators into the new population.
- Until a termination criterion is met.

End

Return the best individual from the population or solution to the problem.

Algorithm 2.1: Generational GP

previous population [3]. The algorithm randomly initializes the population using one of the initial population generation methods discussed in above. The fitness of the individuals in the generation are evaluated. A selection method is used to select parents which the genetic operators are applied to. The offspring created from the results of genetic operators are inserted in the next population. Iterations of the fitness evaluation and offspring creation are performed until a termination criterion is met.

2.9.2 Steady State Model

Algorithm 2.1 provides the steady-state control GP algorithm. Individuals are selected from the population using selection methods. Genetic operators are performed on the offspring returned from the selection methods [3]. Inverse selection methods are used to select the individual replaced by the offspring.

Steady-State GP Algorithm

Begin

- Randomly initialize the population
- Repeat
 - Randomly choose a subset of existing population to take part in tournament.
 - Evaluate subset individuals in the tournament.
 - Obtain the winner or winners from subset tournament.
 - Perform genetic operations on the winner or winners.
 - Apply inverse selection method and replace individual with results of genetic operations.
- Until a termination criterion is met.

End

Return the best individual from the population or solution to the problem.

Algorithm 2.2: Steady State GP

2.10 Termination

Termination criteria are measures used to stop the execution of the GP run. Different termination criteria are used within GP. Termination is problem dependent [74]. One of the most commonly used termination criteria is when a solution is found. In the event that a solution is not found, the best individual throughout all the generations is returned. The maximum number of generations can also be used as a termination criterion [40].

2.11 Introns and Bloat

Introns are blocks of redundant code that have no effect on the fitness of an individual. (NOT(NOT(X))) is an example of an intron, this block of code does nothing within an individual. Bloat is the exponential program growth without any significant increase in terms of fitness [74]. Rapid increase of introns leads to bloat [3]. Bloat increases exponentially towards the end of a GP run and causes the GP algorithm to be stagnate. Introns can reduce the destructive effects of genetic operators [3]. Different methods such as the use of parsimony pressure have been implemented to reduce bloat. Modularisation has also been used to reduce introns and bloat [49].

2.12 Modularisation

Modularisation is commonly used for problem-solving by which functional units of a program are identified and packaged for reuse. The methods encapsulate blocks of code. The encapsulated blocks of code become functions added to the function set and used in the creation of offspring. Modularisation attempts to tackle some of the main problems associated with GP; scaling and inefficiency [3]. Operators such as encapsulation and compression are some of methods which cater for modularisation of programs in GP [2, 40]. Automatically defined functions (ADFs) [8, 39] also caters for modularisation and enables GP to solve problems better [73].

2.13 Strengths and Weaknesses of GP

2.13.1 Strengths

- Seeding is used for each GP run and this results in different solutions obtained for each run.
- Easy interpretation and execution of solutions since each solution resembles a computer program.

2.13.2 Weaknesses

- Premature convergence due to lack of genetic diversity and the destructive effect of genetic operators.
- A number of parameters are required to execute a GP run. Optimization of these parameters is essential in order to get the best solution for each problem domain.
- No guarantee that GP will find a global optimum solution due to the stochastic nature of genetic programming.
- Large run times can be experienced during GP execution.

2.14 Introduction to Grammatical Evolution

Grammatical Evolution (GE) is a grammar-based variation of Genetic Programming pioneered by Ryan *et al.* [59, 76]. GE performs the evolutionary process on variable-length binary strings unlike GP which performs its evolutionary process on actual programs. A bit within the binary string is referred to as an allele and a combination of 8 alleles form a codon. Each binary string codon represents an integer value which is used in a mapping process. GE is inspired by the biological process of generating a protein from the genetic material of an organism follows a similar mapping process. GE unlike GP, is a population of linear genotypic binary strings, which are transformed into functional programs through a genotypic-to-phenotypic mapping process [19, 60]. One of the weakness of GP is the inclusion of redundant code within individuals and GE minimizes redundant code [63].

2.15 Overview of the Generational GE Algorithm

The genotypic-to-phenotypic mapping process is governed by the use of a Backus-Naur Form (BNF) grammar, which describes the syntax of the language for the problem. Algorithm 2.3 provides an overview of the generational GE algorithm. The mapping process takes input (BNF grammar) and

Generational GE Algorithm

Begin

- Randomly initialize the population
- Repeat
 - Evaluate the individual programs in the existing population.
 - Select an individual(s) in the population using selection methods.
 - Perform genetic operations on the selected individuals genotypic string(s).
 - Perform mapping process and generate phenotype.
 - Insert the offspring into the new population.
- Until a termination criterion is met.

End

Return the best individual from the population or solution to the problem.

Algorithm 2.3: Generational GE Algorithm

produces an output (programs). The created individuals within the population are evaluated for fitness. If a solution is not found within the initial population, the algorithm iteratively selects parents from the population using one of the selection methods. Genetic operators are applied to the

genotypic strings. The mapping process is repeated using the new genotypic strings created after genetic operators. The offspring created are evaluated for fitness. If a solution is not found, the evolutionary process iteratively continues until a termination condition is met.

2.16 Representation

Each element of the population is made up of a randomly generated binary string or denary string. This genotype is then converted to a program by the mapping process. The BNF grammar is used to define the syntax of valid programs [34]. BNF grammar and the mapping process are discussed in detail in the following sections.

2.16.1 BNF Grammar

BNF is a metalanguage (i.e. a language used to describe a language) which consists of the symbol ‘ $::=$ ’, denoting “is composed of”; and ‘ $|$ ’ meaning a choice. It provides a notation for expressing the grammar of a language in the form of production rules. The BNF grammar is made up of the tuple N, T, P, S ; where N is the set of all non-terminal symbols, T is the set of terminals, P is the set of productions rules that map N to T , and S is a member of N and the start symbol [60]. An example production rule is of the form:

$$\begin{aligned} \langle \text{expression} \rangle ::= & \langle \text{variable} \rangle \quad \langle \text{operator} \rangle \quad ge \\ & | \quad ge \end{aligned}$$

Where the non-terminals take the form $\langle \text{expression} \rangle$ (enclosed in angle brackets), and ge is an example of a terminal symbol.

The above production rule states that an $\langle \text{expression} \rangle$ is composed of the non-terminal grammar for $\langle \text{variable} \rangle$ and $\langle \text{operator} \rangle$ as well as the terminal symbol ge . Alternatively, $\langle \text{expression} \rangle$ is composed solely of the terminal symbol ge . Production rules which can generate terminal symbols are referred to as terminal-producing production rules [13].

Terminals in the context of GE are elements that appear in the program produced by GE. These terminals include operators such as $*$, $+$, $-$, $/$ and the values they operate on such as constants, variables. Terminals are not limited to just operators and values; control statements and other structures can be referred to as terminals.

The evolutionary process evolves binary strings and uses the binary strings evolved, the grammar and the mapping process in order to generate complete phenotypic programs [19, 60]. Domain

knowledge of a problem can be included within the grammar resulting in the generation of good solutions.

2.16.2 Mapping Process

The mapping process provides a distinction between the search space and the solution space [60]. A suitable BNF must be defined. This grammar specifies the syntax of the phenotypic programs to be produced by the GE. The genotype is used to map the start symbol to the terminals by converting the 8-bit codons into integer values. Taking the leftmost integer codon value, the following mapping function is applied when selecting the appropriate production rule to use:

$$\text{Rule} = \text{Codon Integer Value} \bmod \text{number of rules for the current non-terminal}$$

Where the **MOD** function returns the remainder after a division operation (e.g. $3 \bmod 2 = 1$).

The result produced after applying the mapping function corresponds to the production rule used to replace the Start symbol. If the production rule selected contains non-terminals, the leftmost non-terminal is expanded first. The next integer codon in the chromosome is selected and the mapping function is applied to the next non-terminal. An example of the mapping process is provided below.

If a production rule selected contains non-terminals, the mapping function is iteratively applied to the leftmost nonterminal symbol until one of the following situations arises:

1. When all the non-terminals are converted to elements from the terminal set.
2. The end of the genome is reached and the wrapping operator is applied whereby the mapping is iteratively repeated until a threshold of the maximum number of iterative repeats has been reached during mapping process.

During the mapping process when individuals run out of codons to traverse the genome, the individuals are wrapped around and the codons are reused from the leftmost codon integer value. When the maximum number of wrapping is reached and the individual is still incompletely mapped, the mapping process is stopped and the individual is assigned the lowest fitness value [63]. Another termination criterion of the mapping process is also provided in Genr8 [34] where only the production rules that generate terminals are used to replace the non-terminals in the expression when the threshold of the number of wraps is exceeded.

2.16.2.1 Mapping process example

Consider the grammar:

$$R = \{N, T, S, P\}$$

Where:

Nonterminal symbols (N) are $\{exp, op, var\}$,

Terminal set (T) is $\{+, -, /, *, X, 1\}$,

Start symbol (S) is $\langle exp \rangle$ and

Production rules (P) are:

$\langle exp \rangle ::= \langle exp \rangle \langle op \rangle \langle exp \rangle \quad (0)$

$\quad \quad \quad | \quad \langle var \rangle \quad (1)$

$\langle op \rangle ::= + \quad (0)$

$\quad \quad \quad | \quad - \quad (1)$

$\quad \quad \quad | \quad / \quad (2)$

$\quad \quad \quad | \quad * \quad (3)$

$\langle var \rangle ::= X \quad (0)$

$\quad \quad \quad | \quad 1 \quad (1)$

Rule Number	Choices
$\langle exp \rangle$	2
$\langle op \rangle$	4
$\langle var \rangle$	2

Table 2.2: The number of choices for each production rule

Binary String (this is an element of the population)

00010100	00100001	00010010	00010011	00100011	00000111	00001111	00100000	...
----------	----------	----------	----------	----------	----------	----------	----------	-----

Binary to Integer (Denary) conversion

Denary Values

20	33	18	19	35	7	15	32	...
----	----	----	----	----	---	----	----	-----

Start	=	<exp>	20 % 2 = 0
		<exp> <op> <exp>	33 % 2 = 1
		<var> <op> <exp>	18 % 2 = 0
		X <op> <exp>	19 % 4 = 3
		X * <exp>	35 % 2 = 1
		X * <var>	7 % 2 = 1
		X * 1	Mapping Complete

From the example above the individual created (phenotypic program) from the genotypes and the mapping process is $(x * 1)$. Some of the integer genotypes were not used during the mapping process.

2.17 Initial Population Generation and Evaluation

Initial population generation involves creating random chromosomes. The number of chromosomes created is determined by the population size specified in the parameters. Each chromosome is composed of random binary strings. The number of codons specified as a parameter determines the number of binary strings in each chromosome. Each binary string codon in the chromosome is converted to a denary value.

Evaluation of the chromosome takes place by applying the mapping process using the denary values and the grammar provided to generate a program. The wrap-over threshold limit is also specified as one of the parameters before the program executes to ensure the mapping process terminates when a specified number of wraps is exceeded. Evaluation methods, selection methods and termination criterion used within GE are the same as processes described for genetic programming in sections 2.6, 2.7 and 2.10 respectively.

2.18 Genetic Operators

Genetic operators are used to create the next generation during the evolution process. Dempsey [19] stated that genetic operators from Holland's Genetic Algorithm (GA) can be applied to the genotypic strings. Some of the operators employed from GAs are discussed below.

2.18.1 Crossover

Crossover generates offspring by combining genotypic material from two parents selected using selection methods. A number of crossover operators have been applied in GE. These include one-point crossover, two-point crossover and homologous crossover.

One-point crossover [60] randomly selects a crossover point in each of the parent binary string codons. Alleles located after the crossover point are swapped between the two individuals to generate two offspring. Figure 2.6 illustrates one-point crossover, where the crossover point is four. The highlighted strings represent alleles from parent 1 and the non-highlighted strings represent alleles from parent 2.

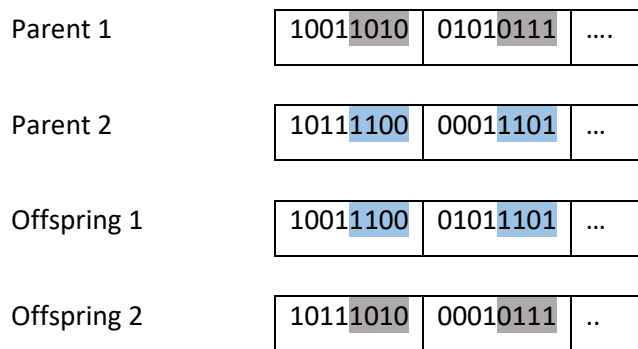


Figure 2.6: GE fixed length one-point crossover

For each codon in the parents, two-point crossover randomly selects two crossover points and swaps alleles located between the crossover points [60]. Figure 2.7 illustrates two-point crossover, where the crossover points are at position three and 7.

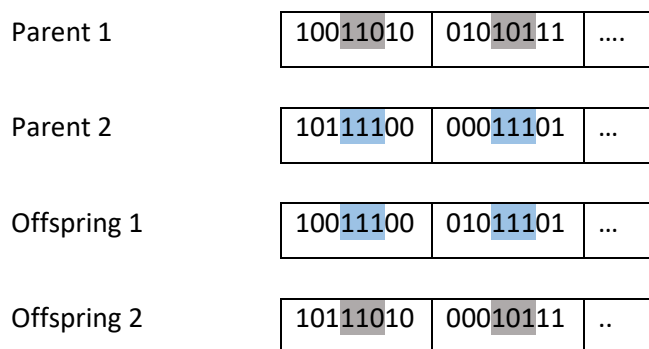


Figure 2.7: GE two-point crossover

Homologous crossover [60] is a modified two-point crossover where the mapping process history of production rules used is kept for each of the individuals. Homologous crossover is applied to the denary values and not the binary strings. The mapping process history of the selected individuals is read from the left until a region of similarity (when the same production rule is selected on both individuals) is found. The first crossover point is selected at the boundary of the region of similarity in

both the individuals. The second crossover point is randomly selected from the region of dissimilarity (when the production rules selected are different on both individuals). The codons located between the two crossovers points are then swapped between the individuals. Homologous crossover has two variations, one that swaps blocks of the same size and the other which swaps blocks of differing lengths.

Figure 2.8 illustrates homologous crossover which swaps blocks of the same size.

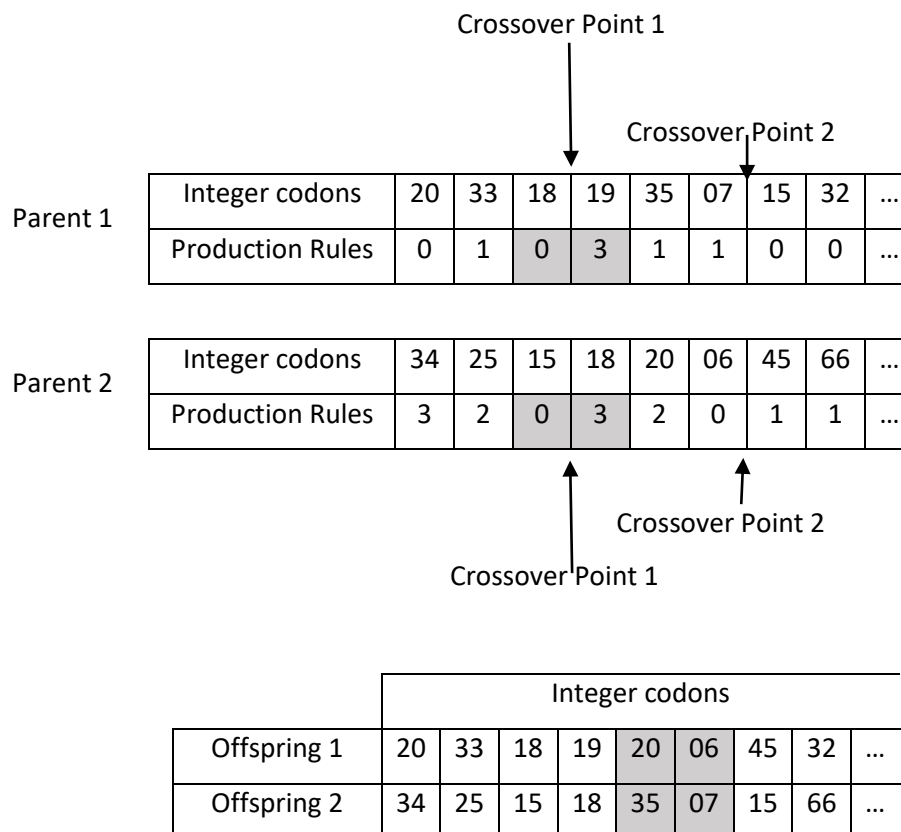


Figure 2.8: Homologous Crossover

In Figure 2.8, the region of similarity is denoted by the area highlighted in the parent integer codons. The second crossover point is the same in both individuals and the region highlighted in the offspring signifies the codons which were swapped between the two parents.

Homologous crossover requires more memory for execution compared to the other approaches and there is no clear procedure in the event that there is no region of similarity between the parents.

2.18.2 Mutation

A suitable parent is selected using the selection methods and mutation changes a bit or an integer value to another random value within the genotype of the parent generating an offspring. Changes

within the genotypic strings might however have no effect on the phenotype. For example, given the following BNF production rule:

$$\begin{aligned} \langle \text{var} \rangle &::= X && (0) \\ &| 1 && (1) \end{aligned}$$

where $\langle \text{var} \rangle$ can either be replaced by variable X or 1. If mutation is performed and each time a binary string which evaluates to an even number is created, the production rule will always select the same variable X. This is termed to as neutral mutation [19].

Different mutation operators are adopted from GA. Some of the mutation operators are discussed below. The bit flip mutation operator inverts the alleles in the binary string meaning that if an allele is a 0, it is changed to 1 or if it is a 1, it is changed to 0. Interchanging mutation randomly selects two points within the binary codon and the alleles corresponding to the positions are interchanged. Flipping mutation randomly generates a binary string (mutation chromosome). The mutation chromosome is aligned with the parent binary codon and traversed from left to right. Whenever a 1 is found in the mutation chromosome, the corresponding bit in the parent binary codon is flipped (0 to 1 and 1 to 0) generating the offspring [81]. Figure 2.9 illustrates the different mutation operators discussed above. The highlighted alleles represent where the changes took place.

Bit Flip mutation	Parent	10101010
	Offspring	0 1010 1 01
Interchanging mutation	Parent	10111110
	Mutation points	Points 4 and 8
	Offspring	101 1 111 1
Flipping mutation	Parent	00101110
	Mutation Chromosome	1 000 1 00 1
	Offspring	1 000 0 00 1

Figure 2.9: Mutation Operator Variations

2.19 Introns and Bloat

Introns are part of the genotypic strings which are not used during the mapping process. If all the non-terminals are not expanded without using all the codons, all the remaining codons are introns. If introns within individuals grow exponentially, they result in bloat as previously discussed in section 2.11. Different methods of controlling introns and bloat have been applied within GE. The use of

parsimony pressure and modularisation are among the methods used to control bloat within GE [33, 58].

2.20 Strengths and Weakness of GE

GE benefits from some of the strengths and weakness addressed in section 2.13 but it also has other strengths and shortcomings different from GP which are discussed below.

2.20.1 Strengths

- The GE wrapping operator allows short chromosomes to translate into very long expressions and provides an efficient way of avoiding invalid expressions.
- Domain knowledge can be included within the BNF grammar resulting in better tailor-made solutions to problems.
- Bloat does not occur in the phenotype solutions, making solutions produced easy to understand.

2.20.2 Weaknesses

- Longer computational time is required to translate from the genotype to the phenotype during program execution.
- Different genotype strings can map to the same phenotype string reducing the diversity of using different genotypic strings.

2.21 Introduction to Multi-Expression Programming

Multi-Expression Programming (MEP) was first introduced by Oltean [62] as a variant of Genetic Programming (GP). MEP enables the automatic generation of computer programs in a similar manner to standard GP. MEP use a linear solution representation and each MEP individual (chromosome) encodes multiple linear expressions (computer programs) referred to as *genes*. MEP individuals have the ability to encode several syntactically correct expressions in a chromosome, this ability is referred to as *strong implicit parallelism*. MEP has been applied to many different domains such as symbolic regression, classification, data analysis, evolving evolutionary algorithms, intrusion detection.

2.22 Overview of the Steady State MEP Algorithm

Similar to standard GP, the first step is to randomly create an initial population of MEP individuals. Each individual in the population is evaluated based on the fitness function. The algorithm iteratively selects two parents randomly from the current population and applies the crossover operator to both parents obtaining two offspring. The mutation operator is then applied to both offspring and if the fitness of the best offspring is better than inverse selection method individual (poorer individual), the poorer individual is replaced with the best offspring. The iteration continues until a termination

Steady-State MEP Algorithm

Begin

- Randomly create an initial population
- Repeat
 - Randomly select two parents from current population.
 - Apply crossover to the parents to generate two offspring.
 - Apply mutation to offspring.
 - **If** the fitness of the best offspring is better than inverse selection method individual,
then
 - Replace inverse selection individual with best offspring.**End if**
- Until a termination criterion is met.

End

Return the best individual from the population.

Algorithm 2.4: Steady-State MEP Algorithm

criterion is met [61]. Typically, the termination criterion is met when a solution is found or the

maximum number of iterations is reached [40]. If a solution is not found, then the best solution found during evolution is returned. Algorithm 2.4 illustrates a steady-state MEP algorithm.

2.23 Representation

MEP chromosomes are made up of genes of variable length. The length of a chromosome is equivalent to the number of genes within the chromosome. A gene can either be a terminal gene or a function gene [61]. A gene within a chromosome is similar to an individual generated using standard GP. A terminal gene is created when only a terminal symbol is selected from the terminal set. A function gene is created by combining a function symbol from the function set and pointers representing other genes within the chromosome. The length of a chromosome is specified as one of the parameters for a MEP run. Evaluation of MEP individuals is similar to the evaluation for standard GP.

2.24 Initial Population Generation and Evaluation

The first gene of a chromosome must be a terminal symbol in order to allow syntactically correct programs to be generated. For all the other genes, either a terminal gene or function gene can be encoded. Function genes include pointers to the function arguments. Function arguments always have positions of lower numerical value than the position of the function gene. Each of the genes in the chromosome are evaluated for fitness and the gene with the best fitness is used to represent the overall fitness of the chromosome. When more than one gene possess the best fitness, the first detected is chosen to represent the chromosome [29, 61]. The MEP chromosome is interpreted into a computer program in a top down manner. A terminal gene specifies a simple expression. A function gene specifies an expression obtained by connecting the operands specified by the argument positions with the function symbol. An example to explain the initial population generation process is provided below and Figure 2.10 illustrates the genes within the chromosome expressed as trees.

The numbers on the left represent gene positions.

Function Set (F) = {+, *, -, /, sin},

Terminal Set (T) = {a, b, c}.

An example of an individual using the sets F and T is given below:

- 1: a
- 2: b
- 3: $+ 1, 2$
- 4: $\sin (2)$

- 5: c
- 6: - 5, 1
- 7: / 2, 3
- 8: / 2, 6
- 9: * 8, 4
- 10: - 7, 8
- 11: + 1, 9

Genes 1, 2 and 5 encode simple expressions formed by a single terminal symbol. These expressions are:

$$E_1 = a,$$

$$E_2 = b,$$

$$E_5 = c.$$

Gene 3 applies the operation + to the operands located at 1 and 2 of the chromosome.

$$E_3 = a + b$$

Gene 4 applies the operation *sin* to operands located at 2.

$$E_4 = \sin(b)$$

Gene 6 applies the operation - to the operands located at 5 and 1.

$$E_6 = c - a$$

Gene 7 applies the operation / to the operands located at 2 and 3.

$$E_7 = b / (a + b)$$

Gene 8 applies the operation / to the operands located at 2 and 6.

$$E_8 = b / (c - a)$$

Gene 9 applies the operation * to the operands at 8 and 4 of the chromosome.

$$E_9 = b / (c - a) * \sin(b)$$

Gene 10 applies the operation - to the operands at 7 and 8 of the chromosome.

$$E_{10} = b / (a + b) - b / (c - a)$$

Gene 11 applies the operation + to the operands at 1 and 9 of the chromosome.

$$E_{11} = a + b / (c - a) * \sin (b)$$

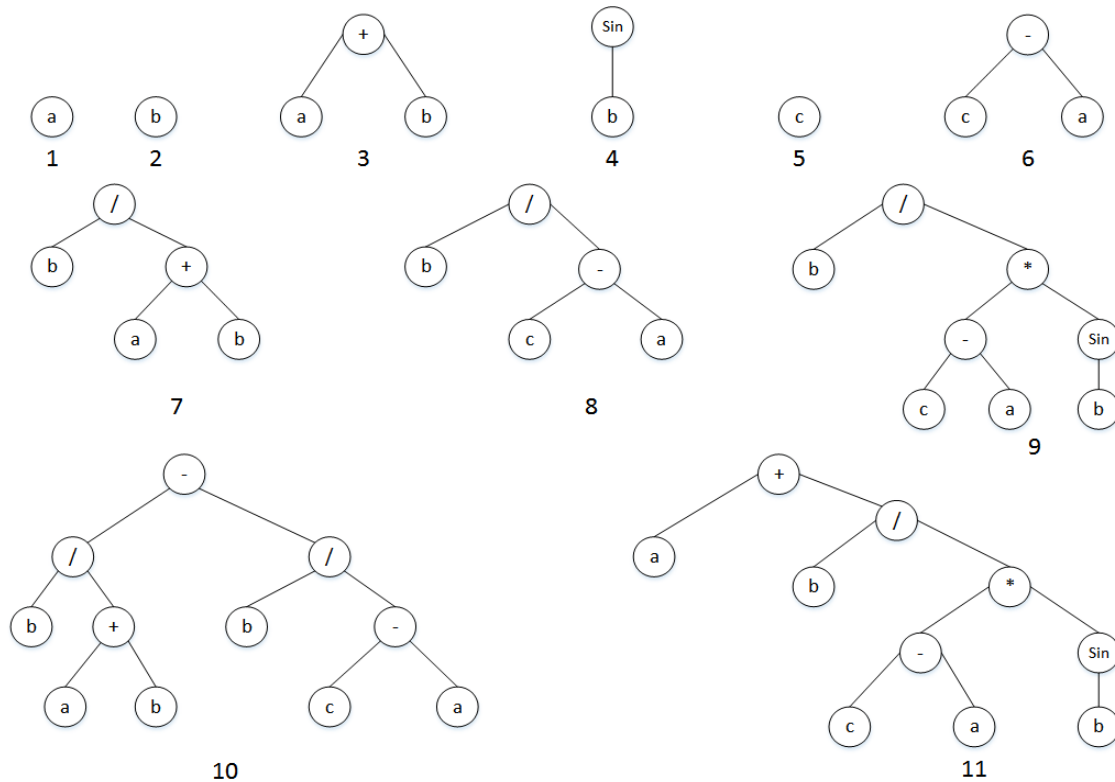


Figure 2.10: MEP chromosome genes represented as trees

2.25 Genetic Operators

This section provides a description of the various genetic operators and how they are applied within MEP. The operators preserve the structure of the chromosome and the offspring produced are syntactically correct expressions. Each of the parents are selected using the same selection methods described for standard genetic programming in section 2.7.

