Developing Efficient and Effective Intrusion Detection System using Evolutionary Computation

Hasanen Alyasiri

PhD

University of York

Computer Science

November 2018

Abstract

The internet and computer networks have become an essential tool in distributed computing organisations especially because they enable the collaboration between components of heterogeneous systems. The efficiency and flexibility of online services have attracted many applications, but as they have grown in popularity so have the numbers of attacks on them. Thus, security teams must deal with numerous threats where the threat landscape is continuously evolving. The traditional security solutions are by no means enough to create a secure environment, intrusion detection systems (IDSs), which observe system works and detect intrusions, are usually utilised to complement other defence techniques. However, threats are becoming more sophisticated, with attackers using new attack methods or modifying existing ones. Furthermore, building an effective and efficient IDS is a challenging research problem due to the environment resource restrictions and its constant evolution. To mitigate these problems, we propose to use machine learning techniques to assist with the IDS building effort.

In this thesis, Evolutionary Computation (EC) algorithms are empirically investigated for synthesising intrusion detection programs. EC can construct programs for raising intrusion alerts automatically. One novel proposed approach, i.e. Cartesian Genetic Programming, has proved particularly effective. We also used an ensemble-learning paradigm, in which EC algorithms were used as a meta-learning method to produce detectors. The latter is more fully worked out than the former and has proved a significant success. An efficient IDS should always take into account the resource restrictions of the deployed systems. Memory usage and processing speed are critical requirements. We apply a multi-objective approach to find trade-offs among intrusion detection capability and resource consumption of programs and optimise these objectives simultaneously. High complexity and the large size of detectors are identified as general issues with the current approaches. The multi-objective approach is used to evolve Pareto fronts for detectors that aim to maintain the simplicity of the generated patterns. We also investigate the potential application of these algorithms to detect unknown attacks.

Contents

List of Tables

List of Figures

Acknowledgements

I would first like to express uttermost gratitude to my supervisor, John A. Clark for all his support and advice throughout his supervision period and after it officially ended.

I am extremely grateful to Daniel Kudenko for his support and guidance whenever it was necessary.

I would like to acknowledge my other thesis-committee members, Howard Chivers, Siamak Fayyaz Shahandashti and Julio Hernandez-Castro, for their invaluable and insightful comments, which improved this thesis in many ways.

I would like to thank the Iraqi Ministry of Higher Education and Scientific Research, as represented by the Iraqi Cultural Attaché in London, and the University of Kufa for supporting my PhD studies.

Also, I am very thankful to my friends for their moral support and the open source software community.

Finally, I would like to thank my family for their unconditional love, constant support, encouragement and motivation, which enabled me to overcome any difficulties that I encountered during my research.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Contributions from this thesis have been published elsewhere as follows:

- Hasanen Alyasiri, John A Clark, and Daniel Kudenko, "Applying cartesian genetic programming to evolve rules for intrusion detection system," in 10th International Joint Conference on Computational Intelligence, SciTePress, 2018, pp. 176–183.
- Hasanen Alyasiri, John A Clark, and Daniel Kudenko, "Evolutionary computation algorithms for detecting known and unknown attacks," in 11th International Conference on Security for Information Technology and Communications, Springer, 2018, pp. 170–184.

Chapter 1 | Introduction

1.1 Internet Environment

The internet is currently a popular subject with application developers, information technology architects, and researchers. It is fast becoming a key instrument in present-day societies and economies by providing fast and flexible information sharing among people and businesses. Its underpinning technologies provide a number of attractive features: ease of use, platform and language independence, interoperability and so on. This is particularly important for enabling the collaboration between components of heterogeneous systems. It is rapidly evolving and expanding to adopt new computing and communication prototypes, for instance cloud computing, Internet of Things (IoT) devices and smart portable platforms. Cloud computing services enable customers to access shared system components (i.e. hardware and software) from a remotely located site and utilise them on-demand [1]. IoT comprises physical objects equipped with hardware and software capabilities that will allow them to interact with one another and with other devices and services over the internet to achieve some purpose [2]. Portable devices have facilitated access from almost everywhere. In addition to personal usage, corporations and governments have come to be increasingly reliant on cyberspace for their everyday tasks. Various activities like communications, business and financial transactions, and even the management of vital infrastructure are now performed over the internet. Until very recently, in most cases, carefully restricted and secured media were used to execute most of these tasks. There were some concerns related to spying and data loss. However, this was limited since systems' infrastructures were usually inaccessible from the outside. Nowadays, most organisations are connected to the internet permanently, and the majority of them are migrating their computing infrastructures to the cloud.

The complexity of the internet landscape grows continually, introducing new functionalities and mixing technologies with great speed. For instance, web applications gave millions of users the opportunity to access vast amounts of data and services in a wide range of areas:

Banking and Financial Services: Web applications have changed the economic sector in various aspects. For instance, a great deal of day-to-day banking is now done online. Cheques are going out of fashion (and the banks would largely like to see their demise). Furthermore, financial exchanges, such as currency exchanges or trading portals, have been transformed.
E-commerce: The internet retail is the activity of buying or selling of products over the internet; it has changed how consumers shop. Typically, e-commerce transactions include the purchase of physical or digital products online, such as Amazon.
Education and Science: The E-learning paradigm had a huge impact on the learning process where it has helped to achieve a more effective delivery of knowledge. It eases the access to online courses and digital libraries offering scientific contents at any time.
Entertainment: Web applications have helped in the evolution of the entertainment field with the introduction of blogs, multimedia streaming and news feeds. The gaming industry has a major commitment to the internet, where it has created communities dedicated to playing specific online games over it.
Communications: Webinars are an example of how web applications have changed communications. Another example is email, and how individuals' interactions are conceived differently since social media emerged.
Other areas: These include established areas such as file downloading and sharing and, more recently, the Internet of Things applications.

As reported by the Internet World Stats [3], internet users constitute more than 55.1% of the world population. Figure 1.1 shows the growth in the number of internet users over the last years. This incessant growth implies an increase in the amount of data generated online, and the services that run over the internet have become major targets for cyberattack.

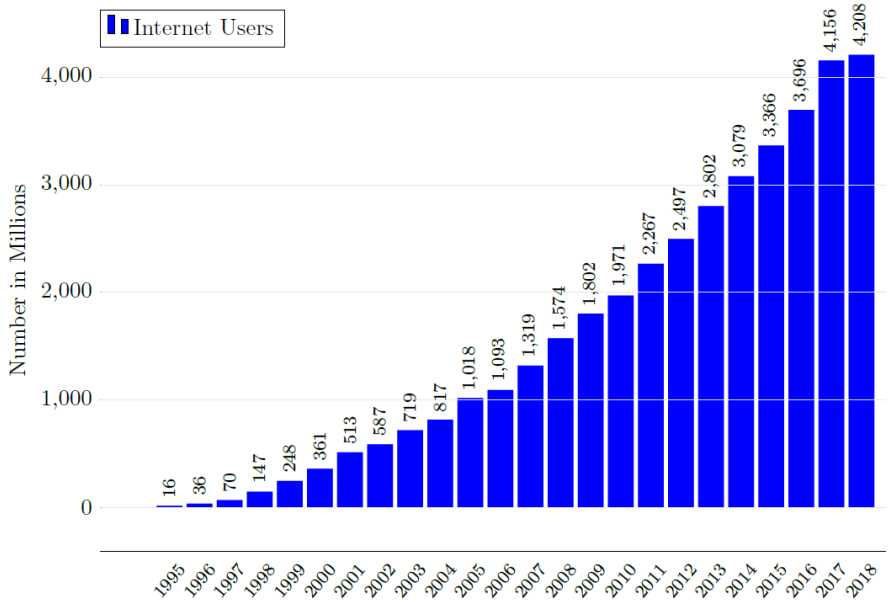

Figure 1.1: Number of Internet Users in the World over the Last Years [3]

1.2 The Security Problem

In the modern threat landscape, the internet and computer networks have been a very significant security focus over the last two decades. Cyberspace security is a group of tools and procedures intended to defend computers, networks, software and data files from attack, illegal access, alteration or damage. Intrusions are known as attempts to undermine the confidentiality, integrity, or availability of a system [4]. How to identify intrusions effectively and efficiently in distributed, resource restricted and constantly evolving environments is a difficult task. In the presence of such complexity, humans are not usually able to produce adequate design decisions. Furthermore, intruders' profile, aims and skills have also changed considerably.

In the recent past, cyber-intruders were considered to usually be young socially isolated individuals [5] driven by a small number of motivations including curiosity, the thrill of the illicit and/or (for more advanced hackers) peer recognition. Although most were very talented, these intruders did not possess sufficient financial means to produce innovative attacks. However, these days sophisticated attacks and motivations have developed. For example, we now see the use of Advanced Persistent Threats [6] utilising multiple advanced techniques such as zero-day exploits with social engineering. This would help them bypass security tools and maintain a presence on the attacked system for long-term control and data collection. Furthermore, instead of separated obsessed individuals, intruders have evolved into well-resourced groups with economic or politi-

cal motives, attacking high-profile infrastructure from governments to big corporations. Symantec's "Internet Security Thread Report" [7] for 2018 revealed that 97% of attacks analysed have one motive or more. These known motives were 90% intelligence gathering, 11% disruption and 9% financial. Risks have recently become more diverse and it is difficult for security teams to keep up with the daily appearance of vulnerabilities, threats, attacks and responses using appropriate countermeasures. For instance, Figure 1.2 shows the number of publicly identified cybersecurity vulnerabilities per year since 1999 as reported by the Common Vulnerabilities and Exposures database [8]. In comparison to 2008, the number of discovered vulnerabilities has increased by a factor of 3 resulting in an average of 45 new vulnerabilities per day.

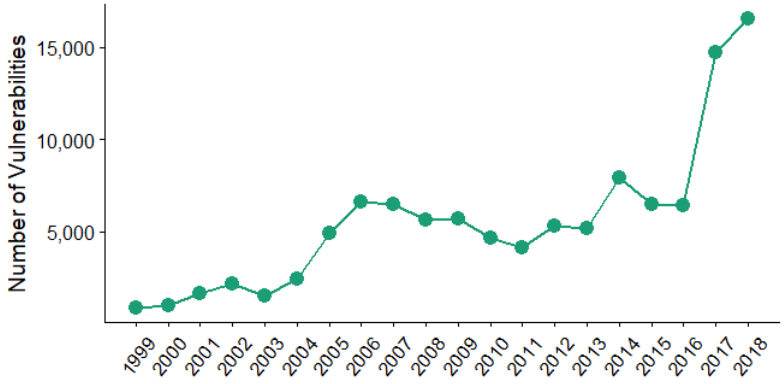

Figure 1.2: Number of New Software Vulnerabilities Reported per Year [8]

Security threats against network infrastructure can be categorised as active and passive [9][10]. Concerning active threats, attackers try to disturb or terminate the targeted host or network operations using, for example, worms, ransomware and denial of service. Whereas, for passive threats, intruders stay hidden and focus on obtaining the information exchanged, for instance, traffic analysis and eavesdropping. In some cases, malware, e.g. a keylogger, may be installed to passively monitor and leak information but carry out no immediately disruptive action.

For example, a recent advanced attack occurred in October 2016. An IoT botnet was used to attack *Dyn*, a pioneer internet infrastructure firm. In this attack, a multi-vector malware called *Mirai* was developed to take advantage of the vulnerabilities of IoT devices (mainly IP cameras) and infect them. These devices were turned into an IoT zombie army that was utilised to carry out massive distributed denial of services attacks against Dyn making its services unreachable for hours for many clients such as CNN, Netflix, Twitter and other sites, despite it being up and running normally [11]. Another attack called *WannaCry* hit many major infrastructures worldwide including many of England's National Health Trusts in May 2017. WannaCry had a huge impact

and the potential to be hugely profitable. Thousands of computers at hospitals were infected by this global ransomware attack within hours of its release. The main reason for this was that it exploits two previously known vulnerabilities in Windows operating systems. Furthermore, the ransomware turns into a worm, capable of spreading itself to any vulnerable systems on the victim's network as well as to other unpatched systems connected to the internet [7].

To provide a comprehensive threat analysis, security experts should not address only network and server layer issues but also threats to the application layer. Traditional protection tools, such as firewalls, security policies, access control and encryption have failed to protect networks and systems adequately from increasingly sophisticated attacks and malware [12]. For instance, Cisco threat researchers [13] analysed more than 400,000 malicious binaries and found that about 70% had used at least some encryption as of October 2017. In addition, security teams have major difficulties handling large volumes of data. The main aim is reducing the risk of losses as far as is practical. The circumstances require more intelligent countermeasures to maintain the security of networks and important systems. Machine Learning (ML) has had success in many areas of computer science, where it was adopted in real world applications such as products recommendation, optical character recognition, and the like [14]. Thus, recently security teams considered adopting ML to rise above the chaos of the war with attackers. The automatic classification of malicious behaviours on the internet holds enormous potential for improving protection. In addition, the extracted knowledge allows discovery of new attack methods, intrusion scenarios and attacker's objectives and strategies, and can enhance the ability to distinguish between attacks and legitimate requests.

1.3 Thesis Hypotheses

The internet is a global network that comprises many properties that are complicated and ill-understood. Humans may possibly benefit from enhanced support to cope with such complexity. Therefore, we intend to research the usage of ML algorithms to explore the IDS building process more efficiently than a human could. In particular, we aim to investigate the use of a branch of ML, namely Evolutionary Computation (EC). The research community has confidence that the potential is good, for example: "Computational intelligence will play important roles for cyber intelligence - tracking, analysing, identifying digital security threats to combat viruses, hackers and terrorists that exist on the Internet for different purposes, including IoT cyber threats, cyberstalking and harassment, extortion, blackmail, stock market manipulation, complex corporate espionage, and planning or carrying out terrorist activities" [15]. EC approaches have a

number of attractive features, such as producing readable outputs, evolving lightweight solutions and supplying a collection of solutions with a variety of trade-offs among often conflicting objectives. We believe that EC can act as a framework for automatic discovery of insights and intrusion detectors. Our first formal hypothesis is therefore:

Hypothesis 1*: evolutionary computation will be able to produce intrusion detection programs for the internet and computer networks based environments. Programs evolved using Genetic Programming, Grammatical Evolution and Cartesian Genetic Programming techniques will be able to detect various known and unknown attacks targeting such environments effectively.*

As the number and sophistication of attacks grow, it is far from clear whether any current approach will lead to effective attack detection. However, based on experience in many other domains a collection of detectors, of similar or different types, might well provide a means of achieving practical detection goals. In ML, such collections of primitive elements (e.g. detectors or classifiers) are termed ensembles. We will investigate their use in this thesis. In particular, we will investigate whether specific combinations can be found via EC approaches. Our second hypothesis is therefore:

Hypothesis 2*: evolutionary computation can act as a meta-learner to evolve a function for generating stacking models (intrusion detection programs) that are more effective than previously demonstrated.*

Efficiency and effectiveness are equally important aspects of intrusion detection programs. There have been few resource-aware applications in the intrusion detection domain that have been implemented in operational real-world settings. Humans are not experienced enough at making good decisions if complicated trade-offs need to be considered. A multi-objective evolutionary algorithm, which enables us to optimise one or more objectives at the same time, is used to discover detectors that are both effective (i.e. perform well against the standard detection and alarm rates) and efficient (i.e. are fast and generally consume minimal resources). Therefore, our final hypothesis is:

Hypothesis 3*: multi-objective evolutionary computation will be able to explore the trade-offs between functional (intrusion detection ability) and non-functional (complexity, memory usage and processing time) properties of evolved programs.*

1.4 Thesis Overview

The remainder of the thesis is structured as follows:

Chapter 2: outlines IDS architecture and provides an IDS taxonomy. Then it gives a review of the related intrusion detection systems in the literature. Finally, a summary of the chapter is given together with a discussion of the major problems in the field.

Chapter 3: describes the learning techniques utilised in this thesis. It starts with a brief introduction to evolutionary computation as well as details three implementations namely genetic programming, grammatical evolution and cartesian genetic programming. It subsequently explains how EC can be extended to solve multiple objectives optimisation issues. The next section presents the ensemble learning paradigm showing how other learning algorithms can be utilised together with EC to increase the learning performance. In addition, an overview of the previous implementations of EC to address IDS problems is given. Lastly, the reasons for adopting ECs are summarised and conclusions are reached.

Chapter 4: examines the characteristics of datasets utilised for training and testing intrusion systems. Details about the features used and the attacks found in each dataset are explained. Conclusions are summarised in the last section.

Chapter 5: This chapter details how to apply evolutionary computation to obtain intrusion detection programs. Firstly, an overview of the utilised frameworks is introduced. Next the application of each algorithm to intrusion detection is outlined. The performance of evolved programs are demonstrated and explained. The results are reported and compared. Furthermore, the use of proposed frameworks to evolve IDS rules for classifying unknown attacks is also introduced. Finally, conclusions are drawn.

Chapter 6: aims to evolve a classifier design scheme that incorporates multiple learners to provide superior predictive performance over a single algorithm. This implementation converts the input vector space (i.e. base learners' prediction) into a decision space to predict the class type. The proposed techniques will combine multiple learning algorithms into a single and powerful prediction function. The evolved programs are demonstrated and explained. The results are reported and the performance of evolved programs are compared. In addition, we demonstrate the adoption of ensemble paradigms to evolve intrusion detection programs for detecting unknown attacks. The

conclusions are presented.

Chapter 7: explores the efficiency of intrusion detection systems in resource constrained environments. Multi-objective evolutionary computation is used to explore the trade-offs among intrusion detection program security performance and non-security constraints such as complexity, memory consumption and processing time. The traditional concept of Pareto fronts produces a principled way to examine the trade-offs implemented. In addition, we examine the capability of the evolved programs for detecting unseen attacks during the training stage. The obtained results are illustrated and analysed. Finally, the conclusions are summarised.

Chapter 8: concludes the thesis. This chapter firstly shows a summary of the thesis. The contributions of the thesis are highlighted and future work is discussed.

Chapter 2 | Intrusion Detection: Concepts and Related Work

This chapter introduces the intrusion detection system. It also presents IDS architecture and taxonomy, with an explanation of each characteristic in the taxonomy. Then it gives a review of the related works in this area. Finally, it provides a summary and discusses the main problems in the field.

2.1 Intrusion Detection System (IDS)

"Intrusion detection is the process of monitoring the events occurring in a computer system or network and analysing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices" [16]. In recent years, security threats have had a major effect on the confidentiality, privacy and integrity of services being performed on the internet. IDS technologies tackle security problems through event gathering, logging, detection and prevention [4]. Many research studies have been focused on building IDSs with high detection rates and low false alarms. However, recent investigation [17] has concluded that IDS should be faster, flexible (instead of having strict thresholds), dynamic towards recognising novel patterns and combining logically related false alarms in order to locate the root cause of alarms. An effective IDS has the potential to tackle these issues.

The earliest work that put forward the concept of automated threat detection reaches back to Anderson in 1980 [18]. In his report, Anderson introduced the idea of detecting unauthorised and malicious activities according to an audit record of the operating system activities. In 1985, Denning and Neumann proposed a real-time intrusion detection model which is called Intrusion Detection Expert System (IDES). This proposal considered IDES as system-independent as it observes logins, program executions, file, device accesses, etc for any signs of anomalies in usage. In 1988, Teresa L et al. [19] improved the model and implemented a real time IDS developed at the SRI International

[20]. This system monitors the activities of different types of subjects, such as users and remote hosts, of a target system to detect security violations by both insiders and outsiders as they occur.

An IDS contains various modules to monitor and decide whether activities are malicious or non-malicious in an efficient manner. The modules are a system to monitor, data collection and pre-processing, intrusion recognition engine and a reporting/response unit. The monitoring unit(s) collect samples of raw audit data (such as system logs, network packets, etc.) and pass them to the pre-processing stage. This unit analyses data and extracts features in such a way that the detection engine can later examine them. The intrusion recognition system searches the treated data for any sign of malicious conduct. Finally, based on detection engine outcomes, the alarm/response unit takes action. Figure 2.1 shows an IDS organisation and the relations between these components. The solid lines refer to data/control flow while dashed lines refer to the response to intrusive activities.

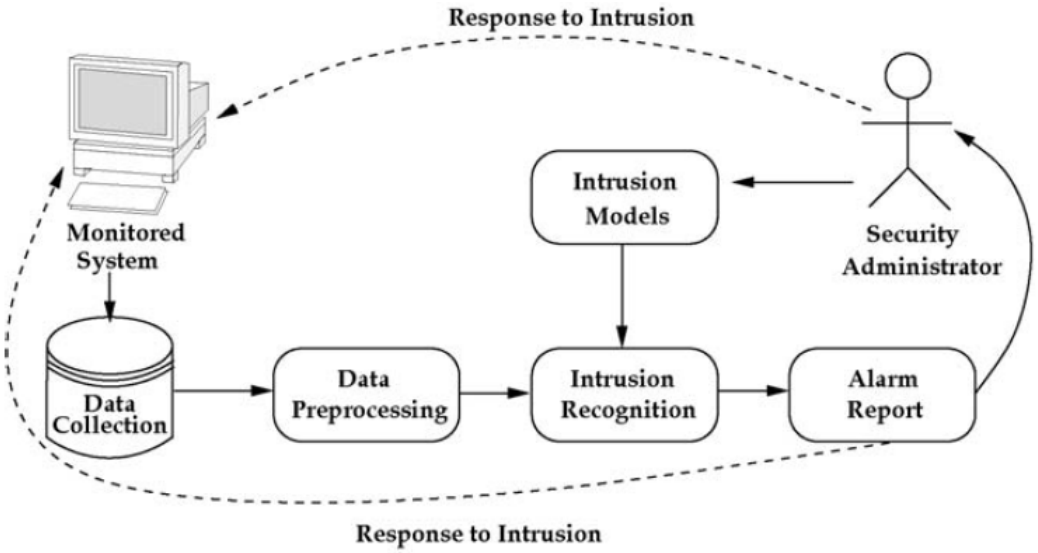

Figure 2.1: Architecture of an IDS [12]

2.2 Taxonomy of Intrusion Detection Systems

It is important for IDS designers and users to understand and study IDS categories in order to select the best IDS characteristics. There are many ways to classify IDSs. IDS categories depend on system deployment, detection strategy, utilised machine-learning techniques, etc. On the other hand, to achieve greater security some IDSs are hybrid systems and belong to more than one category.

2.2.1 Deployment

IDS placements are an important means of giving protection to a system or its monitored environment. The placement of the IDS can be categorised into one of four groups according to its location.

(a) Host-Based IDS: Applies to systems that are located on and monitor a single host machine where it gives them the extra privilege of discovering changes related to file systems [4]. Host-based IDS can also be implemented by observing the system's kernel where system calls show the behaviour of the program. In addition, because the IDS is located at the end of the communication line, they can also identify some network related attacks. A distributed attack is one of the most obvious weaknesses of such systems.

(b) Network-Based IDS: monitors packets passing through a network and able to examine any number of hosts. Network-Based IDS is categorised according to its architecture into three kinds: stand-alone, distributed or hierarchical [21]. The level of detection may vary from one to another. However, most of them have modules in charge of investigating traffic from the network, transport, and application layers in the Open Systems Interconnection (OSI) model. The main problem occurs when network bandwidth is high. In this case, the IDS might have some difficulties examining all the packets.

(c) Hybrid IDS: Combines the workloads of both network-based and host-based IDS systems. Host-based IDSs will be deployed into the critical parts of the protected environment including servers, databases and the like. These types of IDS primarily depend on the host-based parts and employ the network-based system to complement the protection [22].

(d) Application-Based IDS: Examines events located at the application level on a host machine. They can detect intrusion attempts towards a specific application. It often detects many types of attacks by analysing the application log files. However, application-based IDSs are known to consume significant application (and host) resources [22]. An example of an application-based IDS is the web-application firewall.

2.2.2 Detection Methods

Many techniques have been used to identify intrusions. They can be classified into two main groups: misuse or anomaly. Recently, both misuse and anomaly based approaches have been used together in IDSs to improve effectiveness due to their strengths and

weaknesses being complementary [21]. Each of these detection methods is presented below:

(a) Misuse-Based Detection: matches features of current activates against stored profiles of characteristic properties of known attacks. Commercial IDSs mostly adopt misuse-based approaches due to their effectiveness with known attacks and low false alarm rates. They are also fairly easy to implement. Anytime a new attack is discovered the system updates the database by adding attack signatures. The drawback of this approach is that it cannot identify novel attacks, even if they are slightly modified from a known pattern. The system primarily relies on the strength of its signature database which requires constant updating [12].

(b) Anomaly-Based Detection: Anomaly-based intrusion detection seeks to detect "unusual" behaviour of network connections, hosts or users over a period of time [4]. In this approach, firstly it computes the normal model by profiling the regular behaviour of user activities. While in detection mode, the system compares the current monitored activities with these profiles, and any deviation is reported as being anomalous (i.e. an attack attempt). Due to the constantly changing normal behaviour by the addition of new features or technologies, IDS requires frequent updates. The false positives rate (the rate at which regular activities are classified as anomalies) is often high in this type. However, anomaly-based IDS can detect emerging attacks which are unrelated to any known attacks. This is particularly significant in an environment in which novel attacks and new vulnerabilities of systems are discovered now and then.