2.25.1 Crossover

During crossover two parents are selected and recombined. Crossover operations change gene material between selected parents. Three variants of crossover have been used in MEP implementations: one-point, two-point and uniform crossover.

One-point crossover randomly selects a crossover point and the parent chromosomes exchange the genes that appear after the crossover point. Figure 2.11 illustrates one-point crossover. The

crossover point is position 4. The genes from 4 onwards are exchanged between the parents to generate the offspring. The highlighted genes illustrate the changes between the parents.

Parents		Offspring	
P1	P2	O1	O2
1: b	1: a	1: b	1: a
2: * 1, 1	2: b	2: * 1, 1	2: b
3: + 2, 1	3: + 1, 2	3: + 2, 1	3: + 1, 2
4: a	4: c	4: c	4: a
5: * 3, 2	5: d	5: d	5: * 3, 2
6: a	6: + 4, 5	6: + 4, 5	6: a
7: - 1, 4	7: * 3, 6	7: * 3, 6	7: - 1, 4

Figure 2.11: MEP one-point crossover

Two-point crossover randomly selects 2 points and genetic material is exchanged between the chromosomes. Figure 2.12 illustrates two-point crossover with crossover points at positions 3 and 5. The genes between 3 and 5 inclusive are exchanged between the parents.

Parents		Offspring	
P1	P2	O1	O2
1: b	1: a	1: b	1: a
2: * 1, 1	2: b	2: * 1, 1	2: b
3: + 2, 1	3: + 1, 2	3: + 1, 2	3: + 2, 1
4: a	4: c	4: c	4: a
5: * 3, 2	5: d	5: d	5: * 3, 2
6: a	6: + 4, 5	6: a	6: + 4, 5
7: - 1, 4	7: * 3, 6	7: - 1, 4	7: * 3, 6

Figure 2.12: MEP two-point crossover

Uniform crossover randomly selects genes and exchanges these between the parents to generate offspring as illustrated in Figure 2.13

Parents	
P1	P2
1: b	1: a
2: * 1, 1	2: b
3: + 2, 1	3: + 1, 2
4: a	4: c
5: * 3, 2	5: d
6: a	6: + 4, 5
7: - 1, 4	7: * 3, 6

Offspring	
O1	O2
1: a	1: b
2: * 1, 1	2: b
3: + 2, 1	3: + 1, 2
4: c	4: a
5: * 3, 2	5: d
6: + 4, 5	6: a
7: - 1, 4	7: * 3, 6

Figure 2.13: MEP uniform crossover

2.25.2 Mutation

During mutation in order to maintain syntactically correct programs, the first gene must encode a terminal symbol. The mutation operator is applied to the genes within the chromosome. A random number of genes in the chromosome are selected for mutation. If a terminal gene is selected for mutation, it may be changed to another terminal symbol or to a randomly created function gene. If a function gene is selected for mutation, the gene may be mutated to a terminal symbol or the function arguments are altered to point to other genes.

Parent	Offspring
1: b	1: b
2: * 1, 1	2: * 1, 1
3: b	3: + 1, 2
4: * 2, 2	4: * 2, 2
5: b	5: b
6: + 3 , 5	6: + 1, 5
7: a	7: a

Figure 2.14: MEP Mutation

In the example (Figure 2.14) above gene 3 and 6 were randomly selected for mutation. Gene 3 changed from a terminal symbol to a function, with randomly created function arguments. The mutation operator was applied to gene 6 and one of the function arguments where altered. The function arguments mutated always point to lower positions than the function position.

2.26 Introns and Modularisation

Introns within MEP are the genes which are not used within the chromosome. These genes have no effect on the fitness of an individual. Introns are helpful if they reduce the destructive effects of genetic operators. Modularisation methods such as automatically defined functions can be used to reduce introns.

Automatically Defined Functions (ADF) have been implemented in a MEP context as reusable subroutines. An ADF in MEP maintains the same structure as a MEP chromosome and it is made up of a number of genes. The function symbols used for an ADF in MEP are the same as the ones used for standard MEP and the terminal symbols within ADFs are restricted to terminal symbols defined only for ADF's and hence terminal symbols of standard MEP chromosomes cannot be used with ADF structures [61].

2.27 Strengths and Weakness of MEP

Since a single MEP gene within a chromosome is closely related to a standard GP individual, MEP benefits from the same strengths standard GP possesses. Some strengths and shortcomings specific for MEP are discussed below.

2.27.1 Strengths

- Multiple expressions within the same individual can be used to represent the best solution as well as explore a bigger search space as compared to single expressions.

2.27.2 Weaknesses

- High computational effort since multiple expressions are encoded within a single individual.
- Duplicate expressions within an individual can be found.

2.28 Chapter Summary

This chapter described genetic programming and some of its variant's grammatical evolution and multi-expression programming. An overview of the genetic programming algorithm was provided. Aspects of the GP algorithm: representation, initial population generation, evaluation, selection methods, genetic operators, control methods and termination criteria were included in this chapter. Introns and bloat which increases the time taken for evaluation were discussed. The strengths and shortcoming of genetic programming were provided in this chapter. Grammatical evolution, a variant of genetic programming was introduced in this chapter. The generational control model in the context of grammatical evolution was described. Representation and the mapping process within grammatical evolution was discussed. A description of genetic operators applied within grammatical evolution was provided as well as the strengths and weaknesses of grammatical evolution were discussed in this

chapter. Multi-expression programming, another variant of genetic programming was introduced in this chapter. An overview of multi-expression programming using the steady state control model was provided in this chapter. The representation, initial population generation and evaluation of individuals in multi-expression programming were discussed in this chapter. Various genetic operators applied in multi-expression programming as well as the strengths and weaknesses of multi-expression programming were provided in this chapter. This chapter has provided a foundation of the different approaches that will be used throughout this thesis.

3 Network Intrusion Detection

3.1 Introduction

This chapter firstly introduces intrusion detection, Section 3.2 outlines the network intrusion detection process, network intrusion datasets are discussed in section 3.3, details of performance measures used in network intrusion detection are provided in section 3.4, previous work on network intrusion detection is discussed in section 3.6. Section 3.7 provides a summary of the chapter.

Since the inception of networks and the internet, computer security has become a fundamental aspect of ensuring information and access to information is kept as secure as possible. Computer security is the process of preventing and detecting unauthorised access to information. Computer security addresses three main aspects of any computer-related system: confidentiality, integrity and availability [72]. Various mechanisms such as computer-related attacks compromise computer security. These attacks or intrusions put the security of a system at risk. Either internal intruders or external intruders initiate intrusions. Internal intruders are entities with authorized permission to information but still wish to perform unauthorised activities and external intruders are entities without authorised permission but use various techniques to attempt to compromise the security of information [45]. Different mechanisms have been introduced to safeguard against intruders and unauthorised access to information. Some of the mechanisms include firewalls, access controls and encryption. These mechanisms have however failed to fully protect networks and information from the increasing evolutions of sophisticated intrusions. As a result, intrusion detection systems have become an essential technique to detect intrusions before they inflict widespread damage [89]. Intrusion detection is the process of monitoring computer systems or networks for signs of intrusions. Software which automates the intrusion detection process is referred to as Intrusion detections system (IDS) [78].

3.2 Network Intrusion Detection

Network intrusion detection is a classification task that separates normal behaviours of networks from attacks [65]. A typical network intrusion detection system (NIDS) should be able to correctly identify intrusions within a network as well as ensure it does not identify normal connections within the network as possible intrusions [43]. Low time performance and fault tolerance are some of the other desired characteristics of a NIDS. Time performance is the total time required by the NIDS to detect an intrusion [18]. Figure 3.1 provides an illustration of the NID process.

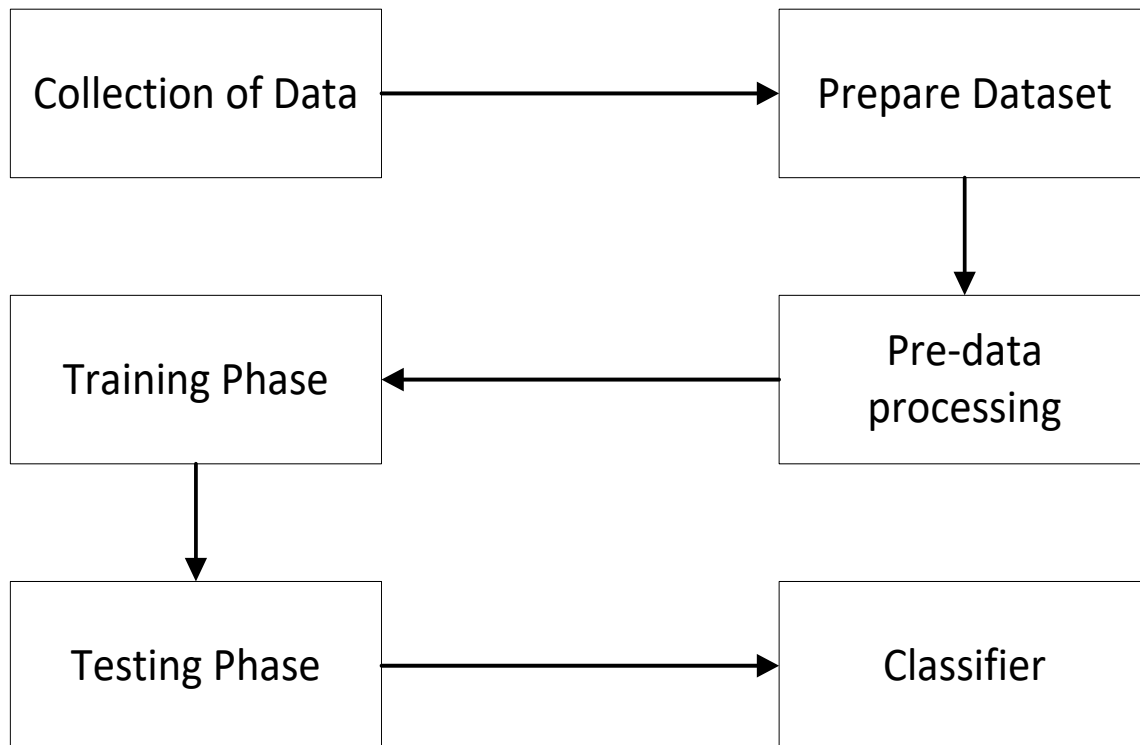


Figure 3.1: The NID process

Network information over the monitored network is extracted into a collection of connection records containing features such as protocol type, service, flag [10]. The collected records termed a dataset, is used to train and develop a classifier. Espejo *et al.* [22] defined a classifier as “a model encoding a set of criteria that allows a data instance to be assigned to a particular class depending on the value of certain variables”. Supervised learning is the approach frequently used to induce a classifier. Supervised learning involves using a dataset which has records that are labelled with their correct classes to induce a classifier which is capable of correctly classifying each record within the dataset [22]. In the context of NID, two types of classifiers exist, binary classifiers and multi-class classifiers. Binary classifiers distinguish between an attack and a non-attack and multi-class classifiers distinguish between different attack classes, i.e. types of attacks. Classification algorithms are

techniques that are used to induce classifiers. Examples of classification algorithms include genetic programming, genetic algorithms, neural networks and Bayesian networks are examples of classifiers.

Pre-processing of the dataset such as feature selection is performed if necessary before classification takes place. K-fold cross-validation and training and testing are some of the methods used to evaluate how well the classifier performs [7].

Training and testing involve splitting the dataset into two sets, the training set and the testing set. Different studies split the dataset differently. Studies in [6, 7] applied a 70/30 split where the training set consisted of 70% of the dataset and the testing set consist of the remaining 30%. Records that make up the training set or testing set are randomly selected. The training set is used by the classification algorithm for developing a classifier. Performance measures are used by the classification algorithms to assess the performance of the classifier. After the classifier generated, it is evaluated over the testing set to evaluate how well the classifier performs.

K-fold cross-validation involves splitting the dataset into k -parts of equal (approximate equal) size. The algorithm is run k -times and for each run, one k -part is used as the testing set whilst the other $k-1$ parts are used as the training set. If the dataset is not exactly divisible by k , the last k -part will contain fewer instances [6, 7].

3.3 Datasets for Network Intrusion Detection

The following section provides a description of the supervised machine learning datasets that have been widely used in network intrusion detection. The most commonly used datasets in NID are DARPA, KDD99 and NSL-KDD. Figure 3.2 provides the relation between the DARPA, KDD99 and NSL-KDD datasets.

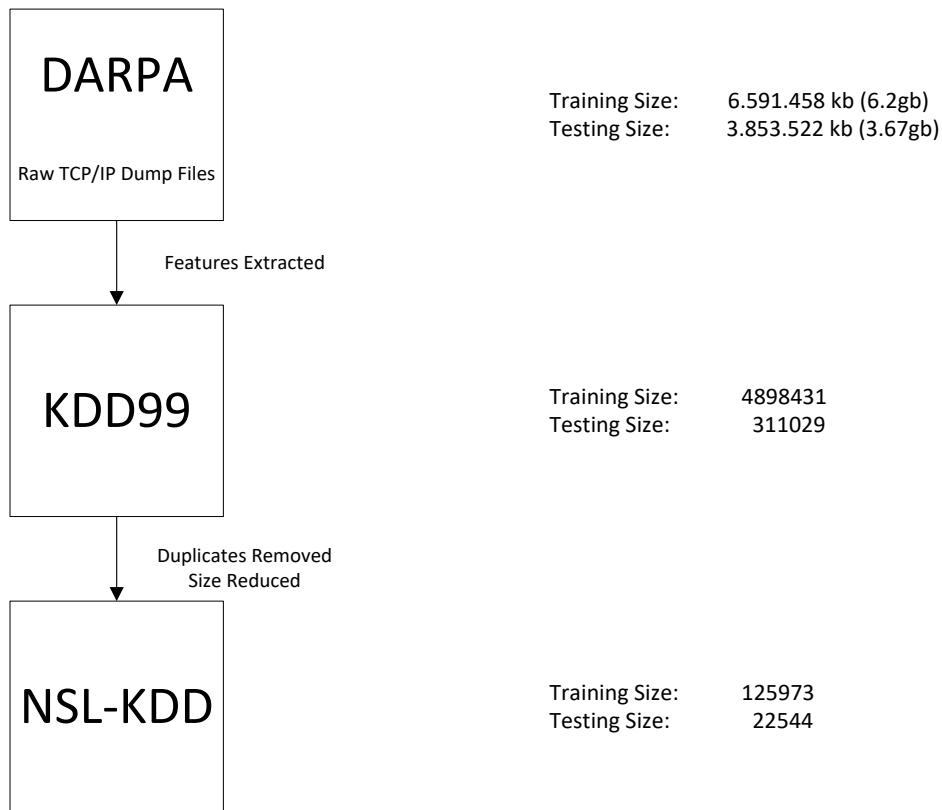


Figure 3.2: Relation between DARPA, KDD99 and NSL-KDD extracted from [65].

3.3.1 DARPA 1998 and 1999

The Defence Advanced Research Projects Agents (DARPA) and the Air Force Research Laboratory (AFRL) funded a project for the development of an evaluation dataset for NIDS. This resulted in the development of the DARPA 1998 dataset by the MIT Lincoln laboratory [16]. The data that was used to develop the dataset was extracted from a simulated network. The dataset is made up of training data collected over 5 days of a week from Monday to Friday over a period of 7 weeks and 2 weeks of test data of normal and intrusion user data. The dataset contains around 5 million connections with each connection approximately 100 bytes in size. Each connection is a sequence of TCP packets which flows under a specific protocol from a source IP address to a target IP address [16].

The dataset set was improved in 1999 to include Windows NT vulnerabilities and stealthier attacks resulting in the 1998 DARPA dataset. The 1999 dataset training set data was collected over 3 weeks and testing set data was collected over 2 weeks. The data in the datasets weeks one and weeks three consist of normal traffic and week two data consists of attacks. The network attack classes that were simulated in the DARPA set include Denial of service (DOS), Remote user to local (R2L), Local to Root (L2R), Probing and Anomalous behaviours [51]. These network attack classes are discussed in detail in section 3.3.4.

Some researchers [46, 50, 51] have criticized the dataset because the traffic generation software used is not publicly available and hence it is difficult to determine the accuracy of the background traffic data presented. They also questioned the use of simulated data as compared to using real-life systems and the use of a specific attacker increased the likelihood of bias in the data recorded.

3.3.2 KDD Cup 99

Feature extraction and data pre-processing techniques were performed on the DARPA'99 dataset to generate the KDD Cup 99. The KDD was prepared by Lee *et al.* [44]. The packet information within TCP dump files from DARPA was extracted into connections using Bro IDS [71], resulting in 41 features representing each connection [92]. The dataset was split into three labelled samples that are used for training and testing. The details of each sample are summarized in Table 3.1

	Attacks	Normal	Total
10% KDD	396743	97277	494020
Whole KDD	3925650	972781	4898431
Corrected KDD	250436	60591	311027

Table 3.1: KDD Cup 99 Sample Distribution

The features representing each connection are made up of 38 continuous or discrete numerical attributes and 3 categorical attributes. Each connection is labelled as either normal or a specific network attack [26]. The specific network attacks fall into one of the following categories; DOS, Probe, R2L and U2R, discussed in section 3.3.4. The dataset contains 24 network attack types in the training set and 38 attack types in the testing set. Among the 38 attack types in the testing set, 14 of them do not exist within the training set enabling the IDS to test how well it performs on unknown attacks. The dataset is heavily imbalanced towards attack connections. Out of 4 898 431 connections which make up the whole dataset, 3 925 650 connections are attack records [65]. Table 3.2 shows the attack distribution for the training set and testing set. U2R and R2L attack connections within the dataset are very few in comparison to the other network attacks.

Due to the huge size of the datasets, some researchers use smaller portions of the datasets. Some researchers have criticized the dataset for containing too many duplicate records within the training and testing set which has resulted in the creation of other datasets such as the NSL-KDD dataset [85].

	Training Size	Testing size
Normal	972781	60591
DOS	3883390	231455
Probe	41102	4166
U2R	52	245
R2L	1106	14570
Total	4898431	311027

Table 3.2: KDD Cup Class distribution

3.3.3 NSL-KDD dataset

The dataset was proposed by Tavallaee *et al.* [85] and consists of selected records of the KDD-CUP'99 dataset [1]. The NSL-KDD (NSL) dataset was created to solve some of the inherent problems of the KDD CUP (KDD) dataset. Tavallaee *et al.* [85] conducted an analysis of the KDD dataset and found problems within the KDD dataset. The KDD dataset suffered largely from a vast number of redundant records and from the results of the KDD analysis, about 78% and 75% of the records were duplicated in both the training and testing set, respectively. A large amount of redundancy within the KDD training set causes learning algorithms to be more partial towards the more frequent records as compared to infrequent records. Duplication within the testing set will cause the evaluation of learning algorithms to be more biased towards better detection rates of frequent records.

3.3.4 Network Attack Categories

Simulated attacks within the previously mentioned datasets fall into one of the following categories: Denial of Service (DOS), Probe, Remote to Local (R2L) and User to Root (U2R).

3.3.4.1 Denial of Service Attacks (DOS)

These are attacks where the attacker denies access to a machine by making the computing resources too busy to allow network requests placed by legitimate users. Different varieties of DOS attacks exist, some create malformed packets that confuse the system, whilst others take advantage of bugs located on particular networks. Smurf and Neptune are examples of applications that perform DOS attacks [38]. Distributed DOS (DDoS) attacks have also emerged which are a variant of DOS attacks but instead of using a single machine to perform this attack, multiple machines are used [43].

3.3.4.2 Probing attacks

When an attacker scans a machine or network in order to determine weaknesses that they might later exploit in order to compromise the system. Examples of probe attacks include portsweep and mscan.

3.3.4.3 Remote to Local (R2L) attacks

External intruders who have the ability to send and collect information from a host machine or network by exploiting the different vulnerabilities that exist on the network or host mainly initiate these attacks. Ftp_write, Guest and Xsnoop are some examples of R2L attacks which attempt to exploit the weak or misconfigured security policies within a network or host machine [38].

3.3.4.4 User to Root (U2R) attacks

The intruder initially starts using the system as a normal user and attempts to abuse the vulnerabilities of the system in order to gain higher privileges within the system. Perl and xterm are examples of user to root attacks [84].

Table 3.3 illustrates categorisation of the attacks that exist in the datasets discussed above.

Category	Network attack type
Denial of Service (DOS)	Back, Land, Neptune, Pod, Smurf, Teardrop, Mailbomb, Processtable, Udpstorm, Apache2, Worm, Syslogd
Probe	Ipsweep, Nmap, Portsweep, Satan, Mscan, Saint
Remote to Local (R2L)	Ftp_write, Imap, Multihop, Phf, Spy, Warezclient, Warezmaster, Guess_passwd, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named, Dictionary, Guest
User to Root (U2R)	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps, Eject, Ffbconfig, Fdformat

Table 3.3: Network intrusion detection categories and attack types [75]

3.4 Performance Measures

Performance measures are used to evaluate the efficiency and quality of a classifier. This section describes the performance measures commonly used for network intrusion detection.

3.4.1 Confusion matrix

The performance of a classifier is described using a confusion matrix. The matrix shows how frequently instances of a class x were correctly classified as class x or misclassified as some other class [7]. The confusion matrix illustrates performance for both binary and multi-class classifiers. The matrix illustrates the performance of the classifier using four measures:

- True Positive (TP) – The number of intrusion connections correctly classified as intrusions
- True Negative (TN) – The number of normal connections correctly classified as normal.
- False Negative (FN) – The number of intrusion connections incorrectly classified as normal connections.
- False Positive (FP) – The number of normal connections incorrectly classified as intrusion connections.

Table 3.4 illustrates a confusion matrix for two classes (intrusion and normal).

		Predicted connection label	
		Intrusion	Normal
Correct connection label	Intrusion	True Positive (TP)	False Negative (FN)
	Normal	False Positive (FP)	True Negative (TN)

Table 3.4: Binary confusion Matrix

The following sections describe performance measures which make use of information from the confusion matrix.

3.4.2 Accuracy and False Positive Rate

Accuracy is the proportion of the correctly classified connections amongst the total number of connections. Accuracy is calculated using the following formula [7]:

$$Accuracy (Acc) = \frac{TP + TN}{TP + FN + FP + TN} \quad (1)$$

False Positive Rate (FPR) measures the proportion of normal connections incorrectly classified over all the normal connections.

$$FPR = \frac{FP}{FP + TN} \quad (2)$$

3.4.3 Sensitivity and Specificity

Sensitivity is the proportion of true positive connections that were correctly classified [82]. It is also referred to as Recall or True Positive Rate (TPR). In NID, the metric measures how well the classifier detects intrusive connections. Sensitivity provides a more accurate measure of the intrusion detection effectiveness of the classifier as compared to the Accuracy.

$$TPR = \frac{TP}{TP + FN} \quad (3)$$

Specificity measures the proportion of true negative connections that were correctly classified within the dataset [82]. It is also referred to as True Negative Rate (TNR).

$$TNR = \frac{TN}{FP + TN} \quad (4)$$

3.4.4 Precision and F-measure

Precision measures how well the classifier correctly detects intrusive connections over all the positive connections returned by the classifier within the dataset. It is also referred to as the Positive Predictive Value (PPV) [7].

$$PPV = \frac{TP}{TP + FP} \quad (5)$$

F-Score is the weighted average of precision and recall. It is also referred to as the f-measure. It provides a compromise between recall and precision.

$$F_Score = 2 * \frac{Precision * TPR}{Precision + TPR} \quad (6)$$

3.4.5 Receiver operating characteristics

Receiver operating characteristics (ROC) graphs are two-dimensional graphs which provide a visual representation of classifier performance. The TP rates are plotted on the y axis and the FP rates are plotted on the x axis [24]. They are commonly used for binary classification problems and depict the tradeoffs between the benefits (true positive rates) and the costs (false negative rates). Figure 3.3 provides an example of a ROC graph.

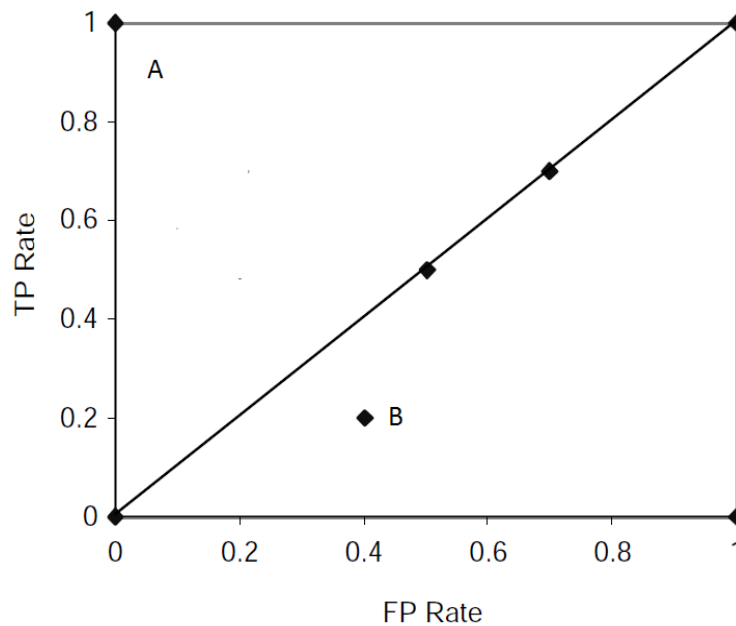


Figure 3.3: ROC graph

From the graph above, classifiers that perform well would be the ones found in the region labelled *A*, these classifiers correctly classify most of the instances. An ideal classifier would be one which generates the point (0.0, 1.0) meaning that all the instances are correctly classified. One classifier is better than the other if it is to the “north-west” of the others [7]. Classifiers found in the region labelled *B*, tend to have a high rate of false positives and are worse than random guessing which is depicted by the diagonal line running from point (0.0, 0.0) to point (1.0, 1.0) [7, 24].

3.5 Feature Selection

Feature selection or attribute selection is defined as “*the process of selecting a subset of original features according to a certain criteria*” [9]. Features within a dataset can be described as either irrelevant, redundant or relevant. Irrelevant features are features which have no effect on the accuracy of a generated NID classifier. Redundant features consist of features which can be used interchangeably and still have the same effect on the performance measure of the classifier. Relevant features are features which have a direct effect on the primary objective of a NID classifier [12]. Feature selection achieves two main goals, it selects high quality features which help ensure the classifier generated retains high accuracy rates and low false positive rates as well as minimize the computation required during the generation of classifiers [91]. Dash and Liu [17] provide a study on feature selection for classification as well as categorize feature selection based on the generation procedures and evaluation functions.

3.6 Previous Work on Network Intrusion Detection

The network intrusion detection domain has been widely researched. Different approaches have been used to generate efficient NID classifiers which can accurately allocate specific network connections to their appropriate classes. Some of the approaches that have been used are discussed below. These approaches were selected using two criteria; either the approach used the same dataset as the work presented in this thesis or produced state-of-the-art results.

3.6.1 Evolutionary Algorithms

Chittur [11] generated a binary classifier using a genetic algorithm (GA) in order to evaluate if a GA can produce classifiers with high accuracy rates. The KDD-99 dataset was used to evaluate the evolved classifiers. Chittur used a combination of the sensitivity and false positive rate as a measure of the fitness of an individual. The classifier generated performed well obtaining a high sensitivity rate and low false positive rate. The results also showed that genetic algorithms are able to produce classifiers with high accuracy rates.

Gong *et al.* [27] evolved production rules for network intrusion detection. The production rules were evolved using a GA. The DARPA dataset was used to evaluate the classifier. The dataset was made up of normal connections and two network attacks portsweep and pod. Feature selection was used to reduce the number of features. Crossover and mutation were used to produce the offspring. The classifier performed fairly well, obtaining a high detection rate for detection of normal connections.

Hoque *et al.* [35] generated a multi-class classifier using a genetic algorithm (GA) to efficiently detect various classes of intrusions. The classifier was generated using the KDD-99 dataset. False positive rate and sensitivity were used to evaluate the overall performance of the classifier. The classifier performed well obtaining a high sensitivity rate for detecting DOS attacks as well as obtaining a low false positive rate.

3.6.2 Neural Networks

Wang *et al.* [87] generated a classifier using artificial neural networks and fuzzy clustering for multi-class classification. The KDD-99 dataset was used to evaluate the performance of the classifier. Wang *et al.* compared the performance of the hybrid neural network to other approaches (decision trees, back propagation neural networks and naïve Bayes) based on three performance metrics: precision, sensitivity and f-measure. The hybrid neural network outperformed the other approaches in terms of the three performance metrics. However, it achieved a higher training time as compared to the other approaches.

Govindarajan and Chandrasekaran [28] applied a hybrid neural network to generate a classifier. The classifier generated was a hybrid designed using a radial basis function (RBF) and support vector machine (SVM). The NSL-KDD dataset was used to evaluate the performance of the classifier. The ensemble was evaluated using the classification accuracy. The study also generated two classifiers, one using RBF and another using SVM and compared the three classifiers for performance. The hybrid classifier outperformed the other two classifiers obtaining a high classification accuracy.

Ibrahim *et al.* [36] implemented a Self-Organization Map (SOM) artificial neural network for intrusion detection. The neural network distinguished between normal connections and the attack connections. The KDD-99 and the NSL-KDD datasets were used to evaluate the performance of the classifiers. The neural network performed well obtaining high accuracy rates for detecting intrusions in both the KDD-99 and NSL-KDD datasets. The neural network was also compared to other approaches [54, 66, 80] which used the same dataset and it outperformed all the approaches for binary classification.

3.6.3 Bayesian Networks

Panda *et al.* [66, 67] generated a naïve Bayes binary classifier for intrusion detection. Naïve Bayes classifiers using different data filtering configurations were generated and evaluated for performance. The NSL-KDD dataset was used to evaluate the classifiers. Cross-validation was used to train and test the classifier. Sensitivity and false positive rate were used to evaluate the performance of the classifier. The classifier using principal component analysis as the filtering approach achieved a higher sensitivity rate and lower false positive rate outperforming the other classifiers.

Mukherjee and Sharma [55] generated a naïve Bayes multi-class classifier for network intrusion detection. The NSL-KDD dataset was used to evaluate the evolved classifier. Feature selection was performed. Cross-validation was applied for testing the classifiers. The Waikato Environment for Knowledge Analysis (WEKA) [31], a toolkit which contains a collection of various algorithms and data processing tools was used for the experiments. Accuracy was used to evaluate the performance of the classifier. The classifier that was generated with 24 features outperformed the other classifiers generated.

3.6.4 Decision Trees

Thaseen and Kumar [86] evaluated the performance of several decision tree classification algorithms for generating binary classifiers. The classification algorithms include AD tree, C4.5, LAD tree, NB tree, random tree, random forest and REPTree. The classifiers generated were evaluated using the NSL-KDD dataset. The experiments for generating the classifiers were performed using WEKA. Feature selection

was performed. Accuracy was used to evaluate the performance of the classifiers generated. The classifier generated using a random tree classification algorithm obtained the highest accuracy rate over the other classifiers.

Chae *et al.* [9] generated a classifier using J48 for network intrusion detection. The NSL-KDD dataset was used to evaluate the classifiers. Ten-fold cross-validation was used to evaluate how the classifier performed. Accuracy was used to evaluate system performance. Several feature selection methods were compared and the feature selection method which used 22 out of the 41 features from the dataset obtained the highest accuracy rate.

3.7 Chapter Summary

This chapter introduced network intrusion detection and provided a description of datasets used for network intrusion detection. Network attack categories were discussed. Performance measures used in network intrusion detection were discussed in this chapter. Previous studies in network intrusion detection were discussed.

4 GP and Network Intrusion Detection

4.1 Introduction

This chapter reviews studies that have been conducted using genetic programming and its variants for network intrusion detection. Section 4.2 provides an overview of using genetic programming for network intrusion detection. Section 4.3 describes studies that have performed binary classification for network intrusion detection and section 4.4 reviews previous work where multi-class classification for network intrusion detection was performed. The strengths and weaknesses of applying genetic programming and its variants for network intrusion detection are provided in section 4.6. Section 4.5 provides an analysis of genetic programming and its variants for network intrusion detection. The chapter is summarized in section 4.7.

4.2 Using genetic programming for network intrusion detection

When using genetic programming (GP) for network intrusion detection (NID) the problem is treated as a classification problem. The classification problem is viewed as either a binary classification problem or a multi-class classification problem. Classification algorithms such as genetic programming is used to create a classifier to solve a classification problem. Binary classifiers are generated for binary classification problems and multi-class classifiers are generated for multi-class classification problems. Binary classifiers are generated to detect whether an intrusion exists or not and multi-class classifiers are generated to detect different types of intrusions. A classifier can be represented in the form of a rule. Genetic programming is used to evolve a rule and the rule can be in the form of either a logical tree or an arithmetic tree or a decision tree or a production rule. Logical trees are made up of logical operators such as the OR operator. Arithmetic trees represent mathematical expressions and the function set consists of mathematical operators such as +, -, *, / [20]. Decision trees represent the features as nodes and the leaf nodes as the classes. Production rules are rules which are used to represent classifiers using IF-THEN statements.

When using GP for multi-class classification problems different methods are used to generate multi-class classifiers. Binary decomposition, static range selection and dynamic range selection are some of the methods used by GP for performing multi-class classification.

Binary decomposition [47] decomposes a problem with n -classes into a number of binary classification problems. Given a problem with 4 classes (1, 2, 3, 4) and each binary problem having two classes a and b , the binary decomposition is performed as follows:

- In the first problem, $a = (1)$ and $b = (2, 3, 4)$.
- In the second problem, $a = (2)$ and $b = (1, 3, 4)$.
- In the third problem, $a = (3)$ and $b = (1, 2, 4)$.
- In the fourth problem, $a = (4)$ and $b = (1, 2, 3)$.

When static range selection [47] is used to solve a multi-class problem, each class is defined by specific boundary regions. Boundary regions are defined based on the problem domain and possible class boundary points. Boundary regions are defined before a GP run. During the GP run, the classifier is considered to have classified a connection to a specific class if the output falls within a specific boundary region. Given a problem with four classes, the following boundary regions can be used to represent the classes: Class 1 = $[-\infty, -1]$, Class 2 = $[-1, 0]$, Class 3 = $[0, 1]$, Class 4 = $[1, \infty]$. In static range selection, each element in the population has the same boundary regions. Dynamic range selection is an alternate approach to static range selection, which dynamically allows each element in the population to use a different set of class boundary regions.

4.3 Binary Classification for NID using GP

This section reviews studies which make use of genetic programming and its variants to evolve binary classifiers for NID.

4.3.1 Genetic Programming

This section focuses on reviewed studies which evolve binary classifiers using standard genetic programming for NID.

Crosbie and Spafford [15] improved manually created rules using genetic programming. The manually created rules were encoded in the initial population. Crosbie and Spafford's work was one of the first implementations that applied genetic programming for intrusion detection. The authors used their own dataset for evaluation. The individuals in the population represented production rules. Each individual was comprised of arithmetic operators, conditional operators, logical operators and features of the dataset. The fitness function was the number of correctly classified intrusions and non-intrusions. Crossover was applied during the evolutionary process. Training and testing were used to

evaluate the overall performance of the classifier. The classifier designed did not perform well but the work provided the foundation for applying genetic programming for intrusion detection.

Lu and Traore [48] used genetic programming to improve manually created production rules. The individuals in the population represented production rules and the initial population was made up of the manually created production rules. The DARPA 1998 dataset was used to evaluate the performance of the classifier. False positive rate (FPR), false negative rate (FNR) and unknown attack detection rate (UADR) were used as fitness functions to evaluate the rules generated. UADR measured the rate of detecting unknown attacks. The genetic operators used were mutation, reproduction and crossover. The overall performance of the classifiers was measured using accuracy and FPR. The classifier performed fairly well achieving an average accuracy rate and FPR of 0.57% and 0.041% respectively over the 10000 runs that were performed.

Yin *et al.* [90] applied genetic programming to generate a rule-based system to detect intrusions. The DARPA 1999 dataset was used for the experiments. The Learning Rules for Anomaly Detection (LERAD) presented by Mahoney *et al.* [50] were used to create the initial rules used by genetic programming during initial population generation. The individuals in the population represented production rules. The accuracy was used as the fitness function. Crossover and mutation were applied to improve existing rules. The overall performance of the classifiers was measured using the number of correctly classified intrusions. The classifier designed by Yin *et al.* outperformed other classifiers detecting 84 out of the 148 intrusions obtaining a 54% accuracy.

Orfila *et al.* [64] designed an intrusion detection classifier using genetic programming. The classifier was compared with the machine learning algorithm C4.5 in terms of efficiency and effectiveness. The Lawrence Berkeley National Laboratory (LBNL) Dataset [68] was used to evaluate and compare the performance of the classifiers. The individuals in the population represented logical trees. Logical and bitwise operators formed the function set whilst the terminal set was comprised of the features of the dataset as well as an ephemeral random constant. The fitness of a classifier was calculated as the difference between sensitivity and false positive rate multiplied by the accuracy. Crossover was applied during the evolutionary process. The authors evaluated the classifiers using cross-validation. GP generated more effective rules than C4.5 rules. GP also had simpler solutions with fewer nodes as compared to the best individual derived by C4.5.

Blasco *et al.* [5] evaluated the performance of classifiers generated using different fitness functions. The KDD-99 dataset and a modified version of the KDD-99 were used for the experiments. The individuals represented logical trees. The terminal set was composed of the features of the dataset as

well as two ephemeral random constants. Two fitness functions were considered for evaluation. The first fitness function was the difference between sensitivity and false positive rate whilst the second fitness function was a function of the sensitivity, false positive rate as well as the frequency of attacks within the dataset. Tournament selection was used as the selection method. Genetic operators crossover and mutation were applied during the evolution process. The overall performance of the classifiers was measured based on the accuracy. The classifier generated based on the function of sensitivity, FPR and frequency of attacks achieved a higher accuracy rate. The classifiers were also compared to other classifiers [30] evaluated on the KDD dataset and they outperformed 3 out of the 6 classifiers compared.

Pastrana *et al.* [69, 70] applied genetic programming to generate intrusion detection classifiers. The Lawrence Berkeley National Laboratory (LBNL) and the KDD-99 dataset were used to evaluate the classifiers. The individuals represented logical trees. The features of the datasets formed the terminal set. Tournament selection was applied as the selection method. Crossover and mutation were applied during the evolutionary process. Cross-validation was used for the experiments that involved the LBNL. Training and testing were used for the KDD-99 experiments. FPR was used to measure the overall performance of the classifiers. The LBNL and KDD-99 NIDS classifiers performed well obtaining an FPR of 4% and 3% respectively. The LBNL classifier was also compared to the classifiers generated using C4.5 and naïve Bayes. The LBNL classifier outperformed the other two classifiers obtaining a lower FPR.

4.3.2 Grammatical Evolution

The study reviewed in this section used grammatical evolution to generate binary classifiers for network intrusion detection.

Sen and Clark [79] applied grammatical evolution to mobile ad hoc networks (MANETs). The objective of the study was to model a classifier that could detect intrusions on MANETs. The authors used their own dataset. The BNF production rules of the grammar were composed of a combination of arithmetic operators, binary operators and MANET's features. Individuals were represented as production rules. The difference between the accuracy and the false positive rate was used as the fitness function. Mutation and crossover were applied during the evolutionary process. The training and testing method was used. The overall performance of the classifiers was measured based on the accuracy. The classifier designed achieved high accuracy rates.

4.3.3 Linear Genetic Programming

The work discussed in this section applied linear genetic programming to generate classifiers for network intrusion detection.

Song *et al.* [83] applied linear genetic programming for network intrusion detection. The KDD-99 dataset was used to evaluate the performance of the classifier. Linear genetic programming was used to represent individuals. The individuals were defined in terms of the number of pages and instructions. Three fitness functions were considered for evaluation: equal class cost, variable class cost and hierarchical cost. The equal class cost fitness function measured the number of correctly classified intrusions and non-intrusions. The variable class cost fitness function favoured infrequent classes by assigning a higher weighting to infrequent class detection as compared to frequent classes. The hierarchical cost fitness function measured the number of misclassified connections. Tournament selection, crossover and mutation were applied during evolution. The overall performance of the classifiers was measured based on the accuracy and the false positive rate. Individuals using hierarchical cost as the fitness function obtained the best results for both the false positive rate and accuracy.

4.4 Multiclass Classification for NID using GP

This section reviews work which generated multi-class classifiers for network intrusion detection using genetic programming and its variants.

4.4.1 Genetic Programming

This sections reviews studies which applied standard genetic programming to generate multi-class classifiers for network intrusion detection.

Faraoun *et al.* [23] applied genetic programming to perform multi-class classification for network intrusion detection. The KDD-99 dataset was used to evaluate the performance of the classifier. The individuals represented arithmetic trees. The terminal set was comprised of the features from the dataset and constants. The function set contained arithmetic operators. Fitness proportionate selection was used as the selection method. Genetic operators crossover and mutation were applied during the evolutionary process. The training and testing method was used. The overall performance of the classifier was measured based on accuracy and the FPR. The classifier obtained high accuracy rates and low false positive rates. The classifier also outperformed other classifiers [21, 44] generated using the same dataset.

4.4.2 Grammatical Evolution

This section reviews work which has applied grammatical evolution for generating multi-class classifiers for network intrusion detection.

Wilson and Kaul [88] applied grammatical evolution for automating the generation of intrusion detection rules. The KDD-99 dataset was used to evaluate the performance of the classifier. Rank selection was used as the selection method. The fitness of the rules was based on a combination of the features and the specific output class for each rule. Crossover and mutation were applied during the evolutionary process. The classification accuracy was used to evaluate the performance of the classifier in detecting the different network attacks. The best classifier evolved achieved a high classification accuracy for detecting attacks such as DOS but also achieved very low detection rates for other attacks such as R2L attacks.

4.4.3 Multi-expression Programming

This section reviews a study which applied multi-expression programming for generating multi-class classifiers for network intrusion detection.

Grosan *et al.* [29] applied multi-expression programming (MEP) for multi-class classification and compared the performance of the classifier generated to linear genetic programming (LGP), support vector machines (SVM) and decision trees (DT). The 1998 DARPA dataset was used for classifier evaluation. LGP applied tournament selection, crossover and mutation during evolution. The MEP individuals represented arithmetic trees. The MEP function set was comprised of a combination of arithmetic and logical operators. The terminal set comprised the 41 features from the dataset. Crossover was applied to generate offspring and accuracy was used as the fitness for MEP. The training and testing method was used for the experiments and accuracy was used to evaluate the performance of the classifiers. MEP outperformed LGP, SVM and DT classifiers for detecting normal, U2R and R2L attacks and LGP outperformed the rest of the classifiers in accurately detecting DOS and Probe attacks.

4.4.4 Linear genetic programming

Mukkamala *et al.* [56] investigated the use of linear genetic programming for modelling intrusion detection systems. The performance of linear genetic programming (LGP) was compared to support vector machines (SVM) and a neural network trained using resilient backpropagation (RBP) learning models in terms of scalability, the time it took to train and test the approaches and detection accuracy. The DARPA 1998 dataset was used for evaluation. The crossover operator was used to exchange sequences of instructions between two tournament winners. For each network attack category, an LGP classifier was evolved. LGP outperformed SVMs and RBP in terms of detection accuracies for each of the network attack type categories.

Hansen *et al.* [32] applied GP to evolve a classifier for network intrusion detection. The authors performed a comparative evaluation of two classifiers: one using standard crossover and the other using the homologous crossover operator, in order to determine the best crossover operator to use for intrusion detection. The KDD-99 dataset was used to evaluate the classifiers. Mutation and crossover were applied. The GP classifier using homologous crossover performed better than the standard crossover classifier. Hansen *et al.* also compared the state-of-the-art approaches with the GP classifier using homologous crossover and the classifier performed better than the state-of-the-art classifiers for DOS, Probe, U2R and R2L attacks.

4.5 Strengths and Weaknesses of GP in NID

This section highlights the strengths and weaknesses of using genetic programming and its variants for network intrusion detection.

4.5.1 Strengths

- Automatic feature selection

During the evolution process, the GP run indirectly performs the process of feature selection by selecting the best features to use in representing the solution to the problem. This eliminates additional feature selection tasks usually performed.

- Flexibility

Different representations can be used to represent classifiers for network intrusion detection. Quality of classifiers generated can also be improved by using modified fitness functions and genetic operators.

4.5.2 Weaknesses

- Introns

Some classifiers generated by GP become very large because of redundant code within the classifier.

4.6 Analysis of genetic programming in network intrusion detection

From the studies discussed above genetic programming has been widely used to generate classifiers for NID. There have been studies which have applied linear genetic programming and from the studies, linear genetic programming has been able to achieve high accuracy rates. From the literature, it can be seen that a lot of research effort has gone into linear genetic programming which has been able to attain high accuracy rates. The work in MEP and GE is still in its initial stages and from the studies reviewed, the two approaches tend to show promise of generating classifiers which can achieve high

accuracy rates. For the work presented in this thesis, MEP and GE will be investigated for generating classifiers for network intrusion detection. Genetic programming will be used to generate classifiers to provide a baseline to compare the performance of the classifiers generated by GE and MEP. Binary and multi-class classifiers will be generated using each of the approaches.

From the studies which generated binary classifiers, the accuracy rate has been successfully used as the fitness function in order to generate classifiers with high overall accuracies [5, 64, 79, 90]. For this study, the accuracy rate will be used as the fitness function for binary classifiers generated using the three approaches (GP, GE and MEP). Different fitness functions have been used for generating multi-class classifiers for NID. Investigations into the effects of different fitness functions have not been conducted and for the study in this thesis, investigations into the use of different fitness functions will be conducted in order to determine the effects of fitness functions on the overall performance of classifiers generated. Based on work done by Loveard *et al.* [47], dynamic range selection and binary decomposition performed better than other approaches for solving classification problems and for the study in this thesis, binary decomposition will be used to generate multi-class classifiers for NID.

In the literature reviewed genetic programming has been widely applied for network intrusion detection. Individuals in genetic programming have been represented using arithmetic trees [23], production rules [15, 48, 90] and logical trees [5, 64, 69, 70]. Logical trees have been applied more frequently and have achieved higher accuracy rates. For this reason, logical trees will be used to represent individuals in this study. Tournament selection which has been applied in a number of studies [5, 29, 69, 70, 83] which have achieved high accuracy rates will be used in this study. Crossover and mutation have been widely used in the studies reviewed in this chapter. In this study, both genetic operators will be applied during the evolutionary process.

Grammatical evolution has been applied in a few studies for network intrusion detection [79, 88]. From the reviewed studies on grammatical evolution, production rules have been used to represent individuals. Production rules have not been able to achieve a high accuracy rate and for this study, logical trees will be used to represent individuals in order to investigate their potential for generating classifiers with better accuracy rates. Studies which have used tournament selection, crossover and mutation have achieved high accuracy rates and for the study in this thesis, tournament selection, crossover and mutation will be used.

Multi-expression programming has been applied once for network intrusion detection [29]. For this study, using logical trees to represent the individuals will be investigated. Genetic operators crossover and mutation will be applied during the evolutionary process for this study.

Different datasets have been used for network intrusion detection. The KDD-99 dataset has been widely used in the literature provided in this chapter compared to the other datasets used for NID. The KDD-99 dataset has however been criticized for being outdated and containing redundant records [50, 77, 85]. The NSL-KDD [85] dataset was proposed to overcome some of the limitations of the KDD-99 dataset possessed. In order to overcome the limitations of the KDD-99 dataset the following processes were performed on the NSL-KDD:

- Removal of redundant and duplicate records in both the training and testing sets.
- Reduction in the number of records in the training and testing sets, eliminating the need to randomly select small portions of the data set.

The NSL-KDD dataset offers a streamlined version of the KDD-99 dataset and for the work presented in this thesis, the NSL-KDD dataset will be used. The accuracy has been widely used in reviewed studies [5, 29, 48, 79, 83, 88, 90] to evaluate the overall performance of the classifiers generated. For this study, the accuracy will also be used to evaluate classifier performance.

4.7 Chapter Summary

This chapter discussed genetic programming and its variants for network intrusion. An overview of the application of genetic programming for network intrusion detection (NID) was discussed. Previous studies which generated binary classifiers for NID using genetic programming and its variants were discussed followed by studies which generated multi-class classifiers for NID. An analysis of previous work which used genetic programming and its variants for generating classifiers for NID was discussed and the strengths and weaknesses of using genetic programming for NID were provided.

5 Methodology

5.1 Introduction

This chapter describes the methodology applied in order to achieve the aims and objectives of the study presented in the thesis. Section 5.2 discusses the aims and objectives of this study. Section 5.3 and section 5.4 discuss the methodologies used to achieve the aims and objectives. Section 5.5 discusses the details of the dataset including the pre-processing methods applied to the dataset. Section 5.6 describes the distributed architecture used for the proposed approaches. Section 5.7 provides the technical specifications and section 5.8 summaries the chapter.

5.2 Research Methodology

Different research methodologies have been used in the field of computer science. Johnson [37] suggests four methods namely empiricism, mathematical proof approaches, hermeneutics (formal proof techniques) and proof by demonstration which have been used in computer science. Empiricism is used to determine the hypothesis validity. it follows a sequence of steps hypothesis, methods and results and conclusion. Mathematical proof approaches use formal proofs to reason about the validity of a hypothesis given some evidence. It can either be by verification; attempts to establish that some good property will hold in a given system, or by refutation. If a model is created and tested in the environment which it is intended to represent, the research methodology is regarded as hermeneutics. Proof by demonstration involves designing a system and iteratively refining the system based on feedback provided after each iteration cycle until the desired output is achieved or no further changes can be made to the system.

The following section outlines the aims and objectives of the work presented in this thesis as well as how each of the objectives will be achieved using the methodology discussed in this chapter.

5.2.1 Aims and Objectives

The main aim of this study is to evaluate the different variants of genetic programming, namely grammatical evolution and multi-expression programming for evolving network intrusion detection

classifiers. Proof by demonstration will be used in order to achieve the aims of this study. The following outlines the objectives of the study presented in this thesis.

- Objective 1: Detecting intrusions using grammatical evolution (GE)
- Objective 2: Classifying network attack types using GE
- Objective 3: Detecting intrusions using multi-expression programming (MEP)
- Objective 4: Classifying network attack types using MEP
- Objective 5: Detecting intrusions using genetic programming (GP)
- Objective 6: Classifying network attack types using GP
- Objective 7: Comparative analysis of GE, MEP and GP for network intrusion detection

Two methodologies will be used in order to achieve the objectives outlined above. Objectives 1 to 6 will be fulfilled using the methodology discussed in section 5.3 (methodology one) and objective seven will be fulfilled using the methodology discussed in section 5.4 (methodology two).