(c) Hybrid Detection: An IDS that is a combination of both previous systems. It is used to increase detection rates of known attacks and lower the false positive rate for unknown attacks. It requires two processing factors. Triggering an alarm is based on performing a decision from multiple methodologies. For example, a method identifies anomalous behaviours while decoding the events, and another detection engine performs the signature matching [22].

2.2.3 Machine Learning Techniques

Another division of IDSs is based on what machine learning techniques have been utilised to construct intrusion detection engines. These techniques are often used as a means of synthesising event classifiers, indicators of whether an event is normal or malicious. Various studies implement single learning techniques whereas some systems combine more than one learning technique, for instance, hybrid or ensemble approaches.

(a) Single Classifiers: the intrusion detection dilemma can be addressed by utilising a single machine learning algorithm only [23]. In the previous works, both supervised and unsupervised approaches have been applied to solve these issues. Normally, the IDS building process goes through two main operations. The learning (training) responsibility is to measure the estimated distance between the input-output instances to build a classifier (model). Following the model construction, it can categorise unseen instances (testing). There is not much flexibility in this type and IDS performance depends on the learning paradigm used.

(b) Hybrid Classifiers: During the building of an IDS, the usual primary objective is to gain the best potential accuracy for the targeted task. This purpose normally requires the concept of hybrid techniques for the problem to be resolved [23]. The main idea of the hybrid classifier is to integrate multiple ML algorithms which allows the system efficacy to be considerably enhanced. Additionally, either supervised and/or unsupervised learning techniques, or both types together, can be used to build the hybrid system. Each technique in hybrid classifiers will perform a specific task. Different methods have been proposed to hybrid classifiers. For instance, an algorithm uses raw data as input and produces intermediate outputs. The second algorithm will then use the intermediate outputs as the input and generate the final outcomes. Another example is when an algorithm is used to tune the parameters of the second algorithm in order to optimise the prediction performance of the second model. The main disadvantage of this kind is that the constructed paradigms are engineered in a way that usually has non-interchangeable positions.

(c) Ensemble Classifiers: these combine various machine learning models which allows the overall performance to be effectively enhanced [24]. The idea behind an ensemble classifier is to boost a weak learning paradigm performance. These models could be from the same or from different learning paradigms. Combining strategies are usually of two types: non-trainable such as majority voting or trainable approaches such as stacking. The difficulty of ensemble approaches come from the selection of the algorithms creating the ensemble. One of the advantages of ensemble approaches is their modular structure, where the designer of an ensemble can easily replace one or more algorithms with a more accurate one. It is necessary to consider the computational cost that is attached to each new algorithm. Ensembles help to meet the following challenges of IDSs [17] [25]:

- An Ensemble classifier capable of modelling the problem is based on different subsets of a dataset or its features. This is particularly important when there is an insufficient amount of quality training data.

- A reduction of false positives and false negatives. This is supported by the idea that different learning algorithms explore different characteristics of the problem.
- Ensemble-based algorithms perform just as well when data are not available in necessary amounts and when we have a large amount of data.
- An ensemble can model with ease various application or parts of a network (i.e. some models can be developed for some parts or for some levels of the network) and eventually aggregate together, to secure more effective performance.

There are many other possible classifications. For example, [22] gives a taxonomy based on the technology where IDSs may be wired or wireless. Wireless IDSs can be further classified as fixed or mobile. Regarding the data processing method and the arrangement of its components, IDSs can be centralised or distributed. Furthermore, IDSs have different types of action depending on the way they are responding. If an IDS responds to the detection of an attack then it is "active" otherwise it is deemed "passive", which means it only generates alarms. An active response could seek either to take actions on the attacked, or the attacking system.

2.3 Intrusion Detection System Performance Metrics

The effectiveness of a proposed intrusion detection system can be measured according to how malicious and normal behaviours are classified correctly. For this purpose, the most widely adopted statistical measure for the binary classification problem is the confusion matrix. A confusion matrix contains four values: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP refers to malicious behaviour being categorised as a threat, and TN indicates normal behaviour being classed as normal. FN refers to malicious behaviour being categorised as normal and FP indicates normal behaviour being classed as malicious, both of these are problematic. False positives waste a great deal of time and can lead to loss of confidence. False negatives are examples of the detector system not performing its primary task, i.e. to detect attacks. There are some derived measurements that are also used in this thesis:

$$Detection\ Rate\ (DR) = \frac{TP}{(TP + FN)} \tag{2.1}$$

$$Accuracy\ (Acc) = \frac{(TP + TN)}{(TN + TP + FN + FP)} \tag{2.2}$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{(FP + TN)} \tag{2.3}$$

$$False\ Negative\ Rate\ (FNR) = \frac{FN}{(FN + TP)} \tag{2.4}$$

Equation 2.1 indicates the fraction of real attacks that are detected; this is sometimes referred to as Recall. Equation 2.2 defines the fraction of all instances (attacks or non-attacks) that are correctly classified. Equation 2.3 describes the fraction of normal behaviours that are erroneously flagged as malicious, whereas equation 2.4 shows the rate of anomaly behaviours that are being examined which yield normal test outcomes. IDS experts consider the effect of false rates as just as critical as the detection accuracy. Equation 2.5 shows how to calculate the rate of the misclassified instances.

$$False\ Alarm\ Rate\ (FAR) = \frac{(FPR + FNR)}{2} \tag{2.5}$$

2.4 Related Work

To keep pace, knowledge extraction using machine learning techniques can help defenders complement threat prevention, detection and remediation [13]. These techniques are used to perform various roles from data pre-processing to a deeper understanding of data which helps automate the detection process. Despite their success in other areas, machine learning techniques have not seen much in the way of real-world IDS application [14]. However, the characteristics of learning methods, for example, adaptation, fault tolerance, high computational speed and error resilience in the face of noisy information match the prerequisites of developing a security tool and could potentially fill the gaps [12]. In this overview, the first part encompasses various machine learning techniques that have been implemented in order to identify threats. The second part lists some results obtained from various techniques deployed against used datasets.

Bouzida and Cuppens [26] adopted two supervised algorithms: multilayer neural network and decision trees (C4.5) for anomaly-based intrusion techniques. Neural networks showed promise for generalisation accuracy, but were very poor at detecting new attacks. Decision trees showed a high efficiency in both generalisation accuracy and new attacks discovery. The KDD dataset was used, it extracted from the DARPA dataset. This dataset comprises 24 attack kinds that could be grouped into four main types namely Denial of Service (DOS), Remote to User (R2L), User to Root (U2R) and probing. In addition, they applied their technique to real network traffic from their laboratory, which contains new attacks not present when DARPA was created. Even though the achieved results are interesting when compared with earlier works, nonetheless the detection rate of a number of attack kinds continues to be poor. There were two contributions made by this study: adopting the idea of anomaly intrusion detection

through considering both normal and known intrusions while in the training stage and adding a new class for the novel intrusion instances (since these are unrepresented in the training dataset).

Bankovic et al. [27] implemented a technique that creates a set of rules to classify network connections as either normal or intrusive and provides an identification of the attack type. A dimension reduction technique, Principal Component Analysis (PCA), also known as Karhunen-Loève transform, was used to identify a subset of features that maintain the most relevant information. For evolving rules, the KDD dataset was used. The 41 features in the KDD dataset were cut down to only 3 at the first step by PCA. The second step uses a genetic algorithm to create a set of rules to classify behaviour as normal or abnormal. Each rule for intrusion detection is an if-then clause form. The conditional part of the rule is composed of the features connected by the AND function. The result of each rule is a verification of an intrusion taxonomy. This approach has the ability to pre-process network data in real-time and offers a high detection rate and low false positive rate. Nevertheless, they considered only three kinds of attacks which are not sufficient to assess the technique.

Stevanovic et al. [28] studied the utilisation of two unsupervised learning algorithms: the Self-Organising Map (SOM) and Modified Adaptive Resonance Theory 2 (Modified ART2) to analyse web logs and detect malicious and non-malicious visitors. Three million log records were obtained from web-based accesses into their domain over a period of four weeks. Before characterising web visitors, they applied data pre-processing in order to point out unique visitor sessions and generate 10 features for each session. The SOM clustering method uses multiple rounds of a competitive learning process to make the underlying algorithm respond in the same way to similar input patterns. The ART2 algorithm also uses the idea of competitive learning, however, combined with the winner-takes-all rule, eventually generating different clustering results. In their case study of web log analysis, they identify four main kinds of website visitors: human visitor, well-behaved crawler, malicious crawler, and unknown.

In [29], Ambusaidi et al. build an IDS using the Least Square SVM algorithm integrated with the feature selection algorithm: Flexible Mutual Information Based Feature Selection (LS-SVM-IDS+FMIFS). The main idea is to remove features that are redundant and irrelevant from the data, as these features slow down the classification process but likewise affect the IDS classification accuracy. The evaluation showed that the feature selection algorithm contributes features that are more critical for LS-SVM-IDS to accomplish better performance and reduce computational cost. Figure 2.2

shows the detection framework main stages: (1) data collection in which flows of network packets are obtained, (2) data pre-processing in which training and testing data are pre-processed, and the significant features that can effectively distinguish various classes are chosen, (3) classifier training in which the model for classification is formed using the LS-SVM algorithm, and (4) attack recognition in which the developed model is employed to detect attacks in the test data. The model was examined utilising 3 datasets that are KDD, NSL-KDD and Kyoto 2006+. The NSL-KDD dataset was introduced to solve some issues in the original KDD dataset. The developed IDS has accomplished promising achievements in detecting intrusions over computer networks. In general, LSSVM-IDS + FMIFS has the best efficacy in comparison to the other state-of-the art algorithms.

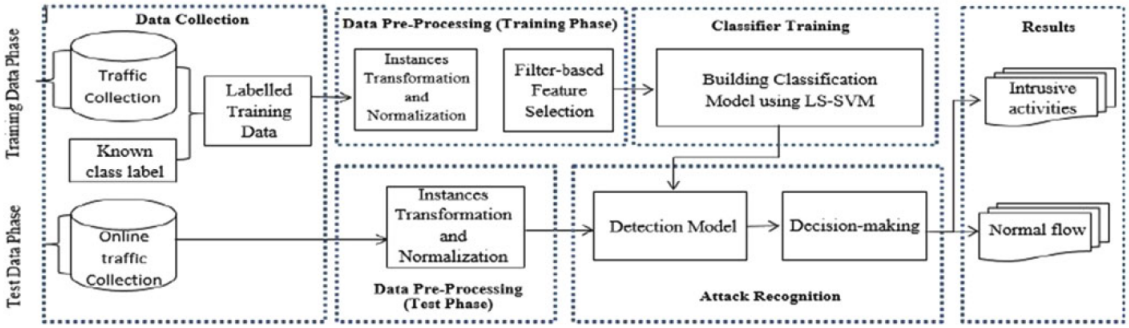

Figure 2.2: Ambusaidi et al. Framework of the LS-SVM-based IDS [29]

Najafabadi et al. [30] used machine learning algorithms to build predictive models to discriminate normal network traffic from malicious network traffic. The proposed architecture includes a data reduction method (i.e. feature selection) to remove irrelevant and redundant features which leads to a decrease in the process time. They tested three classification models (5-nearest neighbor, C4.5 decision tree and Naïve Bayes) alongside four different feature selection methods. The Kyoto 2006+ dataset is utilised for training and testing the models. The classification results showed that while feature selection reduces the number of features, it maintains the same or does not decrease outcomes significantly. They conclude that the feature selection in the IDS application domain is an important pre-processing step and it should not be taken lightly.

Rathore et al. [9] proposed an IDS to tackle the issue of dealing with a large amount of data and a high-speed environment (i.e. Internet and network services). This proposed system comprises a 4 layered IDS structure, which contains a capturing layer, filtration and load balancing layer, parallel processing layer (i.e. Hadoop) and the decision-making layer (see Figure 2.3). DARPA, KDD and NSL-KDD were used for evaluation. Furthermore, two feature selection approaches and DARPA traffic analysis are used

to cut down the number of input features from 41 to 9. In the beginning, the flow of network packets is collected using a high-speed capturing device. The second layer uses the In-Memory intruders' database to carry out efficient searching and comparisons of the incoming flow's traffic in order to classify it as an intrusion or normal flow. Then, it sends the traffic of the unidentified flow with packet header data to the third layer (i.e. Hadoop) master servers. It also balances the burden by determining which packets are forwarded to which master server based on the IP addresses. Hadoop has a MapReduce code that utilises the Map and Reduce function provided with some parameters calculations code to calculate the 9 features values. Since MapReduce runs simultaneously the overall performance is boosted. In the end, the feature values are transmitted to the decision-making layer that categorises the flows as benign, or threats, depending on their features values. This layer was tested using five ML algorithms namely J48, REPTree, random forest tree, conjunctive rule, SVM and Naïve Bayes. Both REPTree and J48 achieved the highest performance when it comes to processing time and accuracy.