5.3 Proof by Demonstration Methodology

Proof by demonstration will be used as the methodology. An initial approach will be implemented based on the critical analysis of the literature discussed in section 4.6. The approach will be iteratively refined if the approach is not performing well enough when compared to previous literature until a termination criterion has been met. During iterative refinement, the approach will be evaluated by

Proof by demonstration

Begin

- Implement initial approach
- Repeat
 - Evaluate the approach.
 - Refine approach (if necessary).
- Until a termination criterion is met.

End

Return implemented approach.

Algorithm 5.1: Proof by demonstration

testing the performance of the implementation on the dataset. If the implementation is not performing well based on the evaluation, the implementation is refined until the desired output is achieved. The algorithm for proof by demonstration is outlined in Algorithm 5.1 and the following sections discuss how each of the steps in the algorithm will be achieved.

5.3.1 Evaluation of approach

Training and testing will be used to evaluate the performance of the approach. The approach will be evaluated on the datasets described in section 5.5. Due to the stochastic nature of genetic programming and its variants, multiple runs of the proposed approaches will be performed. Thirty runs will be performed on the training dataset. A random seed will be used for each of the runs performed. The classifier with the highest fitness from the 30 runs will be considered the best classifier. The best classifier will be run on the test dataset in order to evaluate the overall performance of the approach. Accuracy will be used to evaluate how well the approach performs.

5.3.2 Refinement of approach

During the refinement of the approach different aspects of the approach will be changed in order to improve the performance of the approach. Aspects of the approach that will be changed include:

- Parameter values

A combination of different parameter values affects the performance of the approach. Parameters values such as the maximum tree depth of an individual have an overall effect on the generation of efficient classifiers and the population size which controls the number of individuals created has an effect on the chances of the approach finding a global optimum classifier for network intrusion detection. Changes in parameter values have to be performed in order to increase the performance of the implementation. Parameters that will be changed include the population size, application rates of the genetic operators, the maximum number of generations, maximum tree depth, BNF grammar and the number of genes in a MEP chromosome.

- Representation of individuals

Different representations have different effects for the generation of classifiers. Changes in the representation of individuals may lead to the generation of efficient classifiers. Logical trees will be tested in order to evaluate the best representation.

- Fitness Function

Fitness functions have an effect on the overall performance of implementations. Different fitness functions will direct the approach to different areas of the search space.

- Selection Method

The choice of selection method has an effect on the overall performance of an approach. Tournament selection has been widely used in the literature. Selection pressure has an effect on tournament selection. Selection pressure, which is the degree to which high accuracy classifiers are favoured has an effect on the convergence rate of an approach.

- Genetic Operators

Genetic operators generate offspring of different shapes and sizes. Different genetic operators are used to explore different areas of the search space and changes in the genetic operators applied may lead to the generation of high accuracy classifiers.

5.3.3 Termination Criterion

The termination criteria of the methodology will be met when one of the following is achieved:

- No further improvements in the performance of the classifiers were achieved.
- Performance of the classifier is better than existing implementations from literature.

5.4 Statistical Tests

The statistical significance of each of the approaches will be evaluated. More than 30 runs will be performed for each experiment using the previous methodology in order to obtain a normal distribution which will be used to perform statistical tests.

5.4.1 Statistical Testing

Hypothesis tests will be used to test for the significance of the results when comparing the different variations of genetic programming to evolve classifiers. A one-tailed hypothesis test (Z-test) will be used for the work presented in this thesis, to determine the statistical significance of results obtained when comparing different approaches. Table 5.1 provides the level of significance, critical values and decision rules for the Z-test. Assume two classification approaches, A and B are being compared, with means μ_A and μ_B respectively. In order to apply the Z-test, the first step is to formulate the null hypothesis and the alternative hypothesis as follows:

$$H_0 : \mu_A = \mu_B$$

$$H_a : \mu_A > \mu_B$$

The value of Z is calculated and compared to the critical value. If the Z-value is less than the critical value, there is no statistical significance in comparing the means resulting in the null hypothesis (H_0) being accepted. If the Z-value is greater than the critical value, there is a statistical difference between the two means resulting in the alternate hypothesis being accepted (H_a).

Significance (α)	Critical Value	Decision Rule
0.01	2.33	Accept H_0 – if $Z < 2.33$
0.05	1.64	Accept H_0 – if $Z < 1.64$
0.1	1.28	Accept H_0 – if $Z < 1.28$

Table 5.1: Z-hypothesis test table showing levels of significance, critical values and decision rules

5.5 Dataset

This section describes the datasets that will be used for the work presented in this thesis. This section will also provide the pre-processing steps that will be applied to the datasets.

5.5.1 Dataset description

Datasets which have been used in network intrusion detection have been discussed in section 3.3. The NSL-KDD dataset will be used for the work presented in this study based on the critical analysis of the previous work discussed in section 4.6.

The KDD Train+ set from the NSL-KDD dataset will be used for training and modelling the classifiers. The KDD Test+ set will be used to test the performance and effectiveness of the modelled classifiers. Table 5.2 provides the distribution of network attack categories and the total number of connections for each sample.

	Normal	U2R	DOS	R2L	Probe	Total
KDD Train+	67343	52	45927	995	11656	125973
KDD Test+	9711	67	7460	2885	2421	22544

Table 5.2: NSL-KDD sample distribution

5.5.2 Dataset Pre-processing

Data pre-processing transform data into a format that will be easier and more effective to use for the purpose of the study. The NSL-KDD contains a combination of nominal and numeric values in the 41 features that make up the dataset. Data preprocessing will be applied to the dataset in order to refine the dataset and make it easier to use for this study. Data transformation and normalization will be performed on the dataset based on previous work [4, 28, 36].

5.5.2.1 Data transformation

Data transformation involves converting all the nominal feature values to distinct numerical values. For each of the nominal features (Protocol_type, Service and Flag), the distinct elements for each feature will be mapped to distinct numerical values [4, 36]. Table 5.3 provides the feature name and the mapped numerical value.

Feature	Feature Name	Numerical Value
Protocol_Type	TCP	0
	UDP	1
	ICMP	2

Flag	S0	0
	S1	1
	S2	2
	S3	3
	SF	4
	SH	5
	OTH	6
	REJ	7
	RSTR	8
	RSTO	9
	RSTOSO	10
Service	All services	0 to 69

Table 5.3: Data transformation

5.5.2.2 Dataset normalization

Connections within the dataset contain varying ranges. The numerical values will be scaled using the min-max method of normalisation [36]. Each attribute will be scaled to fall within the range [0, 1], which is consistent with reviewed studies that have used the NSL-KDD dataset [4, 28, 36].

$$X_{new} = \frac{X - X_{Min}}{X_{Max} - X_{Min}} \quad (7)$$

5.5.3 Binary classification dataset

The NSL-KDD dataset was prepared for binary classification by setting all the intrusive connections as one class and the normal connections as another class. The KDD Train+ subset of the NSL-KDD will be used as the training set and the KDD Test+ will be used as the testing set. Figure 5.1 provides the distributions of connections within the binary dataset that will be used for the study presented in this thesis.

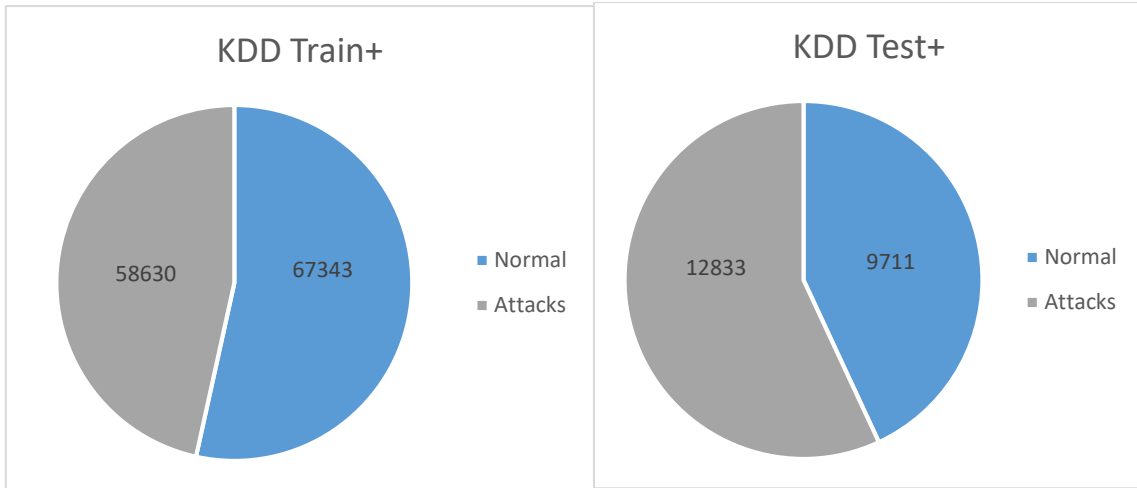


Figure 5.1: Binary classification distribution

5.5.4 Multi-class classification dataset

The attack class for each connection in the dataset which will be used for the experiments presented in this thesis were converted from the specific network attack to the network attack category presented in Table 3.3. The KDD Train+ subset of the NSL-KDD will be used as the training set and the KDD Test+ will be used as the testing set. Figure 5.2 provides the distribution of attacks within the training and testing sets of the NSL-KDD dataset.

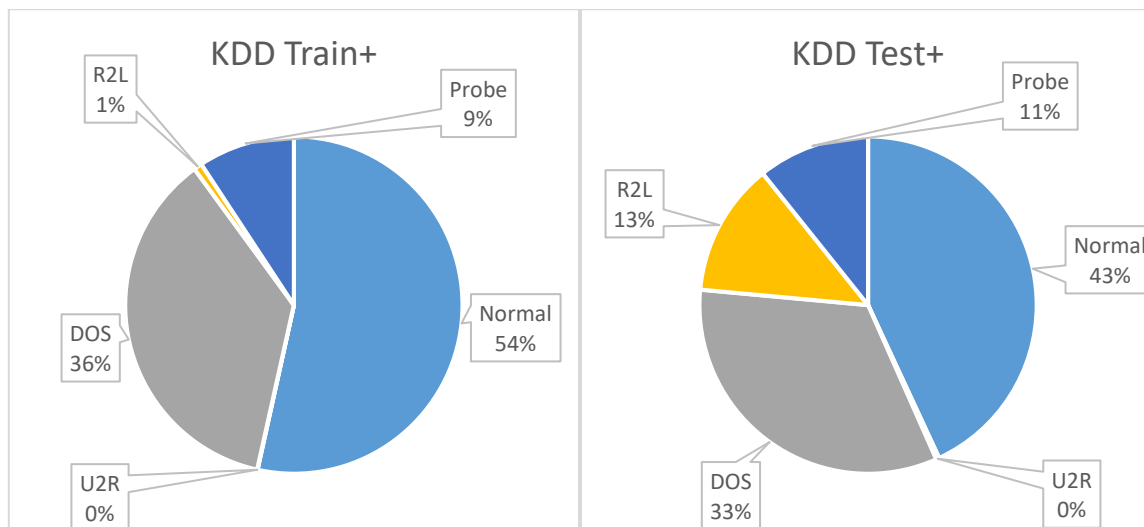


Figure 5.2: Multi-class Classification distribution

5.6 Distributed Architecture for Proposed Approaches

Genetic programming has high runtimes when evolving classifiers and in order to reduce the high runtimes associated with GP, a multicore architecture will be applied. The multicore architecture will be applied during the training phase of the proposed approaches described in this thesis. Two

processes, initial population generation and creation of the new population, are distributed because that is where mainly high runtimes are experienced.

The process of creating the initial population and evaluation of the population is distributed over the architecture by dividing the number of individuals created over the number of cores. If n individuals are to be generated initially and t cores are available, each core will generate $\frac{n}{t}$ individuals in parallel. For instance, if the population size is 200 and 8 cores are allocated, each of the 8 cores will generate 25 individuals in parallel.

Similarly, the creation of the new population for each generation (regeneration) is distributed over the architecture by dividing the number of individuals created by each of genetic operators over the number of cores. For instance, if two genetic operators are used with 8 cores and each genetic operator has an application rate of 50% and a population size of 200, 25 individuals will be generated on each of the 8 cores.

5.7 Technical Specifications

The algorithms proposed in this dissertation were written in Java 1.8 using NetBeans 8.1. The technical specifications of the computer used to develop the proposed algorithms were as follows: Intel(R) Core (TM) i7-3770S Quad Core @ 3.10GHz with 8GB RAM running 64bit Windows 7. Statistical tests were performed using Microsoft Excel 2016. The simulations were performed on the Centre for High-Performance Computing Lenggau cluster.

5.8 Chapter Summary

This chapter discussed the methodology used to achieve the aims and objectives outlined in chapter 1. Methodologies that will be used to achieve the objectives have been discussed. The dataset that will be used to evaluate the performance was described as well as dataset pre-processing which will be performed. This chapter concludes by providing a description of the distributed architecture for the proposed approaches and the technical specifications.

6 Genetic Programming for Network Intrusion Detection

6.1 Introduction

This chapter discusses the proposed genetic programming approach for network intrusion detection. Section 6.2 provides an overview of the algorithm used for the proposed approach. The initial population generation is discussed in section 6.3. The evaluation process of an individual is detailed in section 6.4. Section 6.5 discusses the selection methods and genetic operators. The parameters used are presented in section 6.6 and section 6.7 summarizes the chapter.

6.2 GP Algorithm

The generational control model is used for the genetic programming approach presented in this chapter. An initial population is created and evaluated. The parent(s) used for genetic operators are selected using tournament selection. The genetic operators generate offspring. The offspring is evaluated and forms part of the new population. The algorithm repeats until the maximum number of generations is reached. Algorithm 2.1 provides an overview of the algorithm. The best classifier returned is evaluated on the testing set using accuracy in order to evaluate how well the classifier performs.

GP Algorithm

Begin

- Create an initial population

- Repeat
 - Evaluate individuals in the population.
 - Select parents using tournament selection.
 - Apply genetic operators to selected parents.
 - Insert offspring into new population.

- Until a maximum number of generations.

End

Return the best individual and evaluate on the testing set.

Algorithm 6.1: GP Algorithm

6.3 Representation and initial population generation

Each individual in the population is a logical tree representing a classifier. The terminal set is composed of the 41 features from the NSL-KDD dataset. The function set is made up of nine functions (*and*, *not*, *or*, *equal*, *different*, *max*, *min*, *greater*, *least*). The functions were selected based on the critical analysis of previous literature in section 4.6. Figure 6.1 illustrates the representation of an individual and Table 6.1 provides descriptions of the functions in the function set.

Function	Description	Arity
AND	Performs the logical AND operation between two values	2
NOT	Performs the logical NOT operation on a single value	1
OR	Performs the logical OR operation between two values	2
EQUAL	Compares two values and returns 1 if two numbers are the same, otherwise, 0 is returned.	2

DIFFERENT	Compares two values and returns 1 if two numbers are different, otherwise, 0 is returned.	2
MAX	Compares two values and returns the maximum of the two values.	2
MIN	Compares two values and returns the minimum of the two values.	2
GREATER	Compares two numbers and returns 1 if the first number is greater than the second number, otherwise, 0 is returned.	2
LEAST	Compares two numbers and returns 1 if the first number is lower than the second number, otherwise, 0 is returned.	2

Table 6.1: Function descriptions

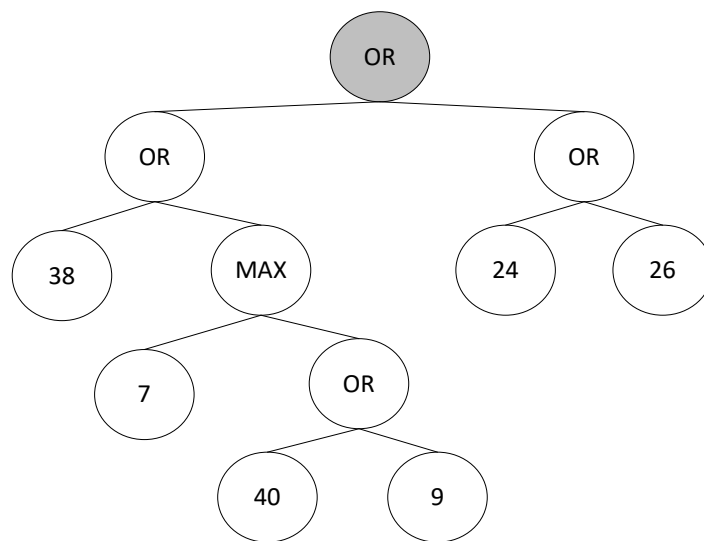


Figure 6.1: Example of an individual

The individual in Figure 6.1 is made up of five functions from the function set, each with an arity of 2. The numbers in the individual represent the feature numbers from the dataset. Each element in the individual is randomly selected beginning with the root node (highlighted node).

The ramped half-and-half method discussed in section 2.5.3 is used for initial population generation. The initial population is generated based on the parameters provided in Table 6.3 and Table 6.4.

6.4 Evaluation

The NSL-KDD training set discussed in section 5.5 is used in the training and generation of the classifiers. The training set is used to calculate the fitness of each individual in the population. The connections in the training set are the fitness cases discussed in section 2.6.1. Each individual in the

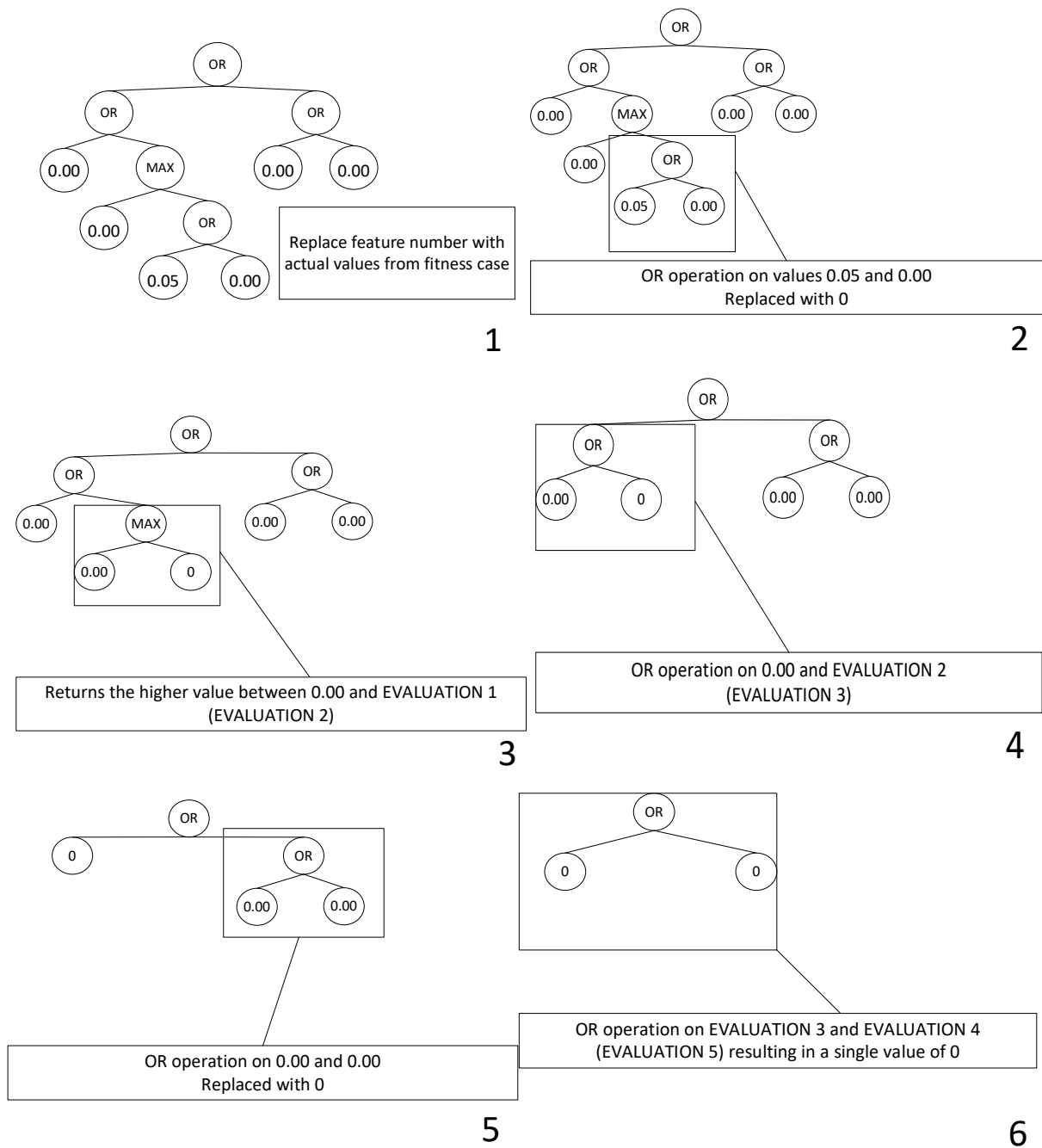


Figure 6.2: Evaluation process

The output from the individual represents the predicted class. The predicted class is compared to the actual class of each of the fitness cases. Based on the results of the comparisons a confusion matrix discussed in section 3.4.1 is constructed. Table 6.2 provides an example of the confusion matrix constructed for the individual in Figure 6.1 after the individual is evaluated over the training set.

		Predicted Class	
		Intrusion	Normal
Actual Class	Intrusion	40367 (TP)	18263 (FN)
	Normal	16887 (FP)	50456 (TN)

Table 6.2: Confusion Matrix

Based on the results from the confusion matrix, the fitness of an individual is calculated using one of the performance measures described in section 3.4. Different performance measures will be compared as fitness measures.

6.5 Selection Method and Genetic Operators

Selections methods have been discussed in section 2.7. Tournament selection is used as the selection

Tournament selection

- Randomly select t (tournament size) individuals from the population creating sample.
- Set first individual in sample as best individual (b).
- Repeat
 - Compare the fitness of b with individuals in sample.
 - If fitness of individual in sample is higher than fitness of b , set b equal to individual with higher fitness.
- Until all individuals in sample have been compared for fitness.

Return the best individual.

Algorithm 6.2: Tournament selection

method and it is outlined in Algorithm 6.2. Individuals are randomly selected from the population. The number of individuals selected is determined by the tournament size. The individual with the best fitness in the tournament is returned as the tournament winner.

Mutation and crossover discussed in section 2.8.2 were used to generate offspring. These genetic operators have been widely used in previous studies. Figure 6.3 and Figure 6.4 illustrates the genetic operators crossover and mutation.

In crossover, random crossover points are selected in each of the two parents selected using tournament selection. The subtrees located at the crossover points were exchanged between the two parents generating the offspring. In Figure 6.3 the highlighted nodes represent the crossover points.

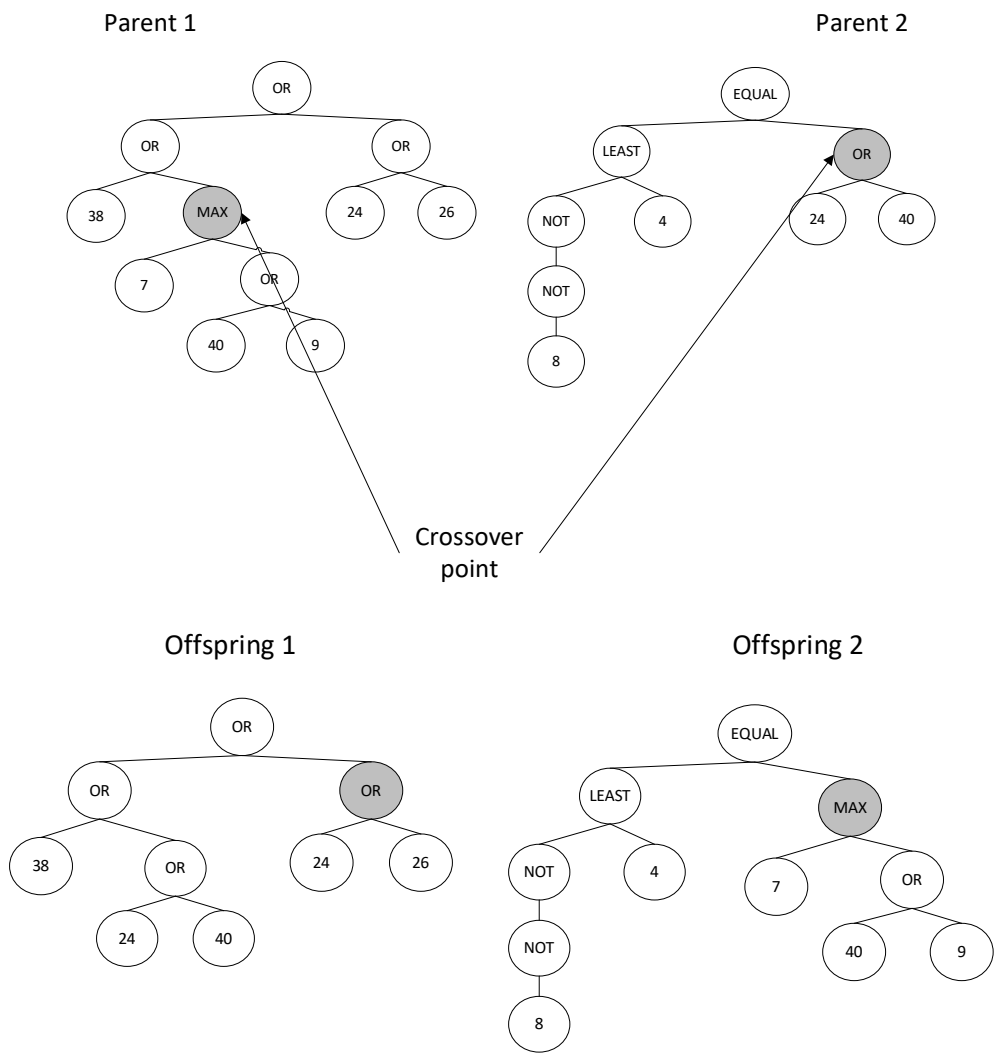


Figure 6.3: GP Crossover

For mutation, a random mutation point is selected in the parent. A randomly generated tree is generated using the grow method of initial population generation. After the random tree is generated, it replaces the subtree at the mutation point of the parent generating the offspring. The mutation point is highlighted in both the parent and offspring in Figure 6.4.