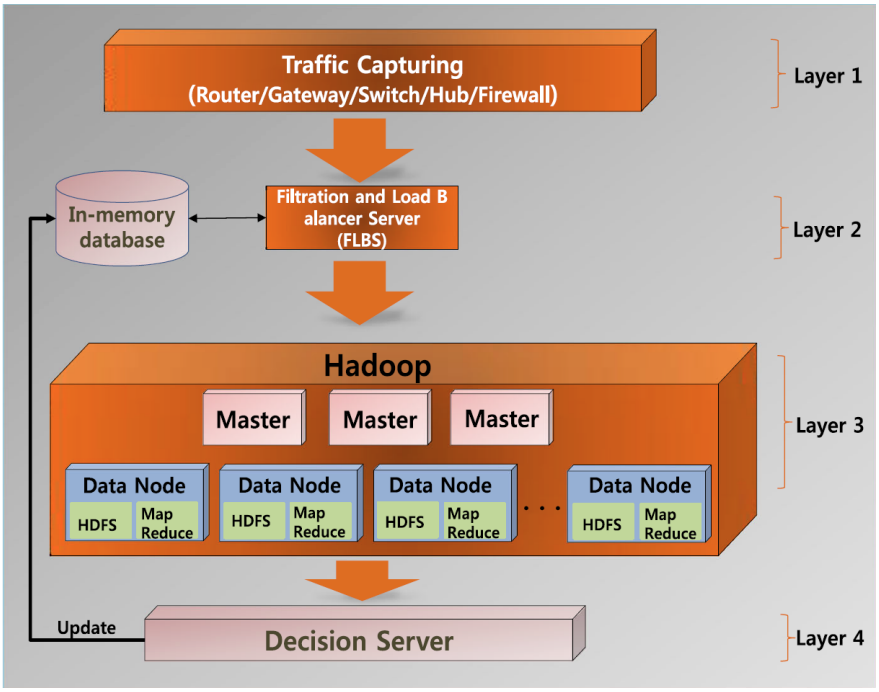

Figure 2.3: Rathore et al. IDS Architecture [9]

Chebrolu et al. [31] researched the use of Bayesian Networks (BN) and Classification and Regression Trees (CART) algorithms for IDS. They used these two paradigms as a hybrid classifier and as an ensemble classifier. The dataset utilised in this study was the DARPA, from which a subset was randomly chosen. Feature selection was also implemented to speed up the computation. First, BN and CART were evaluated separately with full and with subsets of features. The performance on the set of 41

features was compared to a set of 17 selected by BN and 12 selected by CART. BN performed worse with a smaller set of features except on the normal class. CART using a reduced dataset achieved 100% normal class classification accuracy and a increased U2R and R2L classification accuracy as well. The ensemble model was tested using a reduced features set (i.e. 12 and 17) and 41 from the dataset and its workflow is illustrated in Figure 2.4a. In the ensemble method, the final decisions were reached as follows: each paradigm's output is assigned a weight based on the generalisation accuracy. If both paradigms agree then the final result is chosen accordingly. In case there is a conflict in the decision given by the paradigm then the one with the highest weight is favoured. Based on the results, they concluded that the ensemble model provides more effective performance than the two paradigms working individually. After summarising the models performance with both ensemble and individual classifiers, they formed a hybrid IDS architecture shown in Figure 2.4b. Finally, by utilising the hybrid model; normal, probe and DOS instances were detected more accurately.

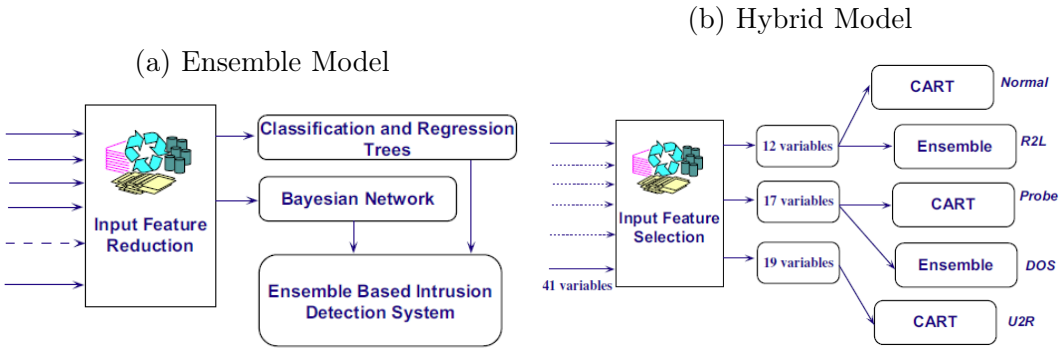

Figure 2.4: Ensemble and Hybrid Models developed by Chebrolu et al. [31]

Xiang et al. [32] proposed a multiple-level hybrid classifier for the IDS. This approach combined both an unsupervised learner (i.e. Bayesian clustering) and a supervised learner (i.e. Decision tree C4.5). The KDD dataset was utilised in this work. At the first stage of classification, the C4.5 model grouped instances into 3 categories DoS, Probe and Other. Both U2R and R2L attacks and Benign instances are categorised as Others at this stage. Stage 2 uses Bayesian clustering to separate out Others into (U2R and R2L) as attacks and Benign instances using only 4 features out of 41. These features were chosen after weighing the significance of each feature by applying a collection of criteria, for instance, information gain. The C4.5 model was utilised to differentiate U2R and R2L at stage 3 using 14 features only. This step is executed much more easier given that the Benign instances have been separated out at stage 2. Finally, C4.5 is used to further categorise each class of attack into its variations. The experimental results showed a very efficient detection rate with a very low false negative rate, and at the

same time it maintained a reasonable false alarm rate level compared with other popular approaches. However, unknown attacks that are missing from the training dataset are very low.

Abraham and Thomas [33] proposed an ensemble approach consisting of three base classifiers (Decision Trees (DT), Support Vector Machines (SVM), and a hybrid system composed of DT and SVM). The structure of the ensemble model is illustrated in Figure 2.5. The IDS model is generated by using KDD as the training dataset. In this experiment, each model afforded complementary information about the observed behaviours (i.e. 5 different classes) in order to be categorised. The final outcome of the ensemble classifier is determined by the highest score of the base models. The score of each model is measured via assigned weights which represent the individual prediction performance on the training dataset. Therefore, for a certain example to be categorised and in case all models provide diverse opinions, then the ensemble classifier uses a models score. The model possessing the highest score is considered to be the winner and utilised to decide the final outcome. Evidently, the ensemble method makes use of the differences in misclassification and enhances the overall performance.

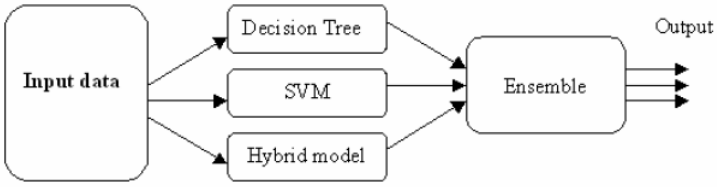

Figure 2.5: Ensemble Architecture Proposed by Abraham and Thomas [33]

Zainal et al. [34] proposed a heterogeneous ensemble by fusing three different classifiers designed for detecting a single class. They adopted learning algorithms such as the following: Linear Genetic Programming (LGP), Random Forest (RF), and Adaptive Neuro-Fuzzy Inference System (ANFIS) to construct a network-based IDS. Each of the algorithms (i.e. LGP, ANFIS and RF) used the same dataset during the training. This study used the KDD dataset. Rough Set Technique and Discrete Particle Swarm Optimisation (RST-BPSO) was employed to extract the important features. The original 41 features were cut down to 15 for all classes and selected features for each class are vary. After building base classifiers, the system then used the weighted voting method to determine the final classification. Overall, the performance of LGP is better in comparison with the other two algorithms, while both ANFIS and RF are nearly at the same level. It is shown that the ensemble system performs better than the best-performing classifier alone. Figure 2.6 depicts the proposed ensemble workflow.

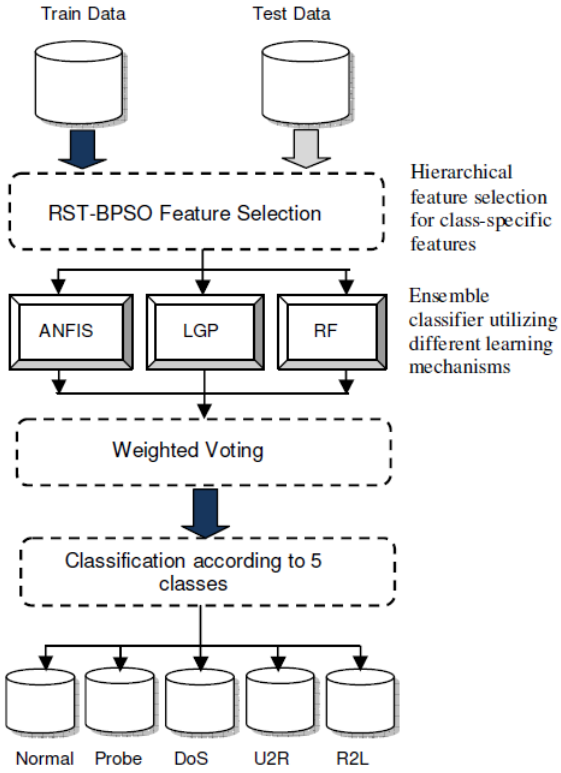

Figure 2.6: Zainal et al. Ensemble Experimental workflow [34]

Syarif et al. [35] investigated the implementation of three ensemble paradigms to improve network-based IDSs performance. They used bagging, boosting and stacking systems to address intrusion detection problems in order to enhance the accuracy and decrease its false positive rates. The four data mining algorithms utilised as base learners for those ensemble classifiers were naïve Bayes, J48 (decision tree), JRip (rule induction) and iBK (nearest neighbor). The NSL-KDD dataset was used. Regarding the 4 base classifiers utilised, J48 achieved better performance compared with the other 3 methods in terms of the highest accuracy rates, lowest false positive rate and faster execution time. They reported that proposed approaches achieved a high accuracy in detecting known attacks, however, it failed to recognise novel attacks effectively. The utilisation of bagging, boosting and stacking displayed no significant gain in accuracy. Although stacking was the only paradigm that resulted in a notable decrease in false positive rates, its execution time was the longest. Therefore, they concluded that it is unqualified to become an effective solution for the intrusion detection problem. However, with recent improvements in computing capabilities, this issue could be overcome easily. In addition, even a small reduction in misclassification error is very important for the IDS field.

Kamarudin et al. [36] proposed an anomaly-based intrusion detection system utilising an ensemble concept to detect novel attacks targeting web servers. The logitboost algorithm is employed as a stacking approach along with the RF algorithm as a base learner because of its power in coping with noise and outliers data. The system was tested utilising 2 datasets (NSL-KDD and UNSW-NB15) focusing only on HTTP traffic. A hybrid feature selection technique was used as a pre-processing step to select significant features only, which helped reduce the overall detection time. For NSL-KDD, the number of features was reduced from 41 to 10, while for UNSW-NB15 the number of features was reduced from 43 to 5. Figure 2.7 presents the workflow of the proposed anomaly detection ensemble model. The authors made sure that the attack traffic in each of the training and testing portions was completely varied. The logitboost classifier algorithm achieved excellent overall accuracy and also maintained a low false alarm rate. However, the results showed that the detection rate of some of the unknown attacks was very low from both datasets.

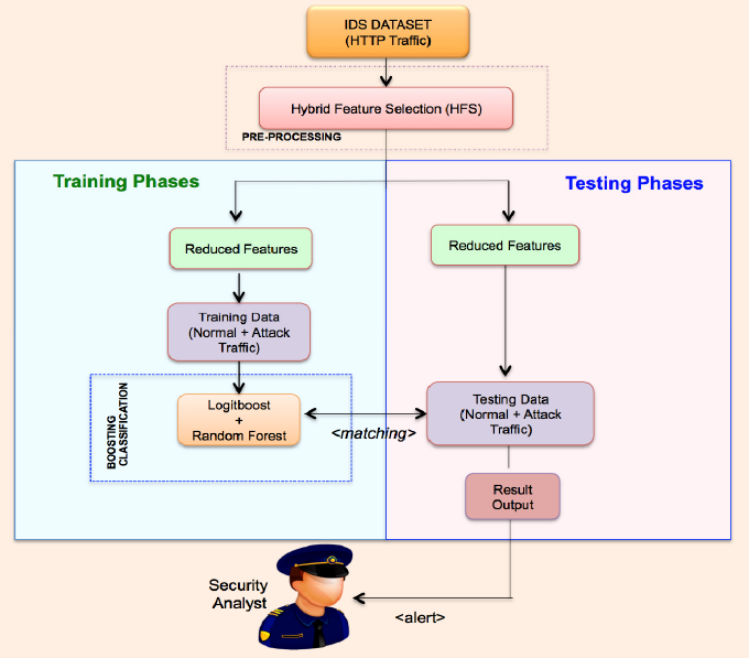

Figure 2.7: Kamarudin et al. Ensemble Anomaly Detection Model [36]

Finally, Table 2.1 shows various techniques experimentally deployed on the utilised datasets in this thesis. In the performance metrics column beside DR, FPR, Acc and FAR, there is the precision ($\frac{TP}{(TP+FP)}$) which describes the fraction of raised attacks that were predicted correctly. In addition, F1-score ($2 * \frac{(Recall*Precision)}{(Recall+Precision)}$), is a measure that includes precision as well as recall and shows approximately the average of the two when they are close.

Author(s)	**Technique(s)**	**Pre-processing Stage(s)**	**Dataset Used**	**Split Percentage**	**Performance Metric(s)**
Ambusaidi et al. [29]	Intrusion Detection System: Least Square Support Vector Machine	Feature Transformation Normalization Feature Selection	Kyoto 2006+	Training = 70% Testing = 30%	DR = 99.64% FPR = 0.13% Acc = 99.77%
Thabtah et al. [37]	Anti-phishing Model: Improved Self-Structuring Neural Network epoch (500)	Feature Selection	Phishing Websites	Training = 80% Testing = 20%	Acc = 93.06% F1-score = 92.30% Recall = 91.12% Precision = 93.71%
Moustafa and Slay [38]	Network Anomaly Detection System: Decision Tree	N/A	UNSW-NB15	Training = 60% Testing = 40%	Acc = 85.56% FAR = 15.78%
Moustafa et al. [39]	Anomaly Detection System: Geometric Area Analysis	Feature Conversion Feature Reduction Feature Normalisation	UNSW-NB15	Training = 60% Testing = 40%	DR = 91.3% Acc = 92.8% FPR = 5.1%
Alkasassbeh et al. [40]	Intrusion Detection System: Multilayer Perceptron	N/A	Modern DDoS	Training = 66% Testing = 34%	Acc = 98.63%
Sharafaldin et al. [41]	Intrusion Detection System: Random Forest	N/A	CICIDS2017	N/A	Precision = 0.98 Recall = 0.97 F1-score = 0.97

Table 2.1: Previous Works Used Adopted Datasets

2.5 Discussion

An intrusion detection system is considered to be a key aspect of present-day security management tools. In IDS research, ML techniques have been applied towards building IDS solutions, however, they are still facing problems and further research is needed. After analysing the aforementioned papers in the literature, we identified major challenges that have been largely ignored or otherwise need further investigation. These challenges, which are also confirmed to be vital by recent surveys, are listed below:

1. Techniques used to address cybersecurity issues should produce understandable solutions (i.e. readable and in an easy to comprehend format) [12] [42].

2. IDSs need to be able to deal with new and undocumented attacks from which signatures are not available [25] [4].

3. "Current IDSs pose challenges on not only capricious intrusion categories, but also huge computational power" [4].

4. IDS should be able to handle the ever increasing number of network connections and process the sheer volume of data generated in real-time [25] [42].

A recent survey [42] concluded that data mining and ML methods are a prevalent and growing research area for cyber security. For instance, ensemble methods have emerged as a promising way of providing reliable and intelligent IDSs [17], [24]. These methods can improve the performance of IDSs (i.e. effectiveness) over a single classifier in terms of both detection rate and false alarms. This is due to the fact that the learning algorithms operate in a different way upon various classes of attacks (e.g. DoS, Probe, U2R and R2L) [43]. However, a recent survey indicates that the application of the heterogeneous ensemble in solving IDS problems is still quite modest [24].

A major limitation of the misuse-based approach is its failure to detect new threats which are previously unknown to the IDS. However, ML holds real value to security teams by overcoming skills and resources gaps, this especially stems from its ability to detect both known and emerging threats [13]. Unknown attacks, which are classified into two types: unseen variations of known attacks or unseen attacks, are emerging as one of the most serious threats to any system [13]. An effective IDS must have the capability to identify attacks that it has never before encountered.

The production of readable output is more and more important as it provides security analysts with some degree of manual analysis so it is easy to understand and

deploy solutions. Finally, due to the highly complex and ill-understood properties of the examined search space (i.e. environment) the ML methods are the best solution to explore these properties [44].

2.6 Conclusions

The internet is a medium that provides services to customers using machine-to-machine interaction on various digital devices. It has become an important factor for many domains, e.g. e-government, e-finance, and e-health. Unfortunately, these services are vulnerable and under constant risk of attack. The security issues related to these services are a major concern to their providers and are directly relevant to the everyday lives of system users. Furthermore, new attacks emerge every day and current mechanisms are not sufficient in themselves. Therefore, researchers have adopted various ML techniques to fill these security gaps after the successes achieved by ML in many other domains.

In this chapter, we have reviewed relevant applications related to developing ML-based IDSs. The contribution/novelty of the proposed systems is discussed and argued. There are still issues that have been largely ignored or otherwise need further research. Furthermore, the research often investigates a very limited number of attacks and makes assumptions about the architectures of solutions. We highlighted the main issues that need to be considered when designing an IDS. We have argued that ML, perhaps today's highest profile set of tools and techniques, offers the potential to help address these issues.

In this thesis, we study the use of ML methods to investigate IDS design space for this deeply complex environment. Evolutionary computation, which is a group of optimisation algorithms based on biological evolution, provides a framework to build computer programs for IDS that are both effective and efficient. In this rich trade-off space, the evolved programs suitability for the explored environment is also taken into account during the IDS design. The following criteria are considered: solution complexity, limited resources and the detection of unseen attacks. The framework used for each proposed paradigm and the evaluation results are given in the subsequent chapters.

Chapter 3 | Learning Techniques

This chapter describes the learning techniques utilised in this thesis. In the beginning, a general introduction to Evolutionary Computation (EC) and a description of its three implementations: Genetic Programming, Grammatical Evolution and Cartesian Genetic Programming. In addition, it explains the concept of the multi-objective optimisation search paradigm. It goes on to describe the ensemble learning paradigm to show how multiple learning techniques can be combined to improve the classification performance. Finally, this chapter looks at previous applications of EC algorithms related to the work given in this thesis, reasons for deploying EC and it draws conclusions.

3.1 Introduction to Evolutionary Computation

The term evolutionary computation is defined as a creative process inspired by evolution in nature, a Darwinian evolutionary system. It is highly suited to addressing real-world issues having a great complexity. EC techniques have been used by researchers to tackle numerous tasks, and it performs different roles, for example, searching for an optimal solution, automatic model design, and the learning of classifiers [12]. The EC approach has a number of attractive features, for instance producing readable outputs, evolving lightweight solutions and supplying a collection of solutions with different trade-offs amongst conflicting objectives [21]. Evolutionary computation has been extensively studied. Variants of it were shown to offer an effective means to generate programs automatically. In general, an EC paradigm begins at producing an initial population including individuals that describe potential solutions to solve a particular problem. Then these individuals are evaluated to show how well each solves or comes near to solving the problem. The evolutionary cycle will continue until some termination criterion is met. Figure 3.1 illustrates the scheme of an evolutionary process with its primary evolutionary operators.

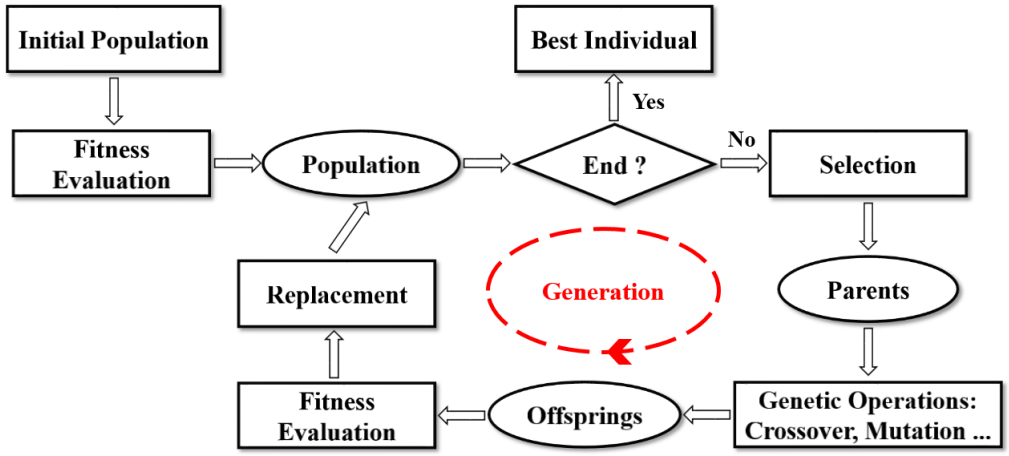

Figure 3.1: The Evolutionary Cycle

Initialisation refers to the creation of the first population. The initial population created will have a significant impact on the result. It is often generated randomly, but techniques that use seeding have also been used (where the initial population members are generated by another suboptimal technique). Next the estimate of how effective the individual is at resolving the problem is captured by means of a fitness function. The main aim here is to optimise the fitness function which correlates with the adopted objectives.

The simulated evolutionary process seeks to evolve ever-fitter populations of candidate solutions, using various operators inspired by biological metaphors for reproduction and evolution, such as crossover, mutation, and selection. **Crossover** is a binary operator that causes solutions to be 'mated': two individuals exchange solution components to form new ones, often referred to as 'offspring'. This allows the possibility of offspring that combine the most effective features of each parent, with fitter offspring subsequently thriving under the survival of the fittest regime of Darwinian natural selection. (Equally, offspring combining poor traits of parents will be unlikely to survive). **Mutation** is a unary operator that causes some solution component to spontaneously mutate, allowing it to take on values that are potentially not possessed currently by any candidate solution in the population. Generally, each element of each individual may be mutated with some small probability. Finally, **selection** determines which solutions go forward to form the next generation of the population. This implements some form of the 'survival of the fittest' regime (fitter solutions have greater chances of going forward). By repeating the indicated cycle it is anticipated that later generations will have fitter individuals and hopefully the process will generate at least one that 'solves' the problem, or is otherwise acceptable (i.e. the solution is highly fit, but it is

not actually known what is the best possible fitness value).

The selection mechanism helps the survival of the best individuals by choosing individuals out of the population according to their fitness measurement. In some cases copies of selected high performing individuals may be chosen to go forward directly to the new population without undergoing further evolutionary operations; this is known as elitism. There are other selection schemes, for instance, roulette wheel, rank based, tournament selection. Tournament selection is utilised in this research. In tournament selection, a set of individuals is randomly drawn from the population then the best individual among them is chosen to be a parent. Tournament size refers to the number of individuals in the aforementioned set. Whereas the probability of an individual being selected using the roulette wheel scheme is proportionate to its fitness measurement. It can be briefly described as follows: $p(i) = f(i) / \sum_{j=0}^{n} f(j)$ where f refers to the fitness value and n indicates the number of individuals.

Like natural evolution, an EC individual has two distinct representations: the genotype which is processed by the genetic search procedures and the phenotype format which it is evaluated by the environment. The representation of individuals helps EC methods to connect the 'real world' to the 'EC world' [45]. It will establish a link between the circumstance of the original problem and the problem solution space in which evolution happens. There are different types of EC with differing means of representing problem solutions. In general, these techniques vary from one another based on how the individuals are represented. It is important that the researcher examines the benefits and limitations of many EC representations instead of limiting themselves to one prevailing structure. Diversity is essential in research just as it is in evolving populations. In this research, the intention is to use GP (tree-based form), GE (grammar-based form) and CGP (graph-based form) to evolve detectors for the internet and computer network attacks. More detail on these techniques is given below.