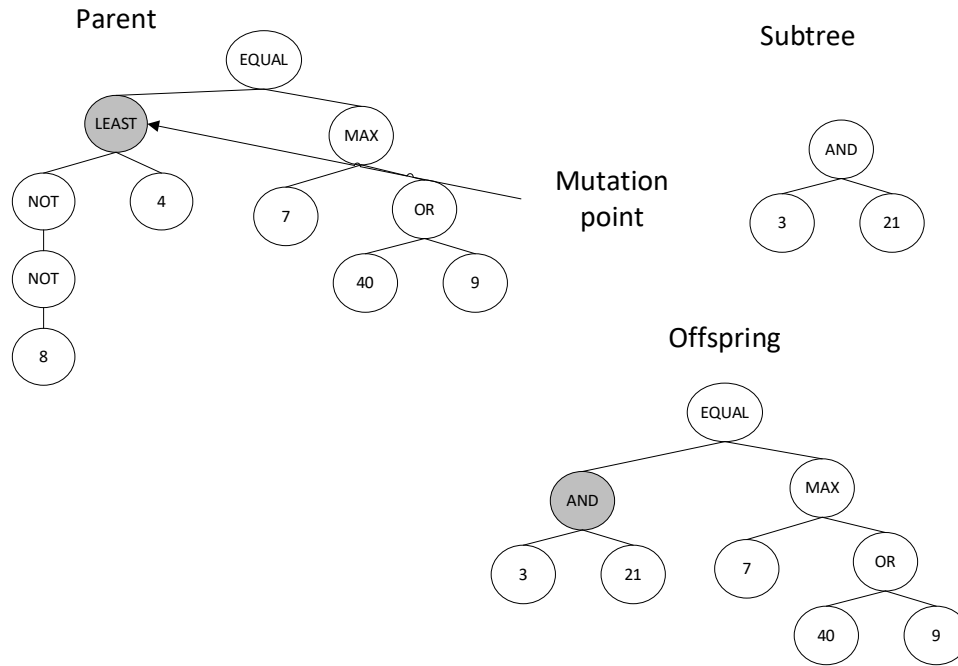


Figure 6.4: GP Mutation

6.6 Parameters

The parameters used for the generation of binary classifiers are provided in Table 6.3. These parameters were determined empirically through multiple trial runs.

GP Parameter	Value
Population size	200
Initial population generation method	Ramped half and half
Initial population maximum tree size	4
Selection method	Tournament with a size of 20
Fitness function	Accuracy
Mutation application rate	50%
Crossover application rate	50%
Maximum mutation depth	3
Maximum offspring depth	8
Maximum number of generations	500

Table 6.3: GP Parameters for binary classifiers

The parameters summarised in Table 6.4 were used during the generation of multi-class classifiers. These parameters were determined empirically through multiple trial runs. Different fitness functions were used in order to evaluate the performance of the classifiers.

GP Parameter	Value
Population size	200
Initial population generation method	Ramped half and half
Initial population maximum tree size	4
Selection method	Tournament with a size of 10
Fitness functions	Accuracy F-Score Matthews correlation coefficient False positive rate Precision True positive rate
Mutation application rate	60
Crossover application rate	40
Maximum mutation depth	3
Maximum offspring depth	8
Maximum number of generations	500

Table 6.4: GP Parameters for multi-class classifiers

6.7 Chapter Summary

This chapter presented the proposed GP approach for generating binary and multi-class classifiers for network intrusion detection using genetic programming. An overview of the GP algorithm used for the generation of the classifiers was outlined. Representation of individuals, evaluation, selection methods and the genetic operators used for generating the classifiers were discussed. The parameters that were used for the generation of the classifiers were also provided.

7 Grammatical Evolution for Network Intrusion Detection

7.1 Introduction

This chapter describes a grammatical evolution approach for generating binary and multi-class classifiers for network intrusion detection. Section 7.2 discusses the representation of individuals and the grammar used for generating the classifiers. The initial population generation and evaluation is discussed in section 7.3. Genetic operators and the selection method that will be used are discussed in section 7.4. Section 7.5 provides the parameters for the proposed approach and section 7.6 concludes the chapter.

7.2 Representation

The GE algorithm discussed in section 2.15 is used for the generation of the classifiers described in this chapter. The best individual evolved will be evaluated on the testing set of the NSL-KDD dataset. Each individual is made up of multiple binary strings. Eight-bits are used to represent a binary string. Each bit in the binary string is randomly selected to be either 0 or 1. The following are examples of randomly created binary strings representing an individual:

01010010 10100001 00011100 00011111 11111001 01000101 11100001

The grammar is described below.

$$R = \{N, T, S, P\}$$

Where

$$N = \{exp, op\}$$
$$T = \{and, not, or, equal, different, max, min, greater, least, 1 \dots 41\}$$

$$S = \{<op> <exp> <exp>\}$$

Production rules (P) are:

(1)	$<exp> ::=$	$<op> <exp> <exp>$	(0)
		$<var>$	(1)
(2)	$<op> ::=$	AND	(0)
		NOT	(1)
		OR	(2)
		EQUAL	(3)
		DIFFERENT	(4)
		MAX	(5)
		MIN	(6)
		GREATER	(7)
		LEAST	(8)
(3)	$<var> ::=$	attr_1	(0)
		attr_2	(1)
	
		attr_41	(40)

Each of the productions rules ($<exp>$, $<op>$ and $<var>$) map to other variables as illustrated above. Each variable in the production rule maps to a numerical value used to distinguish it from other variables and the numerical value is also used during the mapping process. The functions contained in the production rule $<op>$ perform the same functions as the ones described in section 6.3 and the terminal symbols contained in $<var>$ are the 41 features of the dataset. The variables that are mapped from expanding $<exp>$ in production rule (1) are considered as non-terminating symbols and the variables in production rules (2) and (3) are considered as terminating symbols.

7.3 Initial Population Generation and Evaluation

Each individual in the population is randomly created as discussed in section 2.17. The number of individuals created is determined by the population size. The number of codons determines the size of the individual (binary string length).

Each individual is evaluated by converting the binary strings to denary values and then mapping the denary values to an expression tree. The grammar described above is used during the mapping process. The expression tree generated after the mapping process is a logical tree. Figure 7.1 illustrates conversion of binary strings to denary values.

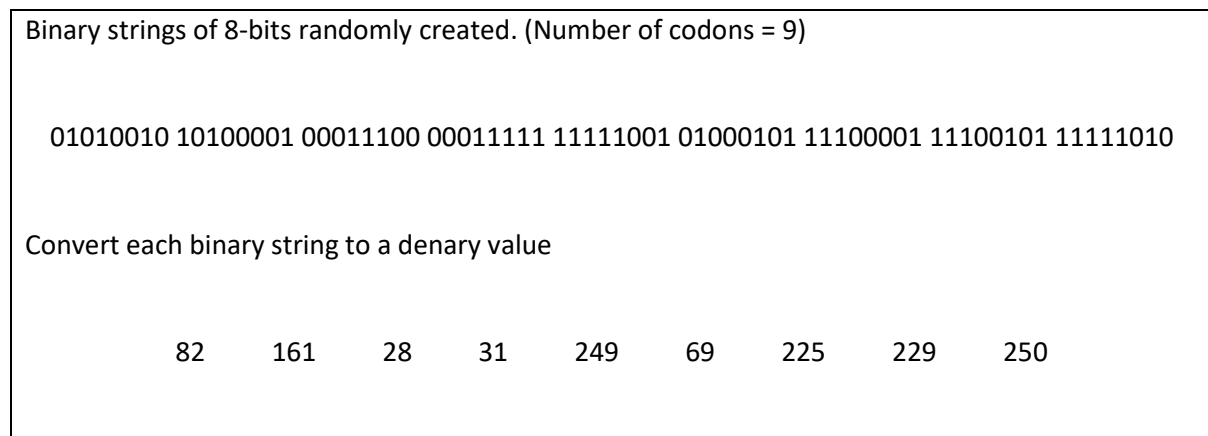


Figure 7.1: Binary to denary conversion

After the conversion of binary strings to denary values, the mapping process outlined in section 2.16.2 is applied, generating an expression tree used to calculate the fitness of the individual. Figure 7.2 illustrates the denary values and the expression tree.

Expression: OR LEAST 29 25 LEAST 17 MIN 30 27
 Denary Codons: 82 161 28 31 249 69 225 229 250 47 255 16 114 109 91 193 247

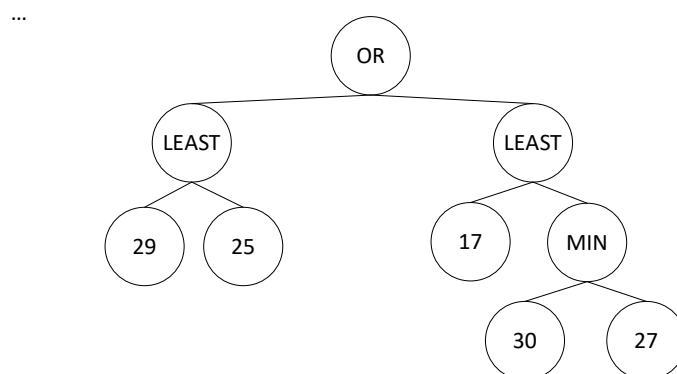


Figure 7.2: GE Individual

The individual in Figure 7.2 is made up of 18 denary codons which were used in the generation of the expression tree. The expression tree is made up of 4 functions and 5 terminal nodes. The OR function represents the root node of the expression tree.

After the expression tree is generated, the expression tree is evaluated on the training set as discussed in section 6.5. The fitness of an individual is determined by the fitness of the expression tree. After evaluation of the expression tree, the fitness of the expression tree is calculated using one of the performance measures described in section 3.4. Accuracy will be used as the fitness function for binary classifiers and different performance measures will be compared as fitness functions for multi-class classifiers. Analysis of previous literature conducted in Chapter 4.6 has shown that accuracy has been successfully applied for generating binary classifiers which achieve high accuracies, and different performance measures have been used as fitness functions for generating multi-class classifiers with high accuracies and investigations into the effects of different performance measures is worth researching.

7.4 Selection Method and Genetic Operators

Tournament selection discussed in section 2.7.1 where a single individual is selected from a sample of individuals is used as the selection method for the proposed grammatical evolution approach.

Genetic operators crossover and mutation are applied during the GE run. Bit flip mutation discussed in section 2.18.2 is applied as the mutation operator. During bit flip mutation, each bit in the individual is inverted (0 is changed to 1 and 1 is changed to 0) generating an offspring. Uniform crossover is applied as the crossover operator. Uniform crossover randomly selects alleles within each of the parents and swaps them between the parents to generate offspring. Figure 7.3 illustrates uniform crossover where alleles at index 1,2,4,5,8 from the first binary string and index 1,4,6,7,8 from the second binary string were exchanged between the two parents to generate the offspring. The alleles highlighted in grey represent alleles from parent 1 and the alleles highlighted in blue represent alleles from parent 2.

Parent 1	10011010	01010111
Parent 2	10111100	00011101	...
Offspring 1	10111110	01011101	...
Offspring 2	10111000	00010111	..

Figure 7.3: GE uniform crossover

7.5 Parameters

The parameters that were used for the generation of the proposed binary classifiers are summarized in the Table 7.1. These parameters were determined empirically through multiple trial runs.

GE Parameter	Value
GE model	Generational Model
Population size	200
Number of codons	[30,100]
Wrap-over limit	12
Nonterminal limit	10
Selection method	Tournament with a size of 14
Fitness function	Accuracy
Mutation application rate	50%
Crossover application rate	50%
Maximum number of generations	500

Table 7.1: GE Parameters for binary classification

The parameters summarized in the Table 7.2 were used for the GE approach producing the proposed multi-class classifiers. These parameters were determined empirically through multiple trial runs. The performance measures which will be compared are mentioned as fitness functions.

GE Parameter	Value
GE model	Generational model
Population size	200
Number of codons	[30,100]
Wrap-over limit	10
Nonterminal limit	8
Selection method	Tournament with a size of 8
Fitness functions	Accuracy F-Score Matthews correlation coefficient False positive rate Precision

	True positive rate
Mutation application rate	40
Crossover application rate	60
Maximum number of generations	500

Table 7.2: GE Parameters for multi-class classification

7.6 Chapter Summary

This chapter presented a grammatical evolution approach for generating binary and multi-class classifiers for network intrusion detection. The grammar used was presented. Genetic operators and the evaluation of the individuals were discussed. The parameters which were used in the generation of the classifiers were provided.

8 Multi-Expression Programming for Network Intrusion Detection

8.1 Introduction

This chapter describes a multi-expression programming approach for generating binary and multi-class classifiers for network intrusion detection. Section 8.2 provides an overview of the MEP algorithm used for the generation of the proposed classifiers. The representation is discussed in section 8.3. Section 8.4 discusses the initial population generation and the evaluation of individuals. The selection method and the genetic operators used are discussed in section 8.5. The parameters used are provided in section 8.6 and section 8.7 summarizes the chapter.

8.2 MEP Algorithm

The generational control model is used. The initial population is created and evaluated. Each individual in the population is evaluated using the training set of the NSL-KDD dataset. Tournament selection is used to select the parents used for genetic operators. New individuals are generated by the genetic operators and form part of the new population. The evaluation and generation of individuals is iteratively repeated until the maximum number of generations is reached. The best individual from the MEP run is returned and evaluated using the testing set of the NSL-KDD dataset. Algorithm 8.1 summarizes the MEP algorithm used to generate the classifiers.

MEP Algorithm

Begin

- Create an initial population

- Repeat
 - Evaluate individuals in the population.
 - Select parents using tournament selection.
 - Apply genetic operators to selected parents to generate offspring.
 - Offspring form new population.

- Until a maximum number of generations.

End

Return the best individual and evaluate on the testing set.

Algorithm 8.1: MEP algorithm

8.3 Representation

MEP individuals are made up of genes of variable length as discussed in section 2.23. Each gene within an individual is either a terminal gene or function gene. A terminal gene is created whenever a single terminal symbol is selected from the terminal set to represent the gene. The terminal set is made up of the 41 features from the dataset. A function gene is created by combining a function symbol from the function set and pointers representing other genes within the same individual. The function set is made up of nine functions (*and, not, or, equal, different, max, min, greater, least*). These functions perform the same functions as described in section 6.3. Each gene in an individual is similar to an individual generated using GP. The number of genes which make up an individual is a MEP parameter. Figure 8.1 illustrates an individual with 11 genes and the pointers representing other genes within the same individuals. Individuals in the population are logical trees representing classifiers.

	Gene	Gene pointer(s)
gene1	38	
gene2	37	
gene3	MIN 37 38	MIN gene2 gene1
gene4	MIN 37 MIN 37 38	MIN gene2 gene3
gene5	36	
gene6	31	
gene7	MAX MIN 37 38 38	MAX gene3 gene1
gene8	NOT MIN 37 MIN 37 38	NOT gene4
gene9	EQUAL 38 38	EQUAL gene1 gene1
gene10	40	
gene11	EQUAL MIN 37 MIN 37 38 MIN 37 38	EQUAL gene4 gene3

Figure 8.1: MEP Individual

The individual in Figure 8.1 is made up of 5 terminal genes and 6 function genes. The gene pointers illustrate the function arguments used to create each function gene. Gene 7 is a function gene created from the combination of gene 3, gene 1 and the MAX function.

8.4 Initial Population Generation and Evaluation

The first gene in an individual is a terminal symbol and the rest of the genes in the individual are either terminal or function genes as discussed in section 2.24. The number of individuals created during initial population generation is determined by the population size.

In order to evaluate the performance of an individual, each gene in the individual is evaluated using the NSL-KDD training set in the same manner as the evaluation of GP individuals discussed in 6.4. After each gene is evaluated, the gene with the best fitness represents the overall fitness of the individual.

8.5 Selection Method and Genetic Operators

Tournament selection discussed in section 2.7.1 is used as the selection method. From the analysis of previous work provided in Chapter 4, tournament selection was widely used in previous literature.

Genetic operators mutation and crossover are applied during the generation of MEP classifiers. Uniform crossover is used as the crossover operator and the mutation operator discussed in section 2.25.2 is used. Figure 8.2 illustrates uniform crossover applied to a MEP individual. For uniform crossover, after genes have been exchanged between the parents, each of the function genes exchanged update the function pointers to point to the genes within the individual. For example, for the individual in Figure 8.2, gene 2 in parent 2 is expressed as (AND 36 36) evaluating to (AND gene1 gene1), after the gene is exchanged to form part of offspring 1, it updates its function pointer evaluating to (AND 38 38).

Parent 1		Parent 2	
1.	38	1.	36
2.	37	2.	AND 36 36
3.	MIN 37 38	3.	29
4.	MIN 37 MIN 37 38	4.	28
5.	36	5.	LEAST 36 28
6.	31	6.	27
7.	MAX MIN 37 38 38	7.	40
8.	NOT MIN 37 MIN 37 38	8.	OR LEAST 36 28 36
9.	EQUAL 38 38	9.	MIN 28 40

- Offspring 1
1. 38
 2. AND 38 38
 3. MIN AND 38 38 38
 4. MIN AND 38 38 MIN AND 38 38 38
 5. LEAST 38 MIN 37 MIN 37 38
 6. 31
 7. MAX MIN AND 38 38 38 38
 8. NOT MIN AND 38 38 MIN AND 38 38 38
 9. EQUAL 38 38

- Offspring 2
1. 36
 2. 37
 3. 29
 4. 28
 5. 36
 6. 27
 7. 40
 8. OR LEAST 36 28 36
 9. MIN 28 40

Figure 8.2: MEP Uniform Crossover

8.6 Parameters

Table 8.1 lists the parameters that were used for the generation of MEP binary classifiers proposed in this chapter. The parameters were determined empirically through multiple trial runs.

MEP Parameter	Value
Population size	200
Number of genes per individual	30
Selection method	Tournament with a size of 15
Fitness function	Accuracy
Mutation application rate	60%
Crossover application rate	40%
Maximum number of generations	600

Table 8.1: MEP Parameters for binary classification

The MEP parameters that were used for the generation of multi-class classifiers proposed in this chapter are summarized in the Table 8.2. These parameters were determined empirically through multiple trial runs.

MEP Parameter	Value
Population size	200
Number of genes per individual	30
Selection method	Tournament with a size of 8
Fitness functions	Accuracy F-Score Matthews correlation coefficient False positive rate Precision True positive rate
Mutation application rate	40
Crossover application rate	60
Maximum number of generations	500

Table 8.2: MEP Parameters for multi-class classification

8.7 Chapter Summary

This chapter presented the multi-expression programming approach for generating classifiers for network intrusion detection. An overview of the algorithm was provided and each aspect of the algorithm was discussed. The parameters that were used for the generation of the binary and multi-class classifiers were provided.

9 Results and Discussion

9.1 Introduction

This chapter presents the results obtained from applying the proposed approaches presented in this thesis for generating binary and multi-class classifiers for network intrusion detection. The following summarizes the objectives of the study presented in this thesis outlined in Chapter 1:

- Develop and analyse the performance of using grammatical evolution for generating intrusion detection classifiers.
- Development and evaluation of applying Multi-expression programming (MEP) for generating binary and multi-class classifiers for network intrusion detection.
- Develop and analyse the performance of using genetic programming (GP) for generating binary and multiclass classifiers for network intrusion detection.
- Investigate the effectiveness of fitness functions for network intrusion detection.
- Comparative analysis of GE, MEP and GP for network intrusion detection.

Section 9.2 presents the results of applying the proposed grammatical evolution approach described in Chapter (GE), Section 9.3 presents the results of applying the proposed Multi-Expression Programming approach described in Chapter (MEP) and Section 9.4 presents the results obtained from applying the proposed genetic programming approach described in Chapter (GP). Section 9.5 compares the results for binary and multi-class classifiers presented for each of the approaches presented in this thesis. Section 9.6 compares the results of the proposed approaches to state of the art approaches for network intrusion detection and section 9.7 summarizes the chapter.

9.2 Grammatical Evolution

This section presents the results obtained by applying the GE approach described in Chapter 7 for binary and multi-class classification.

9.2.1 Binary Classification

The classifiers that produced best and worst results for detecting intrusions using grammatical evolution are presented in Table 9.1. The results in the table represent the accuracy for detecting both intrusive and non-intrusive connections during training and testing. The classifier achieved a high accuracy of 94.06% during training and 74.55% during testing, a true positive rate of 94.13% during training and 60.41% during testing. The classifier achieved a true negative rate of 95.38% during training and 86.78% during testing. The classifier achieved a false positive rate of 6.77% during testing as compared to 6% achieved during training of the classifier.

	Training	Testing
	Accuracy	
Best classifier	94.06 ± 0.13	74.55 ± 0.24
Worst classifier	88.93 ± 0.17	73.19 ± 0.24
Average	91.39 ± 0.15	

Table 9.1: Grammatical Evolution binary classification results

The average runtime of GE to evolve a classifier was 3 hours during training and evaluation of the classifier on the testing set took an average runtime of 30 seconds.

9.2.2 Multi-class classification

The subsections below present the results of each of the six performance measures which were used as fitness functions for generating multi-class classifiers for network intrusion detection using grammatical evolution. The average runtime of GE to evolve each of the classifiers discussed below was 3 hours during training and evaluation of performance on the testing set took an average runtime of 20 seconds for each of the classifiers.

9.2.2.1 Accuracy

Table 9.2 presents the results of the best and worst classifiers generated from using accuracy as the fitness function for generating multi-class classifiers using grammatical evolution. The best classifier obtained a low false positive rate of 3.6% during training and 5.5% during testing. High detection rates

were achieved for detecting U2R attacks with the best classifier achieving a 90% true positive rate during testing and 86.56% during training.

		DOS	Probe	U2R	R2L
Training	Best	95.57 ± 0.11	95.10 ± 0.12	99.97 ± 0.01	99.44 ± 0.04
	Worst	92.80 ± 0.14	90.78 ± 0.16	99.96 ± 0.01	99.21 ± 0.05
	Average	94.30 ± 0.13	92.80 ± 0.14	99.96 ± 0.01	99.27 ± 0.05

Testing	Best	85.33 ± 0.19	92.18 ± 0.15	99.71 ± 0.03	87.21 ± 0.18
	Worst	86.39 ± 0.19	89.27 ± 0.17	99.71 ± 0.03	87.20 ± 0.18
	Average	84.66 ± 0.20	91.68 ± 0.15	99.71 ± 0.03	88.17 ± 0.18

Table 9.2: Grammatical Evolution accuracy multi-classification results

The classifiers generated during training achieved similar detecting rates for U2R attacks resulting in the similar classification rates during testing for U2R. Probe attacks achieved high detection rates during testing despite achieving the lowest detection rates during training.

9.2.2.2 Matthews coefficient correlation

The training and testing results of the classifiers that were generated using Matthews's coefficient correlation as the fitness function for multi-class classification using grammatical evolution are presented in Table 9.3.

During training, the best classifier achieved a high accuracy rate and a low true positive rate of 3%. The false positive rate during testing was higher than training at 5%. Using MCC as the fitness function during training and testing resulted in a high average classification rate. The high average classification is attributed to the balance between the true positive rate and true negative rate by the fitness function.

		DOS	Probe	U2R	R2L
Training	Best	96.21 ± 0.11	95.47 ± 0.11	99.97 ± 0.01	99.21 ± 0.05
	Worst	63.54 ± 0.26	90.75 ± 0.16	99.94 ± 0.01	98.44 ± 0.07
	Average	92.15 ± 0.13	92.94 ± 0.14	99.96 ± 0.01	98.98 ± 0.05

Testing	Best	84.35 ± 0.20	88.94 ± 0.17	99.74 ± 0.02	86.91 ± 0.19
	Worst	66.91 ± 0.26	89.26 ± 0.17	99.71 ± 0.03	89.11 ± 0.17
	Average	83.56 ± 0.20	90.42 ± 0.16	99.71 ± 0.03	87.15 ± 0.18

Table 9.3: Grammatical Evolution MCC multi-classification results

9.2.2.3 F-Score

Table 9.4 presents the results of the best and worst classifiers generated from applying f-score as the fitness function for multi-class classification using grammatical evolution. The best classifier achieved a false positive rate of 3.7% during training and 6.34% during testing. The best classifier achieved high detection rates with the detection of U2R intrusions achieving above 99% during both training and testing.

		DOS	Probe	U2R	R2L
Training	Best	96.17 ± 0.11	95.10 ± 0.12	99.97 ± 0.01	99.42 ± 0.04
	Worst	92.85 ± 0.14	92.44 ± 0.15	58.63 ± 0.27	80.38 ± 0.22
	Average	94.45 ± 0.13	92.95 ± 0.14	97.16 ± 0.03	92.64 ± 0.12

Testing	Best	87.54 ± 0.18	92.18 ± 0.15	99.70 ± 0.03	89.26 ± 0.17
	Worst	84.38 ± 0.20	91.93 ± 0.15	46.52 ± 0.27	84.26 ± 0.20
	Average	84.42 ± 0.20	91.35 ± 0.15	96.13 ± 0.05	85.61 ± 0.19

Table 9.4: Grammatical Evolution f-score multi-classification results

The best classifier achieved a low detection rate for DOS attacks during testing as compared to the detection of all the other intrusive attacks. The average detection rate and the worst performing

classifier achieved similar results for detecting DOS attacks. The worst classifier also struggled to detect U2R attacks both during training and testing.

9.2.2.4 True Positive Rate

The training and testing results of the multi-class classifiers generated using grammatical evolution and using the true positive rate as the fitness function are presented in Table 9.5. The classifiers achieved low detection rates with the best classifier achieving a detection rate of 39.14% during training and 39.44% during testing. The approach (grammatical evolution using TPR as the fitness function) achieved the lowest detection rates as compared to the other grammatical evolution approaches. The classifiers achieved low detection rates due to the datasets including a large portion of normal connections as compared to the intrusive connections. The classifiers correctly classified intrusive connections and also achieved a high rate of false positives resulting in a low accuracy of the classifiers.