3.1.1 Genetic Programming (GP)

Genetic Programming generates complete programs and is optimised towards performing a certain task. It was introduced by Koza in 1992 [46] and considered among the most commonly used evolutionary algorithms. It has been applied to many problems and has rivalled or surpassed the accomplishments of various machine learning algorithms, and has produced programs superior to the best programs created by humans [47]. GP constructs the computer program in the form of a tree which consists of functions and terminals. The leaves elements in a tree (the terminals) are the possible

inputs. These terminals in most cases feature set, constants or other functions without argument. The operation set used on these inputs can be mathematical, Boolean, program statements (if, loop), and the like. The automatically generated computer programs evolved out of input/output behaviour can be expressed only using the given terminal and function sets. GP generates the initial population randomly where the produced trees should not exceed maximum tree depth predefined by the user. The tree depth represents the deepest path from the root node of the tree. For illustration consider the implementation of the equation $x * ((x\%y) - sin(x)) + exp(x)$ shown in Figure 3.2.

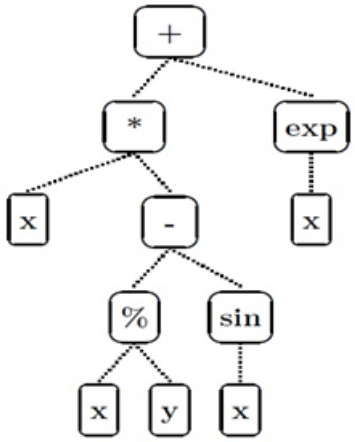

Figure 3.2: GP Syntax Tree of Depth 4

In this example, terminal set = $\{x, y\}$ and function set = $\{+, -, *, \%, cos, sin, exp\}$ are sufficient to produce the tree in Figure 3.2.

There are two main methods utilised for the creating of the initial population in GP; full and grow. The full method formulates trees where nodes are selected at random from the function set until reaching tree depth then the terminal set are selected. Thus, all terminals in this type are at the same level. However, that does not mean that all initial trees are equal in size (i.e. having the same number of nodes). In this method, the range of individual sizes and shapes is somewhat limited [48]. Whereas in the grow method, nodes are picked from both functions and terminals at any tree level until it reaches the maximum depth. Following that only terminals are selected. Hence, the created trees are not going to be full in this method. This method is quite the opposite to the full method, which enables the production of individuals with varied sizes and shapes [48]. Besides these methods, there is a proposed initialisation technique by Koza [46] called ramped half-and-half that combines full and grow methods. The main reason for adopting this technique is to enhance the diversity of the GP population.

This done by splitting the population equally into sets. These sets have a variety of sizes and shapes. During the initialisation phase, half of the individuals are generated by the full method and the other half by the grow method.

The crossover operator is used for the exchanging of subtrees between two individuals. In this research, strongly typed GP is used [49] in which only subtrees with similar constraints (return type) are swapped. The subtree crossover operator is described in Figure 3.3

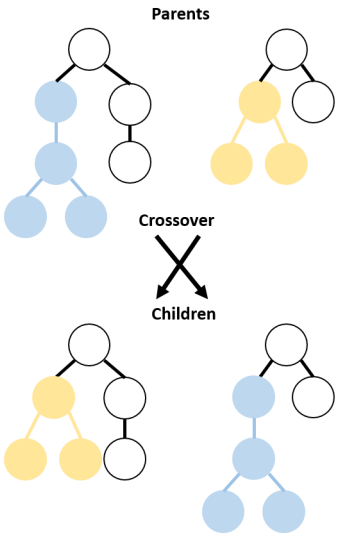

Figure 3.3: Crossover Operator for Genetic Programming

As for the mutation evolutionary process, GP applies point mutation in which a random node in the tree is chosen and replaced with a different random generated subtree. In strongly typed trees, both exchanged subtrees have the same constraints. Figure 3.4 illustrates the mutation process.

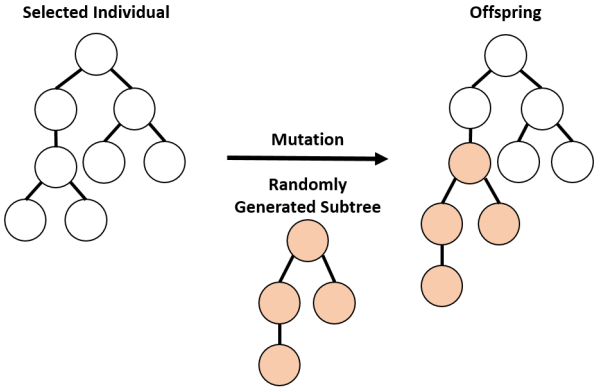

Figure 3.4: Mutation Operator for Genetic Programming

Finally, the new individuals take the place of the old individuals in every generation. Figure 3.5 is a flowchart illustrating the stages of GP algorithm [46]. The index i indicates an individual in the population of size M. The variable GEN is the number of the current generation. The fitness value plays a major factor for selecting individual(s) for breeding, in which GP performs reproduction with a certain probability (Pr) and performs crossover with a certain probability (Pc).

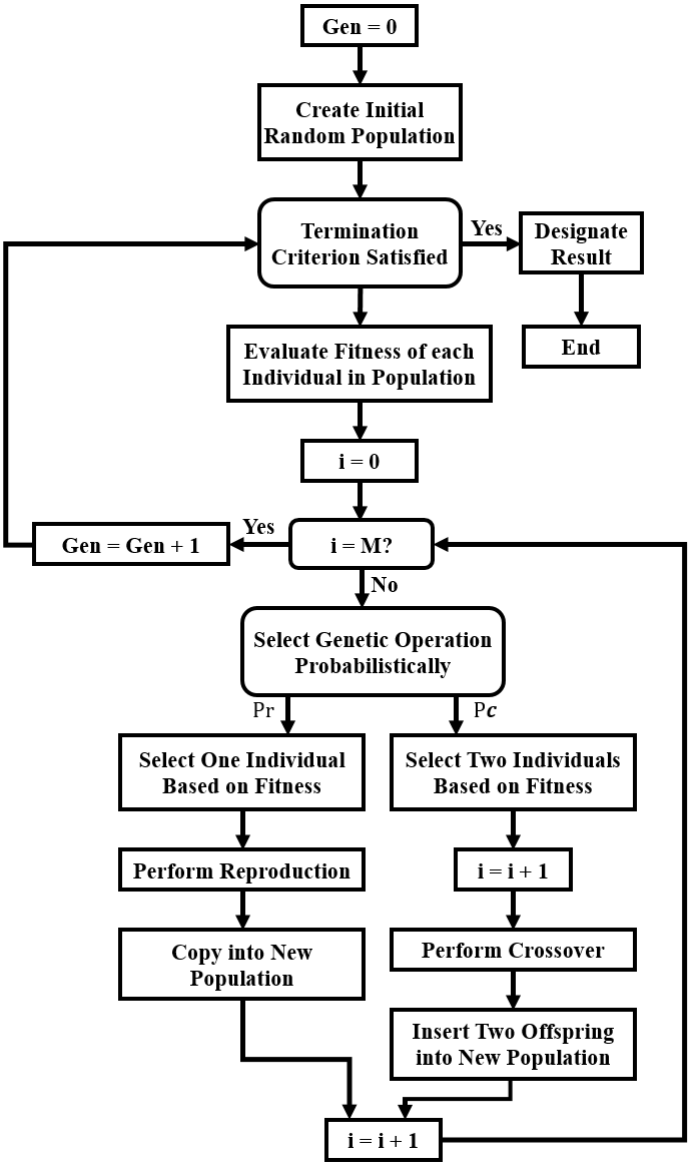

Figure 3.5: Flowchart for The Genetic Programming Algorithm [46]

3.1.2 Grammatical Evolution (GE)

Grammatical Evolution utilises binary string individuals with variable lengths that are mapped to computer programs via grammars. In 1998, Conor Ryan, J. J. Collins and Michael O'Neill introduced GE [50]. There are various techniques that utilised the grammars concept before the GE algorithm, however, GE provides a distinct method for utilising grammars in an evolution process [51].

To solve a problem using GE, it evolves solutions in a space defined by a Backus Naur Form (BNF) grammar which expresses programs in an arbitrary language. BNF is a notation approach for context-free grammars, often used to depict a language by setting a collection of grammars. This grammar tends to be symbolised by a four-tuple $\{N, T, P, S\}$ in which N is the non-terminal set (i.e. functions), T is the terminal set (i.e. inputs), P is the production rules set, and S is the start symbol of the grammar in which the generation process begins. Sentences in the language are derived by successive application of production rules to non-terminals (i.e. replacing a non-terminal that appears on the left-hand side of a rule with its productions on the right-hand side). A typical approach is to expand the leftmost non-terminal. The values in the integer array are interpreted as indices into the relevant production rules (i.e. they indicate which rule to apply). If a non-terminal can be expanded via k rules (indexed by 0 to $(k-1)$), then an integer value V is deemed to indicate the $(V mod\ k)th$ rule. Consider the following grammar as an example:

Predecessor	**::=**	**Production Rules**	**Index of Production**
S	::=	<expr>	0
<expr>	::=	<expr><op><expr> <pre-op><expr> <var>	0 1 2
<op>	::=	+ − / ∗	0 1 2 3
<pre-op>	::=	sqrt sin cos tan	0 1 2 3
<var>	::=	x 1.0	0 1

Table 3.1: BNF Grammar Example

The GE genomes are expressed via variable-length binary strings of 8 bits. This set of

bits produces what generally is referred to as a codon value which is utilised to select the production rule from the BNF grammar. Consider the following genome of a GE individual which is represented by a sequence of integers:

$$\{210, 35, 46, 67, 136, 53, 143, 25\}$$

Since S is the start symbol it must start the production rule and since it has only 1 expansion (as $< expr >$ and so we start the decoding process at this string. $< expr >$ has 3 rules and so we interpret 210 as 210 *mod* 3, i.e. as rule 0, and so we expand $< expr >$ as $< expr >< op >< expr >$. We now expand the leftmost non-terminal. Once again, $< expr >$ has 3 possible rules to expand it, and we select rule (35 *mod* 3), i.e. rule 2. Applying this rule gives the string $< var >< op >< expr >$. This continues until a final string is obtained with no non-terminals, see the illustration in Table 3.2.

Expression	**Gene Value**	**Number of Production**	**Index of The Selected Production**
S	...	1	0
<expr>	210	3	210 mod 3 = 0
<expr><op><expr>	35	3	35 mod 3 = 2
<var><op><expr>	46	2	46 mod 2 = 0
x <op><expr>	67	4	67 mod 4 = 3
x ∗ <expr>	136	3	136 mod 3 = 1
x ∗ <pre-op><expr>	53	4	53 mod 4 = 1
x ∗ sin (<expr>)	143	3	143 mod 3 = 2
x ∗ sin (<var>)	25	2	25 mod 2 = 1
x ∗ sin (1.0)			

Table 3.2: GE Decoding Process

If we run out of integers before this occurs, a recovery strategy needs to be adopted, e.g. continuing from the first element of the array, a strategy called wrapping [50]. However, if the sentence is not valid after a certain number of wraps, GE will consider this chromosome to be invalid.

GE creates the initial population from randomly produced chromosomes, each considered as a candidate solution to the problem [52]. By default, a single-point crossover is used to generate new solutions. In a single-point crossover, two positions are chosen on each binary string at random and their genetic contents are substituted beginning from these positions (see Figure 3.6). In addition, the mutation operator mutates at random a single bit on a particular GE genome with a predetermined mutation probability. Mutation is essential to preserve the diversity of GE individuals through the evolutionary process [52].

Figure 3.6: Crossover Operator on Grammatical Evolution

The steady-state replacement mechanism is applied in the GE algorithm by default to ensure the validity of solutions in evolution [51]. The invalid solutions are given the lowest fitness value, however, using a simple replacement mechanism these solutions may possible remain in the next generations. These invalid solutions may cause a delay in the evolution process. Thus, the positive effect of having the steady-state mechanism is its capability to replace the worst solutions in a population with the new solutions. The following pseudo-code summarises the optimisation steps of the GE algorithm [52]:

1: create initial genotypes
2: determine crossover parameters based on grammarDef.
3: If not given, determine the optimal popSize and iterations
4: $generation \leftarrow 1$
5: $genotypes \leftarrow Add\ suggestions$
6: $genotypes \leftarrow Add\ popSize - len(suggestions)$ random chromosomes of length $seqLen * numExpr$
7: **while** generation < iteratons **do**
8: **for all** genes in each genotype **do**
9: $phenotypes \leftarrow GrammarMap$ (grammarDef, gene, wrappings)
10: **end for**
11: $fitnesses \leftarrow evalFunction(phenotypes)$ [using plapply]
12: **if** terminationCost is given and min(fitnesses) < terminationCost **then**
13: terminate
14: **end if**
15: $genotypes \leftarrow genotypes$[sort by fitness]
16: $newGenotypes \leftarrow genotypes$[1:elitism]
17: **for** i := (elitism+1):popSize **do**
18: $parent1 \leftarrow Select\ using\ Roulette\ Wheel\ operator$
19: $parent2 \leftarrow Select\ using\ Roulette\ Wheel\ operator$
20: $newGenotypes[i] \leftarrow Crossover$(parent1, parent2, crossover parameters)
21: **if** random number > mutationChance **then**
22: Mutate newGenotypes[i]
23: **end if**
24: **end for**
25: $genotypes \leftarrow newGenotypes$
26: $generation \leftarrow generation + 1$
27: **end while**
28: bestExpression = EvalExpression(grammarDef, phenotype with best fitness)

3.1.3 Cartesian Genetic Programming (CGP)

"Cartesian Genetic Programming is a form of automatic evolution of computer programs and other computational structures using ideas inspired by Darwin's theory of evolution by natural selection" [53]. It was invented by Julian Miller in 1999 and the representation of electronic circuits was the idea behind its creation. In CGP, a program is encoded as a linear string of integers representing a directed graph. Each program is divided into subsets of genes (i.e. genotype) of the same length, representing the nodes of the graph. The genotype describes from where a node receives its data, what functions the node carries out on the data, and from where the outcome data wanted by the user is to be collected [53]. During the genotype decoding process, a number of nodes are disregarded. This comes about when these nodes' outcomes are not utilized in the computation of the final output node. When this occurs, these nodes and their genes are considered as 'inactive'. Previous investigations [54], [55] have shown how a significant percentage of inactive nodes can help the efficiency of the evolution process. Unlike a tree representation, where the route among any couple of nodes is always unique, the graphs representation permits the reusability of any path among a couple of nodes. Thus, this helps reduce the computational cost of functions, since the previously computed subgraphs can be used again. GP and GE algorithms are susceptible to bloat, a phenomenon where a large portion of the evolved program code has no influence on the fitness, but whose execution still consumes resources (and so typically extends execution times). CGP avoids such bloat [55].

CGP's genotype decoding process is recursive in nature and starts with the output genes at the beginning [53]. CGP programs may have as many output nodes as necessary. These final output nodes are deemed to be 'active'. The decoding process identifies the nodes whose outputs are used as inputs to these nodes. These input nodes themselves become 'active' and the same procedure is repeatedly applied until the full function is identified, ending with the identification of appropriate terminal input nodes. The decoding process extracts the active nodes; inactive nodes are not dealt with and so having inactive genes presents small computational effort. The user defines the number of nodes. Consider the example shown in Figure 3.7. There are 3 input nodes, which are indexed by 0, 1, and 2 (these do not form part of the genotype, but may be indexed by it). The remaining (computational) nodes of the system are now numbered contiguously, from 3 to 8. System inputs and computational nodes are therefore numbered contiguously over the range 0 to 8.

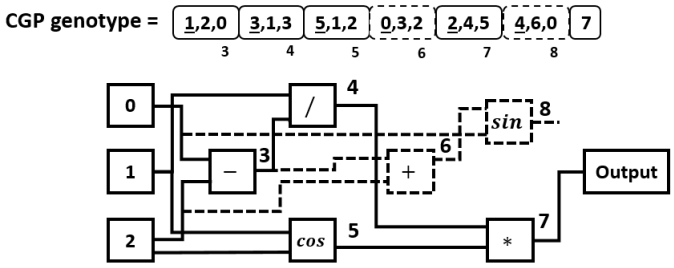

Figure 3.7: CGP Genotype and Corresponding Phenotype

In the above figure, the genotype contains structures of three integers for each of its computational nodes. The underlined genes in the genotype encode the specific function for each node. There are six possible functions, denoted by the integers 0 to 5: add (0), subtract (1), multiply (2), divide (3), sin (4) and cos (5). The remaining integers in each node structure are associated with terminal inputs or with other function nodes that are located to the left. The number of actual inputs depends on the arity of the function. The number of formal inputs to a computational node is the maximum arity of any function in the function set. Any extra inputs will be neglected by function nodes that require fewer inputs than this maximum. The last nodes in the genotype identify the output nodes. Here there is only one such node, which takes the value 7. The choice of node 7 as the output node coupled with the value of the earlier node which triples it and induces the active and inactive status of all earlier nodes.

Crossover operators have not been widely adopted in CGP. In the beginning, a one-point crossover operator was applied but caused disruption to the subgraphs inside the chromosome [53]. In another investigation [56], a new floating-point crossover operator was found to improve performance for symbolic regression problems. However, extra work is needed on a variety of problems to be able to evaluate its successes.

The mutation operator is utilized in CGP. A point mutation type is used in which a value at a randomly chosen gene location is replaced with another valid random value. If the chosen location is a function then it should be replaced with any function label from the function list. Whereas if an input location is selected then a valid value will be the output from any prior nodes or any of the input nodes (i.e. dataset features). In addition, a valid value for an output location is the label of the output of any node in the genotype or the label of the input nodes. The user specifies the mutation rate. An example of the point mutation operator is shown in Figure 3.8. A single point mutation takes place in the program gene, altering the output node input connection from 7 to 8. This makes nodes 6 and 8 active ones, whilst causing nodes 4, 5 and 7 to

become inactive. The inactive sections are displayed in dashes. This demonstrates how a small change in the genotype can at times cause a large change in the phenotype.

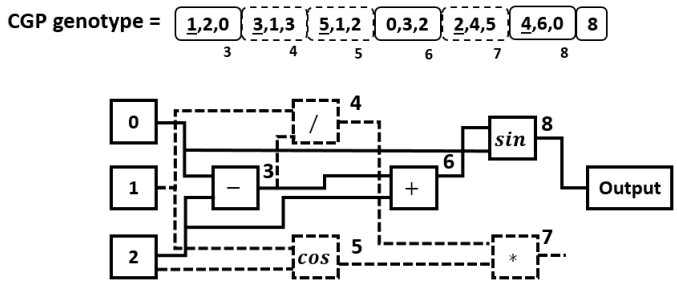

Figure 3.8: CGP Genotype and Corresponding Phenotype after Mutation

For individual selection the (1+ λ) Evolutionary Strategy (ES) is normally used in CGP. Usually λ is selected to be 4. ES sets up a single parent to reproduce 4 children utilising the mutation operator at each generation. The best individual between the parent and children is retained in the next generation and the procedure is replicated. In an ES approach, child genotypes are favoured over the parent. The child replaces his parent when child genotypes have similar fitness as the parent and there are no children that are superior to the parent. This is an important characteristic of the paradigm as it helped to makes good use of redundancy in CGP individuals [53]. A very small population size is normally used in CGP [56]. Thus, a large number of generations are expected to be utilized. In spite of this, numerous studies have been noticed that the average number of fitness assessments needed to work out various problems can be in favour of CGP when compared with other kinds of GP [53].

3.1.4 Multi-objective Evolutionary Algorithms

In many practical problems, it is common to have more than one desired objective to be optimised. Thus rather than assessing solutions based on a single fitness score, every potential solution posses a vector of fitness scores, one for each objective. This is usually known as a multi-objective optimisation problem. The focus of the multi-objective concept is to discover the group of solutions with the best (or near best) trade-offs possible amongst all the objectives. Multi-Objective optimisation seeks to optimise multiple often conflicting objectives at the same time. The multi-objective optimisation problem is described in [57] as "the problem of finding a vector of decision variables which satisfies constraints and optimises a vector function whose elements represent the objective functions. The term optimise is defined as finding such a solution which would give the values of all the objective functions acceptable to the decision maker since the objectives are usually in conflict with each other". In contrast

to the single objective optimisation, multi-objective optimisation has fitness function vectors that belong to a multi-dimensional objective space [58]. These vectors can be either minimised or maximised, susceptible to a number of restrictions and changeable boundaries.

In general, a multi-objective optimisation outcome is not unique; it is the set of the best (or near best) solutions possible known as the Pareto set. In this set, a solution x is considered dominating another solution y (i.e. $x > y$) when there is no criterion of y is better than the equivalent component of x and at least one objective value of x is strictly better than y. Formally and considering maximisation (where i, $j \in$ objectives space):

$$x > y \; if \; \forall i : xi \geq yi \wedge \exists j : xj > yj$$

The Pareto front (i.e. Pareto set) consists of the individuals that are not dominated by another individual. This basically means it contains potentially the best (or near best) solutions which provide a variety of trade-offs between objectives. Figure 3.9a illustrates a pairwise comparison between objective function one ($f1$) and objective function two ($f2$) achieved by every solution. A non-dominated front can be produced by applying the above explanation and checking if there is a solution that dominates another can be proven. Figure 3.9b displays an example of non-inferior solutions spread on the Pareto front, they are points D, E and F. For instance, E dominates C since E($f1$) is better than C($f1$) and E($f2$) is equivalent to C($f2$). Whereas points E and D are not dominating each other since E($f1$) is better than D($f1$) and E($f2$) is worse than D($f2$).

(a) Solution Space

(b) Non-dominated Solution

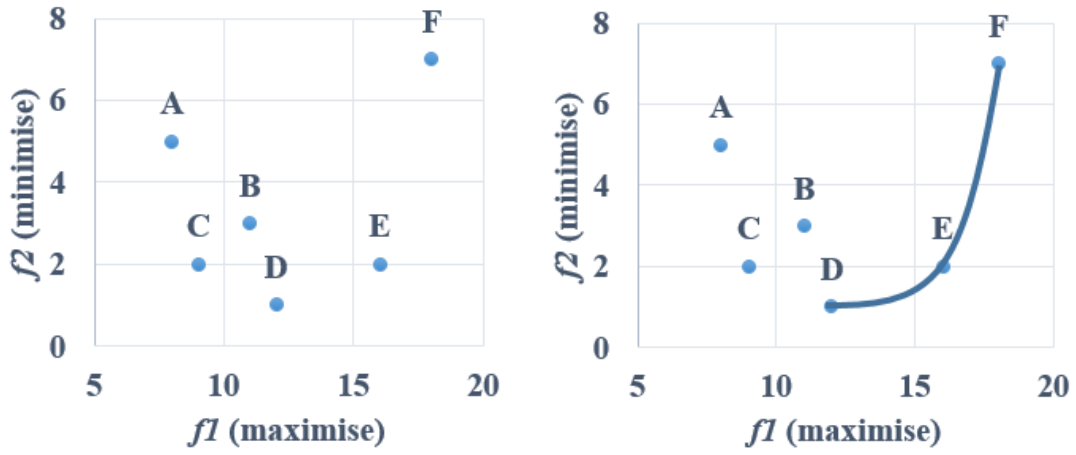

Figure 3.9: An Example of a Pareto Front [58]

The advantage of trade-offs among non-dominated solutions renders many system designers interested in exploring a broad range of them prior to making a final choice. EC algorithms have extensively displayed their potential in resolving many search and optimisation problems over the last three decades. Moreover, they have the ability to deal with problems with multiple objectives, which their traditional counterparts lack [58]. Multi-objective evolutionary computation (MOEC) makes use of the evolutionary search to perform multi-objective optimisation. The population-based nature of evolutionary algorithms is the main motivation behind using EC algorithms to solve problems with two or more often conflicting objectives. This enables the production of several solutions of the Pareto front in a single run [59].

3.1.4.1 Advantages of Pareto Front Based Approaches

In principle, a multi-objective optimisation problem in evolutionary computation is using two methods (i.e. weight-based and Pareto-based) to calculate fitness values [60]. In weight strategy, a single fitness function is outlined and the relations between objectives are revealed by weights. However, choosing the appropriate weights is not an easy task since it has an affect on the obtained solutions. Pareto front approaches own a number of advantages compared to the classical weighted approach:

- The prior determination of weights that affect the trade-offs among objectives is not needed anymore. Besides, figuring out such weights will require a deep understanding of the problem context.

- The relationship among objectives can be revealed easily using the Pareto front. This is especially important information as it helps the system designer to select the best solution.

- The obtained Pareto front can be saved and later restored in case of any changes in requirements which may demand a different solution and thus a different set of trade-offs.

- The issue of local optima can be overcome by optimising multiple fitness functions. This will help to maintain the diversity of the population and raises the opportunity of discovering better solutions.

The weight-based strategy does not permit the examination of objectives individually or even compare trade-offs between them. It describes different trade-offs as equal. Therefore, SPEA2 [60], which is among the most popular Pareto-based evolutionary algorithms, is utilised in this research.