		DOS	Probe	U2R	R2L
Training	Best	39.14 ± 0.27	10.25 ± 0.17	36.21 ± 0.26	28.55 ± 0.25
	Worst	36.46 ± 0.26	9.25 ± 0.16	0.04 ± 0.01	0.79 ± 0.05
	Average	37.08 ± 0.27	9.48 ± 0.16	7.42 ± 0.08	8.88 ± 0.11
Testing	Best	39.44 ± 0.27	11.43 ± 0.18	22.84 ± 0.23	21.74 ± 0.23
	Worst	33.09 ± 0.26	10.74 ± 0.17	0.30 ± 0.03	12.8 ± 0.18
	Average	33.65 ± 0.26	10.91 ± 0.17	7.42 ± 0.08	16.42 ± 0.20

Table 9.5: Grammatical Evolution TPR multi-classification results

The best classifier achieved a true positive rate of 67% during training and 70% during testing. The rate of true negatives (correct detection of normal connections) was low resulting in high false positive rates of 60% during training and 56% during testing.

9.2.2.5 Precision

Table 9.6 presents the results of the multi-class classifiers generated using grammatical evolution and using precision as the fitness function. The best classifier achieved a false positive rate of 7% during training and 14% during testing.

		DOS	Probe	U2R	R2L
Training	Best	64.51 ± 0.26	90.76 ± 0.16	99.96 ± 0.01	99.25 ± 0.05
	Worst	63.54 ± 0.26	90.75 ± 0.16	99.96 ± 0.01	99.21 ± 0.05
	Average	64.20 ± 0.26	90.75 ± 0.16	99.96 ± 0.01	99.21 ± 0.05

Testing	Best	66.34 ± 0.26	89.25 ± 0.17	99.71 ± 0.03	87.37 ± 0.18
	Worst	66.91 ± 0.26	89.15 ± 0.17	99.69 ± 0.03	87.20 ± 0.18
	Average	66.83 ± 0.26	89.24 ± 0.17	99.70 ± 0.03	87.22 ± 0.18

Table 9.6: Grammatical Evolution precision multi-classification results

The classifiers achieved low DOS detection rate during training and testing. The average detection rate, worst classifier detection rate and best classifier intrusion detection rates were similar during training and testing. The approach achieved a high detection rate for Probe attacks during testing and a lower detection rate during training as compared to the detection of R2L attacks.

9.2.2.6 False Positive Rate

The results of the training and testing of the multi-class classifiers generated using grammatical evolution and using false positive rate as the fitness function are presented in Table 9.6. The classifiers achieved low false positive rates of 6% during training and 9% during testing. The classifiers also achieved similar detection rates both during training and testing as summarized in the table.

		DOS	Probe	U2R	R2L
Training	Best	63.82 ± 0.26	90.75 ± 0.16	99.96 ± 0.01	99.21 ± 0.05
	Worst	63.54 ± 0.26	90.75 ± 0.16	99.96 ± 0.01	99.21 ± 0.05
	Average	63.55 ± 0.26	90.75 ± 0.16	99.96 ± 0.01	99.21 ± 0.05

Testing	Best	66.91 ± 0.26	89.26 ± 0.17	99.70 ± 0.03	87.20 ± 0.18
	Worst	66.86 ± 0.26	89.26 ± 0.17	99.70 ± 0.03	87.20 ± 0.18
	Average	66.91 ± 0.26	89.26 ± 0.17	99.70 ± 0.03	87.20 ± 0.18

Table 9.7: Grammatical Evolution FPR multi-classification results

9.2.3 Analysis of multi-class classification for GE approach

The results of the testing phase using the different performance measures as fitness functions for generating multi-class classifiers for GE are summarised in Figure 9.1.

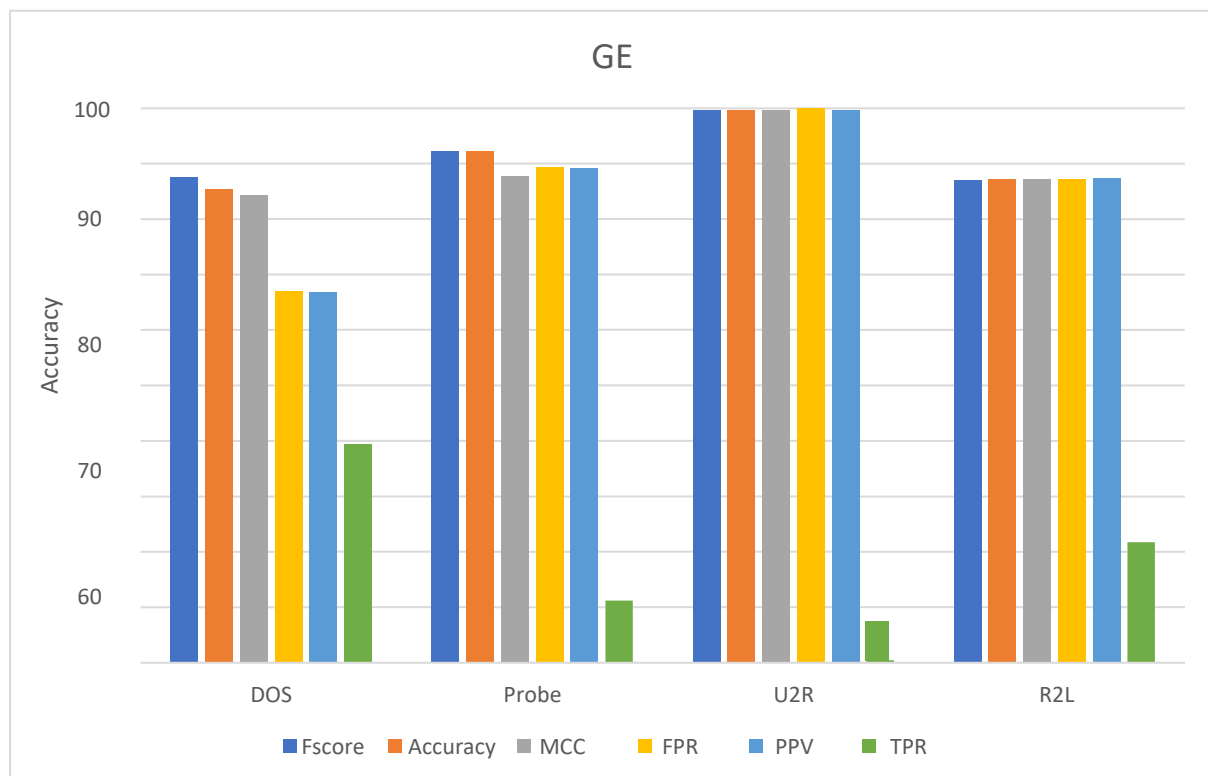


Figure 9.1: GE comparison of fitness function performance

The classifier generated using f-score as the fitness function outperformed the other classifiers. The results were not statistically significant when compared to the results of the classifier produced

using accuracy as the fitness function. Grammatical evolution performed well achieving generally high accuracy rates for both binary classification and multi-class classification. Classifiers that were produced using f-score, accuracy and Matthews’s coefficient correlation achieved higher accuracy rates. The higher accuracy rates suggest that the performance measures perform well for detecting both intrusive and normal connections as well as unknown data. Both classifiers achieved the same accuracy for detecting Probe attacks. The classifier using f-score will be used when comparing the different approaches.

9.3 Multi-Expression Programming

This section presents the results obtained by the applying the MEP approach described in Chapter 8 for binary and multi-class classification.

9.3.1 Binary Classification

Multi-expression programming was successfully applied for the generation of binary classifiers. The results of the approach are presented in Table 9.8. The best classifier achieved a true positive rate of 93.2% during training and 66% during testing. The classifier achieved a higher false positive rate of 5.6% during testing as compared to 3.95% achieved during the training and generation of the classifier. The overall performance of the classifier was high achieving a high accuracy rate of 78.23% during testing as summarised in Table 9.8.

	Training	Testing
	Accuracy	
Best classifier	94.72 ± 0.12	78.23 ± 0.23
Worst classifier	76.69 ± 0.23	76.69 ± 0.23
Average	92.59 ± 0.14	

Table 9.8: Multi-Expression programming binary classification results

The average runtime of MEP to evolve a classifier was 6 hours during training and evaluation of the classifier on the testing set took an average runtime of 1 minute.

9.3.2 Multi-class classification

The subsections below present the results of applying MEP using each of the six performance measures outlined in section 8.6 as fitness functions for generating multi-class classifiers for network intrusion detection. The average runtime of MEP to evolve each of the classifiers discussed below was

6 hours during training and evaluation of performance on the testing set took an average runtime of 1 minute for each of the classifiers.

9.3.2.1 Accuracy

The training and testing results of the classifiers that were generated using accuracy as the fitness function for multi-class classification using multi-expression programming are presented in Table 9.9. The best classifier obtained a high false positive rate of 5% during training and 7% during testing. Tests were performed to find the possible reasons for the high false positive rate. High accuracy rates were achieved for detecting intrusive connections with the best classifier achieving a 96% true positive rate during testing.

		DOS	Probe	U2R	R2L
Training	Best	97.42 ± 0.09	96.08 ± 0.11	99.97 ± 0.01	99.33 ± 0.05
	Worst	93.5 ± 0.14	91.12 ± 0.16	99.96 ± 0.01	99.25 ± 0.04
	Average	95.28 ± 0.12	92.67 ± 0.14	99.96 ± 0.01	99.28 ± 0.04
Testing	Best	86.10 ± 0.19	91.24 ± 0.16	99.70 ± 0.03	88.98 ± 0.17
	Worst	85.65 ± 0.19	89.63 ± 0.17	99.70 ± 0.03	87.20 ± 0.18
	Average	85.00 ± 0.20	90.95 ± 0.16	99.70 ± 0.03	88.44 ± 0.18

Table 9.9: MEP accuracy multi-classification results

The best MEP classifier achieved high detection rates during the training of U2R and R2L intrusive connections and achieved high detection rates for detecting U2R and Probe intrusive connections during testing. The high detection rate for U2R can be attributed to fewer connections existing in the dataset for U2R. Probe attacks achieved the lowest detection rates during training but during testing achieved the second highest detection rates.

9.3.2.2 Matthews's coefficient correlation

The following table presents the results obtained from applying MCC as the fitness function for generating multi-class classifiers. During training, the best classifier achieved a high accuracy rate and a low false positive rate of 4%. The low positive rate during testing was low (3.5%) similar to the one achieved during training. Individuals generated using MCC as a performance measure have a balance between the true positive rate and the true

negative rate resulting in low false positive rates. The low false positive rate and high true positive rate resulted in classifiers achieving a high accuracy as summarised in Table 9.10.

		DOS	Probe	U2R	R2L
Training	Best	97.51 ± 0.09	95.97 ± 0.11	99.97 ± 0.01	99.33 ± 0.05
	Worst	92.8 ± 0.14	92.37 ± 0.15	99.95 ± 0.01	98.32 ± 0.07
	Average	94.94 ± 0.12	94.04 ± 0.13	99.96 ± 0.01	98.98 ± 0.05

Testing	Best	85.72 ± 0.19	92.21 ± 0.15	99.71 ± 0.03	88.72 ± 0.17
	Worst	82.49 ± 0.21	91.75 ± 0.15	99.69 ± 0.03	90.41 ± 0.16
	Average	85.23 ± 0.19	90.15 ± 0.16	99.71 ± 0.03	90.68 ± 0.18

Table 9.10: MEP Matthews’s coefficient correlation multi-classification results

9.3.2.3 F-Score

Table 9.11 presents the results of the best and worst classifiers generated from using f-score as the fitness function for generating multi-class classifiers using multi gene programming. The best classifier during training achieved high accuracies for detecting intrusions with the highest accuracy achieved for the detection of U2R attacks and DOS attacks. The best classifier achieved a false positive rate of 3.7% during training and 5.4% during testing.

		DOS	Probe	U2R	R2L
Training	Best	98.16 ± 0.07	96.45 ± 0.10	99.97 ± 0.01	99.32 ± 0.05
	Worst	92.94 ± 0.14	91.32 ± 0.15	71.46 ± 0.25	79.54 ± 0.22
	Average	95.82 ± 0.11	93.98 ± 0.13	91.49 ± 0.10	90.51 ± 0.14

Testing	Best	86.24 ± 0.19	94.03 ± 0.13	99.76 ± 0.03	87.06 ± 0.18
	Worst	85.26 ± 0.20	91.21 ± 0.16	62.34 ± 0.27	77.83 ± 0.17
	Average	85.30 ± 0.19	90.74 ± 0.16	90.14 ± 0.11	85.44 ± 0.19

Table 9.11: MEP f-score multi-classification results

The worst classifier achieved the lowest detection rate for detecting U2R attacks both during training and testing even though the number of U2R connections within the dataset were the lowest. The low classification rate was a result of a low true positive rate and high false positive rate of 60%.

9.3.2.4 True Positive Rate

Table 9.12 presents the results obtained from applying MEP using the true positive rate as the fitness function. The best classifier achieved a true positive rate of 95% during testing but could not correctly distinguish between normal connections and intrusive connections resulting in a high false positive rate of 18%.

		DOS	Probe	U2R	R2L
Training	Best	64.36 ± 0.26	92.33 ± 0.15	99.21 ± 0.05	99.21 ± 0.05
	Worst	63.54 ± 0.27	90.75 ± 0.16	96.49 ± 0.10	96.49 ± 0.10
	Average	63.60 ± 0.27	90.82 ± 0.16	99.05 ± 0.05	99.05 ± 0.05
Testing	Best	66.87 ± 0.27	91.55 ± 0.15	87.20 ± 0.18	87.20 ± 0.18
	Worst	66.91 ± 0.26	89.26 ± 0.17	81.71 ± 0.21	81.71 ± 0.20
	Average	66.90 ± 0.26	89.37 ± 0.17	86.82 ± 0.19	86.92 ± 0.19

Table 9.12: MEP true positive rate multi-classification results

The high true positive rate and high false positive rate resulted in the classifier achieving a low accuracy rate both during training and testing.

9.3.2.5 Precision

The results obtained from applying MEP for generating multi-class classifiers using precision as the fitness function are presented in Table 9.13. The best classifier achieved a true positive of 85% during testing and 90% during training. The classifier also achieved a high false positive rate of 20% during training and 15% during testing. The performance measure measures how well the classifier detects intrusive connections over all the positive connections (true positive rate and false positive rate) returned during training and testing. The high false positive rate and high true positive rate resulted in the classifier achieving a low overall accuracy.

		DOS	Probe	U2R	R2L
Training	Best	63.54 ± 0.27	90.75 ± 0.16	99.21 ± 0.05	99.21 ± 0.05
	Worst	63.54 ± 0.27	89.81 ± 0.17	99.21 ± 0.05	99.21 ± 0.05
	Average	63.54 ± 0.27	90.71 ± 0.16	99.21 ± 0.05	99.21 ± 0.05

Testing	Best	66.91 ± 0.26	89.26 ± 0.17	87.21 ± 0.18	87.21 ± 0.18
	Worst	66.91 ± 0.26	84.71 ± 0.20	87.21 ± 0.18	87.21 ± 0.18
	Average	66.91 ± 0.26	89.28 ± 0.17	87.21 ± 0.18	87.21 ± 0.18

Table 9.13: MEP precision multi-classification results

9.3.2.6 False Positive Rate

The training and testing results of the classifiers that were generated using the false positive rate as the fitness function for multi-class classification using multi-expression programming are presented in Table 9.14. The best classifier correctly detected 80% of normal connections (true negative rate) during training and 85% during testing. The best classifier achieved a low false positive rate during training of 0.02% and 2.1% during testing but failed to correctly detect intrusive connections resulting in the overall performance of the classifier being low.

		DOS	Probe	U2R	R2L
Training	Best	63.54 ± 0.27	90.75 ± 0.16	99.96 ± 0.01	99.21 ± 0.05
	Worst	63.54 ± 0.27	90.75 ± 0.16	93.38 ± 0.13	99.16 ± 0.05
	Average	63.54 ± 0.27	90.75 ± 0.16	99.49 ± 0.02	99.21 ± 0.05

Testing	Best	66.91 ± 0.26	89.26 ± 0.17	99.70 ± 0.03	87.20 ± 0.18
	Worst	66.91 ± 0.26	89.26 ± 0.17	95.08 ± 0.12	87.19 ± 0.18
	Average	66.91 ± 0.26	89.26 ± 0.17	99.37 ± 0.03	87.20 ± 0.18

Table 9.14: MEP false positive rate multi-classification results

Detection of intrusive connections using the false positive rate as the fitness function resulted in low detection rates for DOS attacks in comparison to previous classifiers using different fitness functions. The approach generated similar solutions during training resulting in all the classifiers trained achieving the same detection rate.

9.3.3 Analysis of multi-class classification for MEP approach

Figure 9.2 illustrates the results of the testing phase using the different performance measures for evolving multi-class classifiers using MEP.

From the results presented for multi-class classification using multi-expression programming, the classifiers which were generated using the f-score as the fitness function achieved higher detection rates than the other classifiers generated using the different fitness functions. There was no statistical significance in the results when statistical tests were conducted. It should be noted that even though no statistical significance of the results was achieved, the classifiers using the f-score as the fitness function will be used for comparison with other approaches.

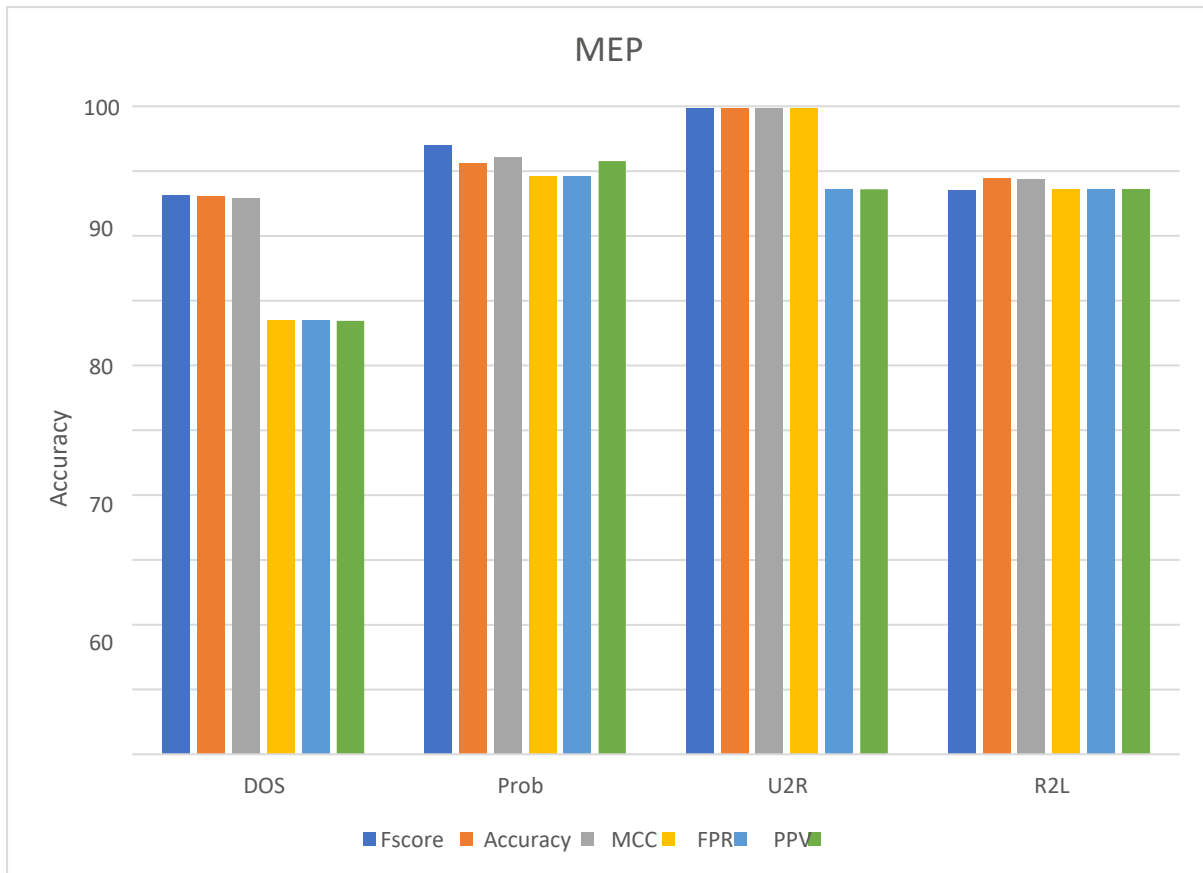


Figure 9.2: MEP comparison of fitness function performance

9.4 Genetic Programming

This section presents the results obtained by the applying the GP approach described in Chapter 6 for binary and multi-class classification.

9.4.1 Binary Classification

The training and testing results of the GP approach are presented in Table 9.1. The table presents the details of the classifier which achieved the highest accuracy (best classifier) as well as the classifier which achieved the lowest accuracy (worst classifier) during training.

	Training	Testing
	Accuracy	
Best classifier	98.06 ± 0.08	80.30 ± 0.22
Worst classifier	96.43 ± 0.10	74.39 ± 0.24
Training Average	97.74 ± 0.08	

Table 9.15: Genetic programming binary classification results

The two classifiers were evaluated on the testing set of the NSL-KDD dataset in order to evaluate the overall performance of the classifiers. The average runtime of GP to evolve a classifier was 5 hours during training and evaluation of the classifier on the testing set took an average runtime of 30 seconds. The best classifier achieved a false positive rate of 1.79% during training. The classifier correctly classified 97.88% of intrusive connections as intrusions and correctly classified 98.21% of normal connections during training. During testing, the best classifier achieved a false positive rate of 3.6%, correctly classified 68.11% of intrusive connections as intrusions and correctly classified 96.40% of normal connections.

9.4.2 Multi-class classification

The results of the six performance measures used as fitness functions for generating multi-class classifiers using the genetic programming approach outlined in Section 6.6 are presented in the subsections below. The average runtime of MEP to evolve each of the classifiers discussed below was 2 hours during training and evaluation of performance on the testing set took an average runtime of 1 minute for each of the classifiers.

9.4.2.1 Accuracy

Table 9.2 presents the results of using accuracy as the fitness function for generating a multiclass classifier. The best classifier achieved high detection rates of more than 99% for each of the network attacks during training and the detection rates reduced during testing. The false positive rate of the best classifier was 6%. The classifier achieved the highest detection rates for U2R attacks. The high detection of U2R attacks can be attributed to the few U2R connections which exist in both the training and the testing sets. The classifier also achieved a high detection rate for Probe attacks which contribute 9% of the training set and 11% of the testing set. This contributes in the classifier achieving an overall high detection rate.

		DOS	Probe	U2R	R2L
Training	Best	99.76 ± 0.03	98.97 ± 0.06	99.98 ± 0.01	99.7 ± 0.03
	Worst	94.52 ± 0.13	92.48 ± 0.14	99.96 ± 0.01	99.21 ± 0.05
	Average	97.89 ± 0.07	96.74 ± 0.09	99.97 ± 0.01	99.32 ± 0.05

Testing	Best	88.93 ± 0.17	92.97 ± 0.14	99.73 ± 0.03	88.34 ± 0.18
	Worst	84.47 ± 0.20	91.69 ± 0.15	99.70 ± 0.03	87.20 ± 0.18
	Average	86.84 ± 0.19	92.36 ± 0.15	99.71 ± 0.03	87.61 ± 0.18

Table 9.16: GP accuracy multi-classification results

9.4.2.2 Matthews's coefficient correlation (MCC)

The results obtained from applying GP for generating multi-class classifiers using MCC as the fitness function are presented in Table 9.3. The best classifier achieved a false positive rate of 4% during testing. During training, the sensitivity rate of the best classifier was high, correctly detecting most of the intrusive connections. The best classifier achieved the same U2R detection rate as the best classifier generated using accuracy as the fitness function during training but during testing, the classifier using accuracy outperformed the classifier generated from using MCC in correctly detecting two of the four network attacks.

		DOS	Probe	U2R	R2L
Training	Best	99.78 ± 0.03	98.10 ± 0.08	99.98 ± 0.01	99.67 ± 0.03
	Worst	63.54 ± 0.26	92.49 ± 0.14	99.96 ± 0.01	99.02 ± 0.05
	Average	89.02 ± 0.12	95.23 ± 0.11	99.97 ± 0.01	99.35 ± 0.04

Testing	Best	87.84 ± 0.18	92.43 ± 0.15	99.75 ± 0.03	89.23 ± 0.17
	Worst	66.91 ± 0.26	94.08 ± 0.13	99.70 ± 0.03	87.98 ± 0.18
	Average	81.87 ± 0.20	90.78 ± 0.16	99.73 ± 0.03	88.10 ± 0.18

Table 9.17: GP Matthews's coefficient correlation multi-classification results

9.4.2.3 F-Score

The training and testing results of the classifiers that were generated using f-score as the fitness function for multi-class classification using genetic programming are presented in Table 9.4. The classifiers generated using f-score achieved high accuracy rates with a high detection of U2R attacks as compared to other network attacks. The best classifier achieved a false positive rate of 2% during training and 3% during testing. The classifier generated using the f-score as the fitness measure achieved the lowest false positive rate as compared to the classifiers generated using accuracy and MCC as the fitness measure.

		DOS	Probe	U2R	R2L
Training	Best	99.79 ± 0.03	98.62 ± 0.06	99.98 ± 0.01	99.75 ± 0.03
	Worst	95.16 ± 0.12	92.43 ± 0.15	77.40 ± 0.23	86.63 ± 0.19
	Average	97.91 ± 0.07	96.83 ± 0.06	95.30 ± 0.09	98.85 ± 0.05

Testing	Best	90.31 ± 0.16	90.12 ± 0.16	99.73 ± 0.03	87.27 ± 0.18
	Worst	84.13 ± 0.20	91.37 ± 0.15	70.76 ± 0.25	86.33 ± 0.19
	Average	87.29 ± 0.18	91.27 ± 0.15	93.22 ± 0.11	87.97 ± 0.18

Table 9.18: GP f-score multi-classification results

9.4.2.4 True Positive Rate

Table 9.5 presents the results of using the true positive rate as the fitness function for generating multi-class classifiers using genetic programming. The best classifier achieved a high true positive rate of 96% during training and 92% during testing because the fitness measure favours detection of true positives within a dataset. The overall performance of the classifier was lower because the false positive rate of the classifier was high. The classifier achieved a false positive rate of 16% during training and 20% during testing. The high false positive rate was a result of a low detection rate of intrusive connections.

		DOS	Probe	U2R	R2L
Training	Best	64.36 ± 0.26	92.33 ± 0.15	99.21 ± 0.05	99.21 ± 0.05
	Worst	63.54 ± 0.27	90.75 ± 0.16	96.49 ± 0.10	96.49 ± 0.10
	Average	63.60 ± 0.27	90.82 ± 0.16	99.05 ± 0.05	99.05 ± 0.05

Testing	Best	66.87 ± 0.27	91.55 ± 0.15	87.20 ± 0.18	87.20 ± 0.18
	Worst	66.91 ± 0.26	89.26 ± 0.17	81.71 ± 0.21	81.71 ± 0.20
	Average	66.90 ± 0.26	89.37 ± 0.17	86.82 ± 0.19	86.92 ± 0.19

Table 9.19: GP true positive rate multi-classification results

9.4.2.5 Precision

The results of applying precision as the fitness function for generating multi-class classifiers for intrusion detection are presented in Table 9.6. The best classifier achieved high accuracy rates during training but did not achieve a similar accuracy rate during testing. The best classifier achieved a false positive rate of 6% during training and 9% during testing.

		DOS	Probe	U2R	R2L
Training	Best	90.02 ± 0.16	92.87 ± 0.14	99.96 ± 0.01	99.25 ± 0.05
	Worst	63.54 ± 0.26	90.75 ± 0.16	99.96 ± 0.01	99.20 ± 0.05
	Average	65.59 ± 0.26	90.89 ± 0.16	99.96 ± 0.01	99.21 ± 0.05

Testing	Best	71.97 ± 0.25	89.78 ± 0.17	99.72 ± 0.03	87.37 ± 0.18
	Worst	66.91 ± 0.26	89.26 ± 0.17	99.70 ± 0.03	87.21 ± 0.18
	Average	67.11 ± 0.26	89.29 ± 0.17	99.71 ± 0.03	87.21 ± 0.18

Table 9.20: GP precision multi-classification results

9.4.2.6 False Positive Rate

The training and testing results of the classifiers that were generated using the false positive rate as the fitness function for multi-class classification using genetic programming are presented in Table

9.7. The fitness function favours generating effective classifiers which are similar in structure resulting in both the best and worst classifiers obtaining the similar classification results. The best classifier achieved a low false positive rate of 3% during training and 5% during testing.

		DOS	Probe	U2R	R2L
Training	Best	63.54 ± 0.27	90.75 ± 0.16	99.96 ± 0.01	99.21 ± 0.05
	Worst	63.54 ± 0.27	90.75 ± 0.16	93.38 ± 0.13	99.16 ± 0.05
	Average	63.54 ± 0.27	90.75 ± 0.16	99.49 ± 0.02	99.21 ± 0.05

Testing	Best	66.91 ± 0.26	89.26 ± 0.17	99.70 ± 0.03	87.20 ± 0.18
	Worst	66.91 ± 0.26	89.26 ± 0.17	95.08 ± 0.12	87.19 ± 0.18
	Average	66.91 ± 0.26	89.26 ± 0.17	99.37 ± 0.03	87.20 ± 0.18

Table 9.21: GP false positive rate multi-classification results

9.4.3 Analysis of multi-class classification for the GP approach

Figure 9.3 illustrates the test results of using the different performance measures for evolving multi-class classifiers using GP.

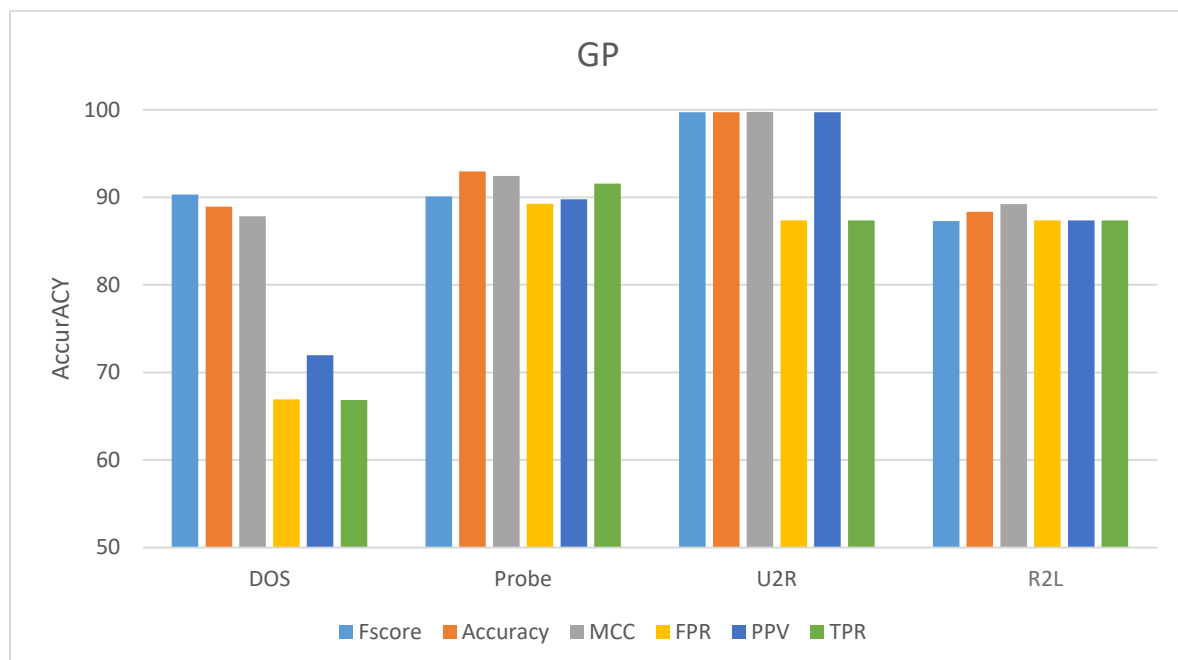


Figure 9.3: GP comparison of fitness function performance

From the results presented for multi-class classification using genetic programming, the classifiers which were generated using the Matthews coefficient correlation as the fitness function achieved higher detection rates than the other classifiers generated using the different fitness functions. The difference in the detection rates were not statistically significant when compared to the results achieved by using f-score and accuracy as fitness functions. It should be noted that even though these results were not statistically significant, the classifiers using the MCC as the fitness function will be used for comparison with other approaches.

9.5 Comparison of GP, GE and MEP

The following sections compare the results produced from using Genetic Programming, Multi-Expression programming and Grammatical Evolution for evolving binary and multi-class classifiers.

9.5.1 Binary classification

The results of the best classifiers for the three approaches GP, GE and MEP described in this thesis are presented in Figure 9.4.

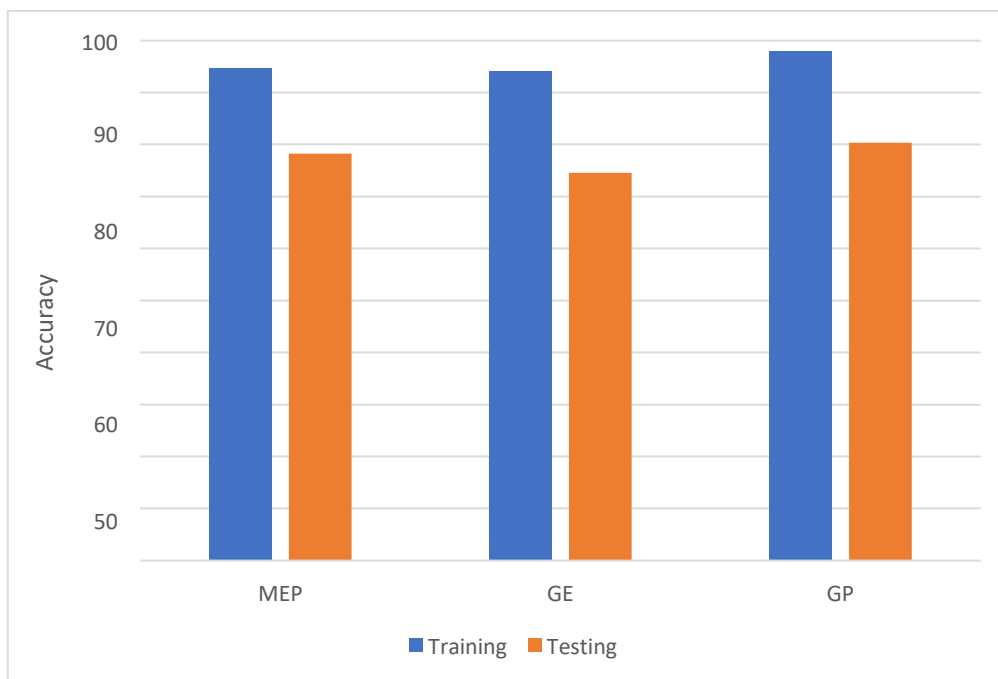


Figure 9.4: Binary classification comparison

From the results presented in the previous sections, the proposed genetic programming approach outperforms the other two approaches for binary classification. The GP binary classifier achieves a high accuracy during both training and testing. MEP achieves similar results compared to GP during testing. The results also show that MEP can generalize well on unseen data. During the comparison of

the three approaches, the GE approach had a lower runtime during training compared to the other two approaches.

9.5.1.1 Statistical Comparison

The section below outlines the statistical tests used to evaluate the statistical significance of the differences in performance of GE, MEP and GP for binary class classification.

The one-tailed Z- test discussed in section 5.4.1 was used to determine the statistical significance. The results of the Z-test were used to determine whether to accept or reject the null hypothesis. The value of Z was calculated and compared to the critical value. If the Z- value was lower than the critical value (1.64), the null hypothesis (H₀) was accepted, otherwise the alternate hypothesis was accepted (H_a). The tests were evaluated at the 0.05 significance level.

Hypothesis 1:

- H₀: There is no difference in the mean objective value for genetic programming and grammatical evolution.
- H_a: The objective mean value for genetic programming is greater than the objective mean value for grammatical evolution.

Hypothesis 2:

- H₀: There is no difference in the mean objective value for genetic programming and multi-expression programming.
- H_a: The objective mean value for genetic programming is greater than the objective mean value for multi-expression programming.

	GP vs GE Test	GP vs MEP Test
Z-Value	24.80	8.60

Table 9.22: Statistical test results for binary classification

Table 9.22 presents the Z-value for Hypothesis 1 and for Hypothesis 2 for binary classification. From the results presented in the table above, the Z-value for GP vs GE test was greater than the critical value resulting in the null hypothesis being rejected. The alternate hypothesis was accepted and from the results of the performance, genetic programming performed significantly better than grammatical evolution resulting in the performance of genetic programming being statistically significant. From the Z-value for GP vs MEP test, the null hypothesis was also rejected and the performance of genetic

programming was significantly better than multi-expression programming performance resulting in the alternate hypothesis being accepted.

9.5.2 Multi-class classification

Figure 9.5 presents a summary of the results achieved from applying grammatical evolution, genetic programming and multi-expression programming for multi-class classification. The results represent a comparison of the best performing classifiers obtained using each of the approaches.

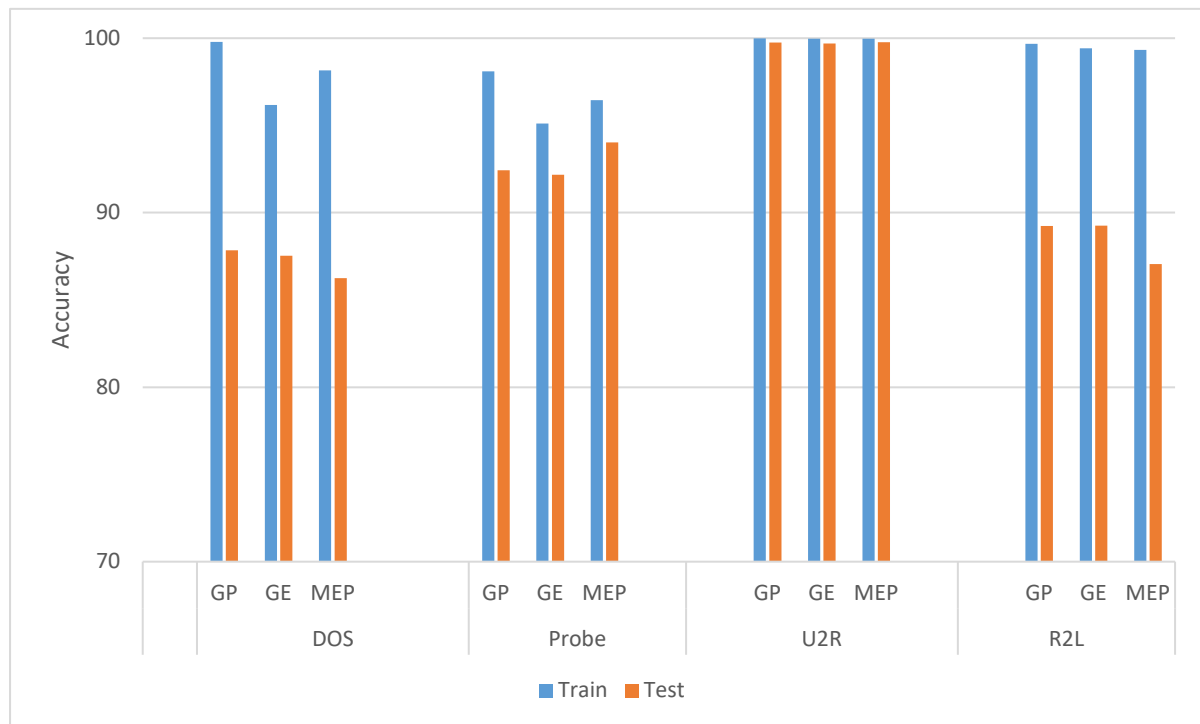


Figure 9.5: Multi-class classification comparison

From the results presented in the previous sections, all the classifiers achieved high detection rates during both training and testing for user to root attacks (U2R) and also achieved high detection rates for remote to user (R2L) during training but achieved lower detection rates during testing. Each approach achieved similar results for detection of the different network attacks. For the detection of probe attacks, the classifier generated using MEP outperforms the other 2 approaches, for the detection of DOS attacks, the GP classifier achieves higher detection rates as compare to the other two approaches and for the detection of R2L attacks, the classifier generated from applying GE achieves a higher detection rates as compared to the other two classifiers. The GE approach averaged 5 hours for generating the classifier whilst MEP averaged 8 hours and GP averaging around 14 hours per run.

9.5.2.1 Statistical Comparison

The section below outlines the statistical tests used to evaluate the statistical significance of the differences in performance of GE, MEP and GP for multi-class classification.

Hypothesis 1:

- H_0 : There is no difference in the mean objective value for genetic programming and grammatical evolution.
- H_a : The objective mean value for genetic programming is greater than the objective mean value for grammatical evolution.

Hypothesis 2:

- H_0 : There is no difference in the mean objective value for genetic programming and multi-expression programming.
- H_a : The objective mean value for genetic programming is greater than the objective mean value for multi-expression programming.

	GP vs GE Test	GP vs MEP Test
Z-Value	-2.70	-3.38

Table 9.23: Statistical test results for multi-class classification

From the results presented in the Table 9.23, the Z-value for GP vs GE test was lower than the critical value resulting in the alternate hypothesis being rejected. The null hypothesis was accepted and from the results of the performance, both the performance of grammatical evolution and genetic programming achieved similar detection rates resulting in the performance of grammatical evolution and genetic programming not being statistically significant. The null hypothesis was also accepted the alternate hypothesis rejected based on the Z-value for GP vs MEP, with both multi-expression programming and genetic programming achieving similar results for intrusion detection.

9.6 Comparison with state of the art

The following sections compare the performance of the proposed approaches to the state-of-the-art methods for network intrusion detection.

9.6.1 Binary Classification

Table 9.23 presents a comparison of the proposed binary classifier described in this thesis with state-of-the-art binary classifiers within network intrusion detection.

The comparison was empirical in nature as a formal performance comparison was not possible due to the different experimental setup applied in each of the studies used for comparison. Furthermore, it was not clear in all of studies whether all records were used or only a subset of the training and testing sets. The proposed approaches performed well with the proposed GP approach outperforming four of the seven approaches used for comparison. The proposed approaches were also compared to other approaches such as Self-Organization Map (SOM) and Support Vector Machine (SVM). The state-of-the-art approaches which applied neural networks for intrusion detection also performed feature selection which resulted in the approaches achieving higher detection rates as compared to the proposed approaches. Different unsupervised and supervised filter approaches such as Random Projection (RP) and Nominal-to-Binary (N2B) were applied by some state-of-the-art approaches such as Naïve Bayes which resulted in the approach achieving higher detection rates as compared to the proposed approaches in this thesis

Approach	Accuracy
Proposed GP	80.30
Proposed MEP	78.23
Proposed GE	74.55
Decision Tree (J48) [85]	81.05
Naïve Bayes [85]	76.56
Support Vector Machine [85]	69.52
Multi-layer Perceptron [85]	77.41
Multinomial naïve Bayes [66]	96.50
Hybrid (Fuzzy logic + GP) [41]	82.74
Self-Organization Map (SOM) [36]	75.49

Table 9.24: State of the art for binary classification.

9.6.2 Multi-class classification

Table 9.24 presents a comparison of the proposed approaches presented in this thesis with state-of-the-art approaches for multi-class classifications. The proposed approaches outperform the state-of-the-art approaches for the detection of U2R attacks. The proposed approaches also outperform some

of the state-of-the-art approaches for the detection of probe attacks and obtain similar results for detecting R2L attacks.

Support Vector Machines outperformed the proposed approaches for detecting DOS attacks and R2L attacks. Random Forests which applied feature selection to determine the 13 most effective features from the 41 features in the dataset outperformed the proposed approaches for detecting all the network attacks.

Approach	Network Attack			
	DOS	Probe	U2R	R2L
Proposed GP	87.84	92.43	99.75	89.23
Proposed GE	87.54	92.18	99.70	89.26
Proposed MEP	86.24	94.03	99.76	87.06
Random Forest [75]	98.70	97.60	97.50	96.80
J48 decision Tree [75]	82.40	80.20	73.90	87.60
Support Vector Machine [75]	97.80	90.70	93.70	91.80
Naïve Bayes [75]	72.70	70.90	70.70	69.80
CART [75]	82.70	82.10	73.10	80.80

Table 9.25: State of the art for multi-class classification

9.7 Chapter Summary

This chapter presented and discussed the results of the proposed approaches discussed in the previous chapters. The investigations discussed in this chapter include determining the best fitness function measure for generating classifiers which can obtain high detection rates for intrusive connections. The chapter also compared the classifiers generated from approaches genetic programming, grammatical evolution and multi-expression programming for the detection of intrusive connections. Furthermore, a comparison between the proposed approaches and state of the art approaches for network intrusion detection was conducted.

10 Conclusion and Future Work

10.1 Introduction

This chapter summarizes the findings of this dissertation and provides a conclusion to each of the objectives outlined in chapter 1. Possible future work based on the research provided in this thesis is also provided.

10.2 Objectives and Conclusion

The section provides a summary of how each of the objectives outlined in Chapter 1 was met and a summary of the findings. Future work based on the objective is also provided.

10.2.1 Effectiveness of using grammatical evolution (GE) for generating intrusion detection classifiers

Grammatical Evolution (GE) was used to evolve intrusion detection classifiers. Few studies from previous literature had successfully applied GE for generating classifiers achieving high detection rates. Experiments presented in this thesis were conducted on the KDD'99 dataset. The results from the experiments revealed that classifiers generated from applying GE for network intrusion detection achieve high detection rates for binary and multi-class classifiers. Multi-class classifiers achieved higher detection rates for R2L attacks than detecting other network attack categories. The classifiers did not outperform some of the state-of-the-art approaches. Future work will investigate different ways to efficiently explore the search space when evolving classifiers using GE.

10.2.2 Development and evaluation of applying multi-expression programming (MEP) for generating binary and multi-class classifiers for network intrusion detection.

Binary and multi-class intrusion detection classifiers were evolved using multi-expression programming. Multi-expression programming (MEP) was used for evolving the classifiers based on the analysis of previous literature. The proof by demonstration methodology was applied to refine the evolved classifiers.

The results from the experiments conducted revealed that binary classifiers evolved using MEP were able to generalize well on unseen data. The implementation also revealed that even though the use for MEP to evolve intrusion detection classifiers has not been common, the approach has the potential to evolve classifiers which can achieve high detection rates. The classifiers generated outperformed some of the state-of-the-art approaches which used the same dataset for binary and multi-class intrusion detection. Future work will investigate different representations of MEP individuals for generating effective intrusion detection classifiers.

10.2.3 Develop and analyze the performance of using genetic programming (GP) for generating binary and multiclass classifiers for network intrusion detection.

Genetic programming (GP) was applied to evolve network intrusion detection (NID) classifiers and evaluate the performance of the classifiers on the publicly available KDD'99 dataset. GP was widely used for generating NID classifiers in previous literature with some of the evolved classifiers achieving high detection rates. The results presented in this thesis show that binary and multi-class classifiers evolved using GP can achieve high detection rates which can outperform other state of the art approaches used in previous literature. Future research will aim to investigate using different representations for GP individuals.

10.2.4 Investigate the effectiveness of fitness functions for network intrusion detection

This study investigated the effectiveness of using different fitness functions for the generating multi-class intrusion detection classifiers. Six fitness functions were proposed based on analysis of previous literature within the network intrusion detection domain. Each of the six fitness functions were applied during the generation of multi-class classifiers. The motivation behind this study was based on different studies applying different fitness functions for the generation of intrusive detection classifiers and achieving different detection rates. Experiments were conducted using the KDD'99 dataset. The results revealed that different fitness functions affected the detection rates of classifiers evolved. Using accuracy, f-score and Matthew's correlation coefficient as fitness functions yielded classifiers which achieved high detection rates.

Future research will aim to investigate the use of weighted fitness functions for generating intrusion detection classifiers. Future work will also include evaluating other fitness functions not commonly used for intrusion detection.

10.2.5 Comparative analysis of GE, MEP and GP for network intrusion detection.

This study compared the performance of using three variants of genetic programming (GE, MEP and GP) to evolve binary and multi-class intrusion detection classifiers. The rationale behind this study was based on an analysis of previous literature in which a comparison of the effect of using different variants of genetic programming for network intrusion detection was not performed.

Binary intrusion detection classifiers evolved using genetic programming obtained high detection rates as compared to binary classifiers evolved using multi-expression programming and grammatical evolution. The results from evolving multi-class classifiers revealed that classifiers evolved using genetic programming achieved high detection rate for detecting DOS attacks, multi-expression programming classifiers achieved high detection rates for detecting Probe attacks and grammatical evolution classifiers achieved high detection rate for detecting R2L attacks. Similar results were achieved for detecting U2R attacks using classifiers evolved using the three approaches. Future research will investigate the generation of a hybridized intrusion detection classifier which combines the different variants of genetic programming.

Bibliography

- 1 <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, accessed 05 March 2015
- 2 Angeline, P.J., and Pollack, J.: 'Evolutionary module acquisition', in Editor (Ed.)^(Eds.): 'Book Evolutionary module acquisition' (Citeseer, 1993, edn.), pp. 154-163
- 3 Banzhaf, W., Nordin, P., Keller, R.E., and Francone, F.D.: 'Genetic programming: an introduction' (Morgan Kaufmann Publishers San Francisco, 1998. 1998)
- 4 Bhavsar, Y.B., and Waghmare, K.C.: 'Intrusion detection system using data mining technique: Support vector machine', International Journal of Emerging Technology and Advanced Engineering, 2013, 3, (3), pp. 581-586
- 5 Blasco, J., Orfila, A., and Ribagorda, A.: 'Improving network intrusion detection by means of domain-aware genetic programming', in Editor (Ed.)^(Eds.): 'Book Improving network intrusion detection by means of domain-aware genetic programming' (IEEE, 2010, edn.), pp. 327-332
- 6 Borovicka, T., Jirina Jr, M., Kordik, P., and Jirina, M.: 'Selecting representative data sets': 'Advances in data mining knowledge discovery and applications' (InTech, 2012)
- 7 Bramer, M.: 'Principles of data mining' (Springer, 2007. 2007)
- 8 Bruce, W.S.: 'The application of genetic programming to the automatic generation of object-oriented programs', 1995
- 9 Chae, H.-s., Jo, B.-o., Choi, S.-H., and Park, T.: 'Feature Selection for Intrusion Detection using NSL-KDD', Recent Advances in Computer Science, ISBN, 2015, pp. 978-960
- 10 Chebrolu, S., Abraham, A., and Thomas, J.P.: 'Feature deduction and ensemble design of intrusion detection systems', Computers & Security, 2005, 24, pp. 295e307
- 11 Chittur, A.: 'Model generation for an intrusion detection system using genetic algorithms', Internet link: <http://www1.cs.columbia.edu/ids/publications/gaids-thesis01.pdf>, 2001
- 12 Cios, K.J., Swiniarski, R.W., Pedrycz, W., and Kurgan, L.A.: 'The knowledge discovery process', in Editor (Ed.)^(Eds.): 'Book The knowledge discovery process' (Springer, 2007, edn.), pp. 9-24
- 13 Cleary, R.: 'Extending grammatical evolution with attribute grammars: An application to knapsack problems', Master of science thesis in computer science, University of Limerick, Ireland, 2005
- 14 Črepinšek, M., Liu, S.-H., and Mernik, M.: 'Exploration and exploitation in evolutionary algorithms: A survey', ACM Computing Surveys (CSUR), 2013, 45, (3), pp. 35

- 15 Crosbie, M., and Spafford, G.: 'Applying genetic programming to intrusion detection', in Editor (Ed.)^(Eds.): 'Book Applying genetic programming to intrusion detection' (MIT, Cambridge, MA, USA: AAAI, 1995, edn.), pp. 1-8
- 16 Cunningham, R.K., Lippmann, R.P., Fried, D.J., Garfinkel, S.L., Graf, I., Kendall, K.R., Webster, S.E., Wyschogrod, D., and Zissman, M.A.: 'Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation', in Editor (Ed.)^(Eds.): 'Book Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation' (DTIC Document, 1999, edn.), pp.
- 17 Dash, M., and Liu, H.: 'Feature selection for classification', *Intelligent data analysis*, 1997, 1, (3), pp. 131-156
- 18 Debar, H., Dacier, M., and Wespi, A.: 'Towards a taxonomy of intrusion-detection systems', *Computer Networks*, 1999, 31, (8), pp. 805-822
- 19 Dempsey, I., O'Neill, M., and Brabazon, A.: 'Foundations in grammatical evolution for dynamic environments' (Springer, 2009. 2009)
- 20 Dufourq, E., and Pillay, N.: 'A comparison of genetic programming representations for binary data classification', in Editor (Ed.)^(Eds.): 'Book A comparison of genetic programming representations for binary data classification' (IEEE, 2013, edn.), pp. 134-140
- 21 Eskin, E., Arnold, A., Prerau, M., Portnoy, L., and Stolfo, S.: 'A geometric framework for unsupervised anomaly detection': 'Applications of data mining in computer security' (Springer, 2002), pp. 77-101
- 22 Espejo, P.G., Ventura, S., and Herrera, F.: 'A survey on the application of genetic programming to classification', *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 2010, 40, (2), pp. 121-144
- 23 Faraoun, K., and Boukelif, A.: 'Genetic programming approach for multi-category pattern classification applied to network intrusions detection', *International Journal of Computational Intelligence and Applications*, 2006, 6, (01), pp. 77-99
- 24 Fawcett, T.: 'An introduction to ROC analysis', *Pattern recognition letters*, 2006, 27, (8), pp. 861-874
- 25 Fisher, R.A.: 'The genetical theory of natural selection: a complete variorum edition' (Oxford University Press, 1930. 1930)
- 26 Gogoi, P., Bhuyan, M.H., Bhattacharyya, D., and Kalita, J.K.: 'Packet and flow based network intrusion dataset': 'Contemporary Computing' (Springer, 2012), pp. 322-334

- 27 Gong, R.H., Zulkernine, M., and Abolmaesumi, P.: 'A software implementation of a genetic algorithm based approach to network intrusion detection', in Editor (Ed.)^(Eds.): 'Book A software implementation of a genetic algorithm based approach to network intrusion detection' (IEEE, 2005, edn.), pp. 246-253
- 28 Govindarajan, M., and Chandrasekaran, R.: 'Intrusion detection using an ensemble of classification methods', in Editor (Ed.)^(Eds.): 'Book Intrusion detection using an ensemble of classification methods' (2012, edn.), pp.
- 29 Groşan, C., Abraham, A., and Han, S.Y.: 'MEPIDS: Multi-expression programming for intrusion detection system': 'Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach' (Springer, 2005), pp. 163-172
- 30 Gu, G., Fogla, P., Dagon, D., Lee, W., and Skoric, B.: 'Towards an information-theoretic framework for analyzing intrusion detection systems', in Editor (Ed.)^(Eds.): 'Book Towards an information-theoretic framework for analyzing intrusion detection systems' (Springer, 2006, edn.), pp. 527-546
- 31 Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I.H.: 'The WEKA data mining software: an update', ACM SIGKDD explorations newsletter, 2009, 11, (1), pp. 10-18
- 32 Hansen, J.V., Lowry, P.B., Meservy, R.D., and McDonald, D.M.: 'Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection', Decision Support Systems, 2007, 43, (4), pp. 1362-1374
- 33 Harper, R., and Blair, A.: 'Dynamically defined functions in grammatical evolution', in Editor (Ed.)^(Eds.): 'Book Dynamically defined functions in grammatical evolution' (IEEE, 2006, edn.), pp. 2638-2645
- 34 Hemberg, M., and O'Reilly, U.-M.: 'Extending grammatical evolution to evolve digital surfaces with genr8': 'Genetic Programming' (Springer, 2004), pp. 299-308
- 35 Hoque, M.S., Mukit, M., Bikas, M., and Naser, A.: 'An implementation of intrusion detection system using genetic algorithm', arXiv preprint arXiv:1204.1336, 2012
- 36 Ibrahim, L.M., Basheer, D.T., and Mahmood, M.S.: 'A comparison study for intrusion database (Kdd99, Nsl-Kdd) based on self organization map (SOM) artificial neural network', Journal of Engineering Science and Technology, 2013, 8, (1), pp. 107-119
- 37 Johnson, C.: 'Basic Research Skills in Computing Science', Department of Computer Science, Glasgow University, UK, 2006

- 38 Kendall, K.: 'A database of computer attacks for the evaluation of intrusion detection systems', in Editor (Ed.)^(Eds.): 'Book A database of computer attacks for the evaluation of intrusion detection systems' (DTIC Document, 1999, edn.), pp.
- 39 Koza, J.R.: 'Genetic programming II: Automatic discovery of reusable subprograms', Cambridge, MA, USA, 1994
- 40 Koza, J.R.: 'Genetic programming: on the programming of computers by means of natural selection' (MIT press, 1992. 1992)
- 41 Krömer, P., Platoš, J., Snáael, V., and Abraham, A.: 'Fuzzy classification by evolutionary algorithms', in Editor (Ed.)^(Eds.): 'Book Fuzzy classification by evolutionary algorithms' (IEEE, 2011, edn.), pp. 313-318
- 42 Langdon, W.: 'Genetic Programming and Data Structures: Genetic Programming+ Data Structures= Automatic Programming!' (Springer Science & Business Media, 1998. 1998)
- 43 Lazarevic, A., Kumar, V., and Srivastava, J.: 'Intrusion detection: A survey': 'Managing Cyber Threats' (Springer, 2005), pp. 19-78
- 44 Lee, W., Stolfo, S.J., and Mok, K.W.: 'A data mining framework for building intrusion detection models', in Editor (Ed.)^(Eds.): 'Book A data mining framework for building intrusion detection models' (IEEE, 1999, edn.), pp. 120-132
- 45 Li, W.: 'Using genetic algorithm for network intrusion detection', Proceedings of the United States Department of Energy Cyber Security Group, 2004, pp. 1-8
- 46 Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., and Das, K.: 'The 1999 DARPA off-line intrusion detection evaluation', Computer networks, 2000, 34, (4), pp. 579-595
- 47 Loveard, T., and Ciesielski, V.: 'Representing classification problems in genetic programming', in Editor (Ed.)^(Eds.): 'Book Representing classification problems in genetic programming' (IEEE, 2001, edn.), pp. 1070-1077
- 48 Lu, W., and Traore, I.: 'Detecting new forms of network intrusion: using genetic programming', Computational Intelligence, 2004, 20, (3), pp. 475-494
- 49 Luke, S., and Panait, L.: 'A comparison of bloat control methods for genetic programming', Evolutionary Computation, 2006, 14, (3), pp. 309-344
- 50 Mahoney, M.V., and Chan, P.K.: 'An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection', in Editor (Ed.)^(Eds.): 'Book An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection' (Springer, 2003, edn.), pp. 220-237

- 51 McHugh, J.: 'Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory', *ACM transactions on Information and system Security*, 2000, 3, (4), pp. 262-294
- 52 Miller, B.L., and Goldberg, D.E.: 'Genetic algorithms, tournament selection, and the effects of noise', *Complex systems*, 1995, 9, (3), pp. 193-212
- 53 Mitchell, M.: 'An introduction to genetic algorithms' (1998. 1998)
- 54 Moradi, M., and Zulkernine, M.: 'A neural network based system for intrusion detection and classification of attacks', in Editor (Ed.)^(Eds.): 'Book A neural network based system for intrusion detection and classification of attacks' (2004, edn.), pp.
- 55 Mukherjee, S., and Sharma, N.: 'Intrusion detection using naive Bayes classifier with feature reduction', *Procedia Technology*, 2012, 4, pp. 119-128
- 56 Mukkamala, S., Sung, A.H., and Abraham, A.: 'Modeling intrusion detection systems using linear genetic programming approach': 'Innovations in Applied Artificial Intelligence' (Springer, 2004), pp. 633-642
- 57 Nordin, P., Francone, F., and Banzhaf, W.: 'Explicitly defined introns and destructive crossover in genetic programming', *Advances in genetic programming*, 1995, 2, pp. 111-134
- 58 O'Neill, M., and Ryan, C.: 'Grammar based function definition in Grammatical Evolution', in Editor (Ed.)^(Eds.): 'Book Grammar based function definition in Grammatical Evolution' (Morgan Kaufmann Publishers Inc., 2000, edn.), pp. 485-490
- 59 O'Neill, M., and Ryan, C.: 'Grammatical evolution', *IEEE Transactions on Evolutionary Computation*, 2001, 5, (4), pp. 349-358
- 60 O'Neill, M., and Ryan, C.: 'Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language' (Springer Science & Business Media, 2003. 2003)
- 61 Oltean, M.: 'Multi expression programming', 2006
- 62 Oltean, M., and Dumitrescu, D.: 'Multi Expression Programming', 2002
- 63 Oltean, M., and Grosan, C.: 'A comparison of several linear genetic programming techniques', *Complex Systems*, 2003, 14, (4), pp. 285-314
- 64 Orfila, A., Estevez-Tapiador, J.M., and Ribagorda, A.: 'Evolving high-speed, easy-to-understand network intrusion detection rules with genetic programming': 'Applications of Evolutionary Computing' (Springer, 2009), pp. 93-98
- 65 Özgür, A., and Erdem, H.: 'A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015', *PeerJ Preprints*, 2016, 4, pp. e1954v1951

- 66 Panda, M., Abraham, A., and Patra, M.R.: 'Discriminative multinomial naive bayes for network intrusion detection', in Editor (Ed.)^(Eds.): 'Book Discriminative multinomial naive bayes for network intrusion detection' (IEEE, 2010, edn.), pp. 5-10
- 67 Panda, M., and Patra, M.R.: 'Network intrusion detection using naive bayes', International journal of computer science and network security, 2007, 7, (12), pp. 258-263
- 68 Pang, R., Allman, M., Bennett, M., Lee, J., Paxson, V., and Tierney, B.: 'A first look at modern enterprise traffic', in Editor (Ed.)^(Eds.): 'Book A first look at modern enterprise traffic' (USENIX Association, 2005, edn.), pp. 2-2
- 69 Pastrana, S., Orfila, A., and Ribagorda, A.: 'A functional framework to evade network IDS', in Editor (Ed.)^(Eds.): 'Book A functional framework to evade network IDS' (IEEE, 2011, edn.), pp. 1-10
- 70 Pastrana, S., Orfila, A., and Ribagorda, A.: 'Modeling NIDS evasion with genetic programming', 2010
- 71 Paxson, V.: 'Bro: a system for detecting network intruders in real-time', Computer networks, 1999, 31, (23), pp. 2435-2463
- 72 Pfleeger, C.P., and Pfleeger, S.L.: 'Security in computing' (Prentice Hall Professional Technical Reference, 2002. 2002)
- 73 Pillay, N.: 'An Investigation into the Use of Genetic Programming for the Induction of Novice Procedural Programming Solution Algorithms in Intelligent Programming Tutors', University of Natal, 2004
- 74 Poli, R., Langdon, W.B., McPhee, N.F., and Koza, J.R.: 'A field guide to genetic programming' (Lulu. com, 2008. 2008)
- 75 Revathi, S., and Malathi, A.: 'A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection', in Editor (Ed.)^(Eds.): 'Book A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection' (Citeseer, 2013, edn.), pp.
- 76 Ryan, C., Collins, J., and Neill, M.O.: 'Grammatical evolution: Evolving programs for an arbitrary language': 'Genetic Programming' (Springer, 1998), pp. 83-96
- 77 Sabhnani, M., and Serpen, G.: 'Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set', Intelligent Data Analysis, 2004, 8, (4), pp. 403-415
- 78 Scarfone, K., and Mell, P.: 'Guide to intrusion detection and prevention systems (idps)', NIST special publication, 2007, 800, (2007), pp. 94

- 79 Şen, S., and Clark, J.A.: 'A grammatical evolution approach to intrusion detection on mobile ad hoc networks', in Editor (Ed.)^(Eds.): 'Book A grammatical evolution approach to intrusion detection on mobile ad hoc networks' (ACM, 2009, edn.), pp. 95-102
- 80 Siddiqui, M.A.: 'High performance data mining techniques for intrusion detection', 2004
- 81 Sivanandam, S., and Deepa, S.: 'Introduction to genetic algorithms' (Springer Science & Business Media, 2007. 2007)
- 82 Sokolova, M., and Lapalme, G.: 'A systematic analysis of performance measures for classification tasks', *Information Processing & Management*, 2009, 45, (4), pp. 427-437
- 83 Song, D., Heywood, M.I., and Zincir-Heywood, A.N.: 'Training genetic programming on half a million patterns: an example from anomaly detection', *Evolutionary Computation, IEEE Transactions on*, 2005, 9, (3), pp. 225-239
- 84 Sung, A.H., and Mukkamala, S.: 'Identifying important features for intrusion detection using support vector machines and neural networks', in Editor (Ed.)^(Eds.): 'Book Identifying important features for intrusion detection using support vector machines and neural networks' (IEEE, 2003, edn.), pp. 209-216
- 85 Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A.-A.: 'A detailed analysis of the KDD CUP 99 data set', in Editor (Ed.)^(Eds.): 'Book A detailed analysis of the KDD CUP 99 data set' (2009, edn.), pp.
- 86 Thaseen, S., and Kumar, C.A.: 'An analysis of supervised tree based classifiers for intrusion detection system', in Editor (Ed.)^(Eds.): 'Book An analysis of supervised tree based classifiers for intrusion detection system' (IEEE, 2013, edn.), pp. 294-299
- 87 Wang, G., Hao, J., Ma, J., and Huang, L.: 'A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering', *Expert Systems with Applications*, 2010, 37, (9), pp. 6225-6232
- 88 Wilson, D., and Kaur, D.: 'Using grammatical evolution for evolving intrusion detection rules', *WSEAS Transactions on Systems*, 2007, 6, (2), pp. 346
- 89 Wu, S.X., and Banzhaf, W.: 'The use of computational intelligence in intrusion detection systems: A review', *Applied Soft Computing*, 2010, 10, (1), pp. 1-35
- 90 Yin, C., Tian, S., Huang, H., and He, J.: 'Applying genetic programming to evolve learned rules for network anomaly detection': 'Advances in Natural Computation' (Springer, 2005), pp. 323-331

- 91 Zhang, F., and Wang, D.: 'An effective feature selection approach for network intrusion detection', in Editor (Ed.)^(Eds.): 'Book An effective feature selection approach for network intrusion detection' (IEEE, 2013, edn.), pp. 307-311
- 92 Zincir-Heywood, A.N., and Heywood, M.I.: 'Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets', in Editor (Ed.)^(Eds.): 'Book Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets' (Citeseer, edn.), pp.

A. User Manual

Program requirements

In order to run the NID system, Java must be installed. Java can be obtained from the following website (<https://java.com/en/download/>). Once Java has been installed the program can be used.

Initialising the Program

The program can be started by executing the **NetworkIntrusionDetection.jar** located in the SYSTEM folder on the CD. The main menu will appear as shown in the Figure A.1.

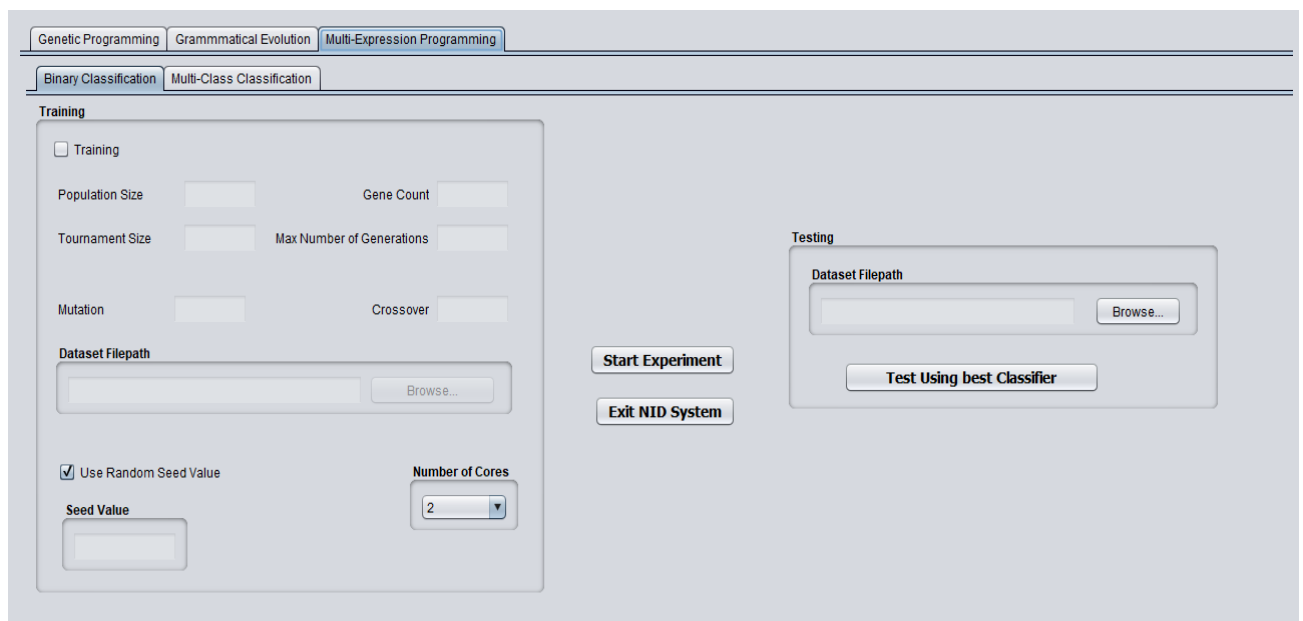


Figure A.1: Network Intrusion Detection System Main Menu

Overview of the program

The Top-level tab menu is made up of the Genetic Programming approaches discussed in this thesis. Each top-level tab menu has two lower level tab menu binary classification and multi-class classification.

- **Genetic Programming** – corresponds to using Genetic Programming for Network Intrusion Detection described in Chapter 6.
- **Grammmatical Evolution** – corresponds to using Grammmatical Evolution for Network Intrusion Detection described in Chapter 7.
- **Multi-Expression Programming** – corresponds to using Multi-Expression Programming for network Intrusion Detection described in Chapter 8.

Experiment Configurations

Two configurations are provided for interacting with the program. Each configuration requires either a training dataset, testing dataset or both training and testing dataset. These datasets are located in the Datasets folder on the CD provided. The datasets are divided into Binary Classification datasets and Multi-class datasets. The multi-class datasets are split based on the attack category described in Chapter 5.

- **Train and Test**

All the parameters must be entered before the run can start as illustrated in Figure A.2. After training and testing datasets have been selected. The run will begin by selecting **Start Experiment**. The run will continue running in the background for the duration of the experiment until it has completed and a popup message has appeared. The experiments use the distributed architecture discussed in Chapter 5.

The screenshot shows a software interface for configuring an experiment. At the top, there are three tabs: 'Genetic Programming', 'Grammatical Evolution', and 'Multi-Expression Programming'. Below these are two sub-tabs: 'Binary Classification' and 'Multi-Class Classification'. The main area is divided into 'Training' and 'Testing' sections. The 'Training' section includes a checked 'Training' checkbox, several input fields for parameters (Population Size: 200, Initial Tree Depth: 7, Tournament Size: 7, Max Number of Generations: 100, Max Offspring Depth: 5, Max Mutation Depth: 4, Mutation: 45, Crossover: 55), a 'Dataset Filepath' field with a 'Browse...' button, a checked 'Use Random Seed Value' checkbox, a 'Seed Value' field with the value '0681334179427', and a 'Number of Cores' dropdown menu set to '2'. The 'Testing' section includes a 'Dataset Filepath' field with the value 'ClassificationNormal+IntrusionKDD_Test.nidTest' and a 'Browse...' button. At the bottom, there are three buttons: 'Start Experiment', 'Exit NID System', and 'Test Using best Classifier'.

Figure A.2: Train and Test using NID System

Once training and testing has completed a popup message will appear as in illustrated in Figure A.3. This message indicates the location of the output file. The output file contains information about run, and performance measures for the evolved classifier.

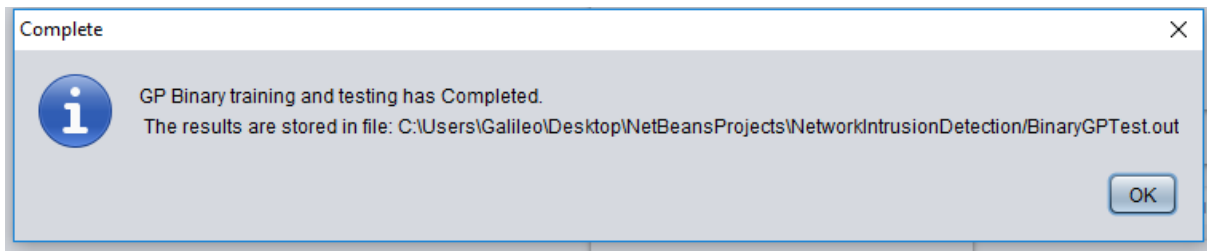


Figure A.3: End of run Message

- **Test using best classifier**

This Configuration is used to evaluate the performance of pre-defined classifiers on different datasets. For example, if you want to evaluate just how well the best classifier achieved from using Genetic Programming for Multi-class classification, the user selects the test dataset using the File chooser option invoked when “Browse” on the Testing panel is selected.

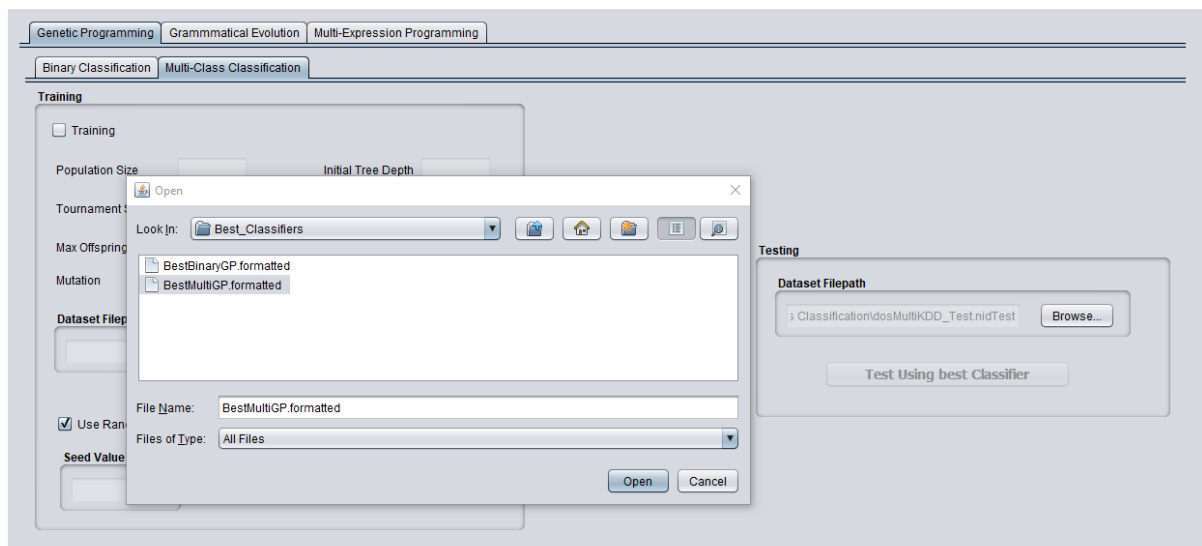


Figure A.4: Selecting best classifier

After selecting a test dataset, click “Test Using best Classifier” which will open a file choose dialog and navigate to **.../Datasets/Best_Classifiers** folder which contains all the best classifiers for the experiments conducted in this thesis. Once the preferred classifier has been selected the system will evaluate the performance of the classifier on the dataset and write the results to an output file. Figure A.4 and Figure A.5 illustrate test using best classifier for multi-class genetic programming for DOS

attacks.

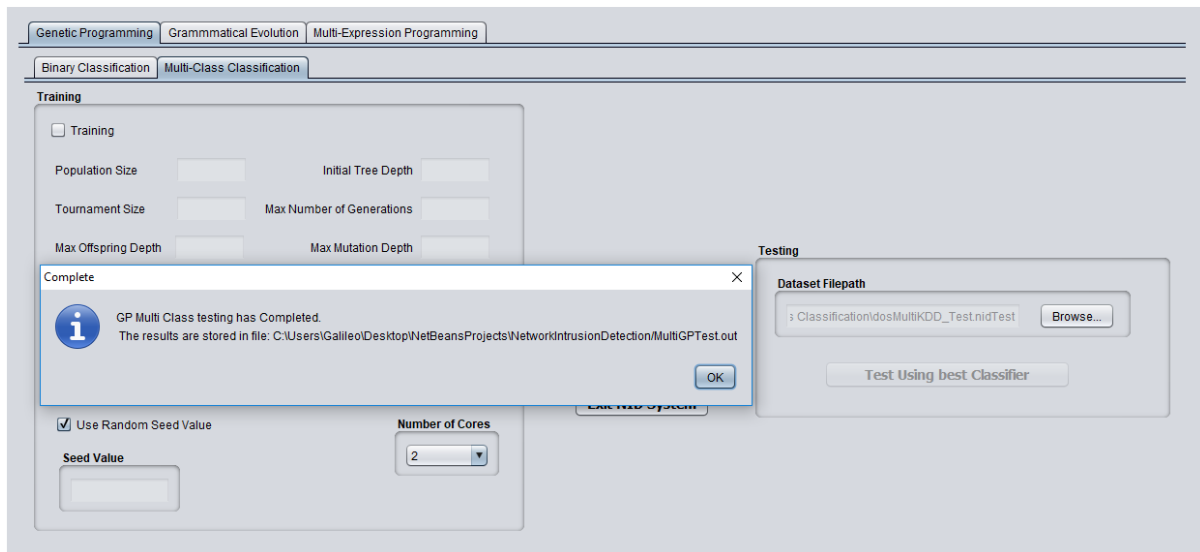


Figure A.5: Results of evaluation