3.1.4.2 Strength Pareto Evolutionary Algorithm (SPEA2)

As implied by its name, SPEA2 is an upgrade from its earlier implementation. SPEA2 is considered as an elitist method that employs a fixed size archive (external set) to preserve non-dominated solutions discovered in each generation. In order to generate a new population (i.e. offspring) in this method, tournament selection is utilised to chose individuals from the old population and perform evolutionary operations. The archive set members have a better probability to be chosen compared to other individuals [60]. The process of selection is dependent on the fitness scores of the individuals.

Because it is using an archive with a fixed size, we can witness three possible scenarios: non-dominated solutions are equal to, less than, or greater than the predefined archive size. There is no action needed when it is equal. In case of less than, the archive is filled using dominated solutions. If it surpasses the archive size, additional archive solutions are removed using a clustering technique which maintains the properties of the non-dominated solutions [60]. This procedure is repeated until the archive is able to accommodate the non-dominated solutions. In essence, the aim is to have a variety of non-dominated solutions (i.e. widely distributed over the objective space). The solution with a smaller fitness score is better. The fitness score of an individual is based on two variables as in the following equation:

$$Fitness\ (i) = Raw\ fitness\ (i) + Density\ estimation\ (i) \tag{3.1}$$

The raw fitness is determined through the strengths of its dominators in both archive and population. The aim here is to minimised the raw fitness score. Thus if it is equal to 0 that represents a non-dominated solution. Whereas a high score of raw fitness implies that it is dominated by many solutions. However, in a situation when the majority of solutions do not dominate each other, the raw fitness score alone is not enough. Therefore, SPEA2 utilised density estimation and added it to the raw fitness score to yield the final fitness score. The density estimation practice is an adaptation of the k-th nearest neighbour method. The density estimate is assigned the inverse of the distance to the k-th nearest neighbour in the objective space, in which k is equivalent to the square root of the sum of the whole individuals (i.e. both regular population and archive). The main steps of the algorithm [60] are given as follows:

Input:	N population size $\overline{N}$ archive size T maximum number of generations
Output:	A non-dominated set
Step 1 :	**Initialisation** generate an initial population ($P0$) and create an empty archive ($\overline{P0} = \emptyset$)
Step 2 :	**Fitness assignment** calculate fitness values of individuals in Pt and $\overline{Pt}$.
Step 3 :	**Environment selection** copy all non-dominated individuals in Pt and $\overline{Pt}$ to $\overline{Pt+1}$. If size of $\overline{Pt+1}$ exceeds $\overline{N}$ then reduce $\overline{Pt+1}$ with the truncation operator, otherwise if it is less than $\overline{N}$ then fill $\overline{Pt+1}$ with dominated individuals in Pt and $\overline{Pt}$.
Step 4 :	**Termination** if termination criteria ($t \geq T$) is satisfied then set A to the non-dominated individuals in $\overline{Pt+1}$ and stop.
Step 5 :	**Parent Selection** perform tournament selection with replacement on $\overline{Pt+1}$ to fill the mating pool.
Step 6 :	Variation apply variation operators to the mating pool and set Pt to the new population. Increase generation number ($t = t + 1$) and go to Step 2.

3.2 Ensemble Learning

The concept of combining outputs from a set of learners into a single output is described as an ensemble [61]. An ensemble may incorporate multiple (heterogeneous or homogeneous) learners to obtain reliable and more accurate predictions. A variety of schemes can be employed to produce learners and to incorporate them, i.e. different datasets can be used to train same learning paradigms and/or the same dataset can be used to train different paradigms [62]. The main challenge for the ensemble learning is the selection of the algorithms creating the ensemble and the decision function which combines the results of these algorithms. Of course, the more algorithms the better, however, it is essential to consider the computational cost of adding a new algorithm. In his review article [61], Dietterich listed three main reasons for utilising an ensemble-based system. The statistical reason is associated with the lack of sufficient data to accurately identify the best hypothesis in the search space. The computational reason is to address the concern that many ML algorithms may get stuck in local optima during the search for the best hypothesis. Finally, the representational reason is to overcome the issue of inability of many ML algorithms to effectively represent the sought decision boundary.

The construction of the ensemble model goes through two main steps: generation and combination [63]. The generation phase is responsible for creating a set of base classifiers. In the combination phase, the decision regarding how to combine the base classifiers outputs into one is made. Currently, a lot of the well-known contemporary

machine learning algorithms are in fact built around the ensemble concept. Bagging, boosting and stacking are considered to be the commonly used ensemble practices [24]. These methods aggregate multiple learning models over to a single model so that it will reduce variance (bagging), bias (boosting), or increase predictions (stacking). Figure 3.10 illustrates general ensemble methods workflow.

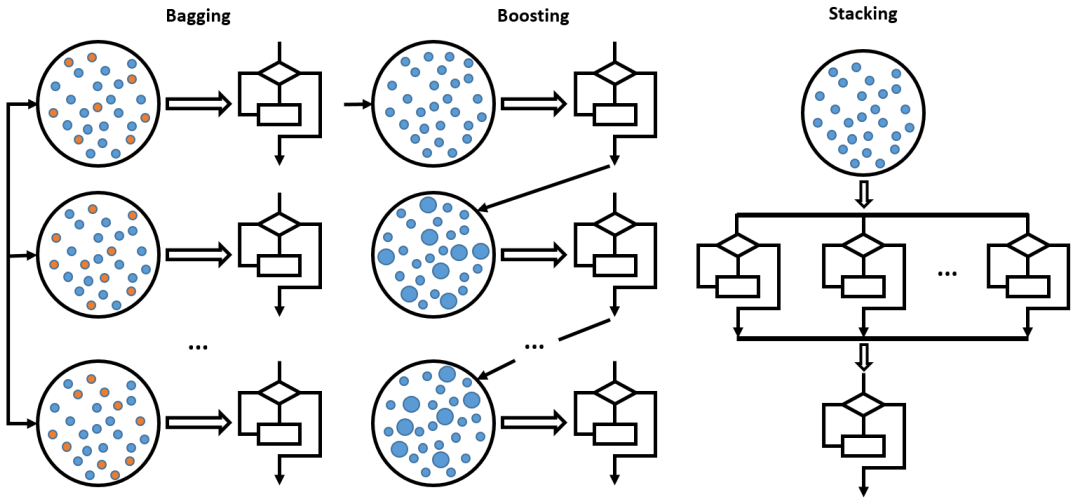

Figure 3.10: Ensemble Methods Workflow

Bagging is one of the earliest and most intuitive ways of assembling, where it implements in a parallel fashion. The models are fed a variety of bootstrapped copies from the training dataset so that generated models are little different from each other. These copies are subsets from the entire training dataset and each copy has been taken out randomly with replacement. Finally, the outputs from every predictor are assembled to obtain the final output of the ensemble. Another ensemble technique is boosting where algorithms are used sequentially. The first algorithm examines all the instances in the dataset and attaches weights to each of them. The instances with a higher value for the weight are the ones that were categorised incorrectly by the algorithm. Then, the next algorithm receives as an input the dataset as well as the weights for all instances in the dataset. The weights allow the algorithm to focus on the instances that were the hardest to categorise. The second algorithm updates the instances weight according to its result and passes the dataset to the third algorithm. This process continues until the last algorithm of the ensemble has processed the data. Making the final decision for both these methods could be reached by making use of majority voting. So for any instance input, the class that has been predicted the most by the learner models will represent the ensemble's decision. Nevertheless, there are alternative ways to analyse and to obtain an optimal combination.

The stacking technique is a supervised machine learning paradigm that searches for the best incorporation among a set of predictors (algorithms). In this type, a second level algorithm called "meta-learner" is trained to look for the ideal combination of the base learners. As opposed to bagging and boosting, the stacking method is often assembling strong, diverse groups of learners with each other. Furthermore, bagging and boosting when utilised usually produces homogeneous ensembles, in contrast to, stacking and voting which can be utilised to generate heterogeneous ensembles.

Not all combination approaches need to be learned [24]. A variety of means with good general motivation can be employed, e.g. "Majority Voting" may assign a class to an instance when at least the overall models indicate that a particular class is higher than half of the total number. Similar considerations apply for continuous outputs. If each classifier produces a set of class probabilities for a particular object, then the averages for each class can be computed, e.g. "Simple Average".

3.3 Previous Work

This section reviews various implementations of GP, GE, MOEC and EC ensembles in intrusion and anomaly detection systems. Machine learning algorithms have been widely utilised in the research of IDS. In the same way, EC algorithms, which include global optimisation strategies, are used to resolve this issue. When it comes to intrusion detection, the issue being resolved is the classification of normal and attack activities. The first attempt at utilising GP for intrusion detection can be dated back to 1995 by Crosbie and Stafford [64]. In this study, given a set of extracted features and a set of functions, GP built autonomous agents that were able to identify normal and intrusive scenarios. This method has the advantage of using many small autonomous agents rather than a single large one. However, the communication among them remains an issue. Furthermore, the training stage takes a longer time when agents are not adequately initialised. The experimental outcomes showed that GP was computationally efficient and powerful, in addition, it generated immediately usable programs.

Lu and Traore [65] investigated utilising GP to evolve rules whose aim is to discover novel or known network attacks on networks. The initially generated rules were selected based on past knowledge of known attacks where these rules were represented as a parse tree. GP then evolved new rules out of these rules. The approach was evaluated using the DARPA benchmark dataset and simulated attacks on a real network environment. Their investigation showed the ability of the new rules evolved by GP to identify novel intrusion signatures. The evaluation of the generated rule has low false rates as well as

a high detection rate for unknown attacks. Nevertheless, their utilisation of GP made the implementation procedure require more data and time during the training procedure. Note in this implementation, the detecting is for previously unseen variations of known attacks (i.e. DoS attack subfamilies).

Yin et al. [66] developed a network anomaly detection system using GP to generates the detection rules. These rules were evolved from previously learned rules aimed at classifying the attack traffic from the normal traffic. GP helped to minimise the set of rules used by the system by discarding underperforming rules. The classic DARPA dataset and real inside traffic were utilised to generate the rule set. For this dataset, an omit approach is also introduced to simulate the attributes of artefacts because of over-optimistic evaluation of detectors. This approach significantly outperformed the original rule learning algorithm, however, it underperformed when tested with real inside traffic. Furthermore, there were a number of attacks that went undiscovered from the testing stage. Finally, two stages of training are required by the GP algorithm that led to unwanted inefficiency.

Faraoun and Boukelif [67] attempted to make use of a dynamic GP for multi-class classification problem applied to create a network-based IDS. This approach evolves non-linear transformations of the original space (i.e. input data) and generates a new space with a decreased dimension that leads to a better multi-class classification. The KDD dataset was used to train and test the proposed approach. GP evolves non-linearly and can map the 41 attributes of the dataset into a one-dimensional decision space using a dynamic threshold to discriminate the 5 groups in this dataset (DOS, R2L, U2R and probing). The comparison with other existing solutions showed that the GP algorithm can provide competitive results without notably increasing the training and detection runtime.

Orfila et al. [68] examined the implementation of GP to evolve a network-based IDS, specifically for the automatic construction of patterns to utilise in the discovery of the type of activities seen in a system. They produced very lightweight and straightforward patterns which can be comprehended easily by humans. Their experiment has been applied to traffic recorded at a medium-sized website of a business venture. A dataset consisting of packet header information between October 2004 and January 2005 which is publicly available [69]. The obtained results in this study support the notion that GP can be successfully used for intrusion detection. However, additional experiments need to be performed using other datasets to prevent the dependency of the outcomes on a single dataset. The function collection used provided an additional explanation

with regards to why an event should be considered as an intrusion. However, there is a need to include non-linear behaviour exploration in the solution space.

Blasco et al. [70] make use of a new IDS assessment measurement as a fitness function to guide GP search for IDS rules. They applied their approach to the KDD dataset. Their function set was reduced to the one that only produced two logical outcomes (i.e. 0 or 1). The comparison showed that GP produced rules used fewer nodes and function calls (i.e. shorter rules). Therefore, this method is increasing the throughput while decreasing the time required to deal with any monitor event. In addition, these rules are produced in a simple structure. Thus this could help us to understand the reasons behind why an activity is considered to be malicious. Their conclusions demonstrate that an intelligent utilisation of GP delivers a system that competes with state-of-the-art approaches when it comes to effectiveness, productivity and simplicity.

Besides GP, GE is another evolutionary computation paradigm that is adopted to tackle IDS issues. Wilson and Kaur [71] investigated the possibility of employing GE to evolve IDS rules on wired networks. It has been utilised to evolve detectors for different classes of attacks. They apply their method to the KDD dataset. GE is proven to be effective in producing computer programs for the IDS. Their GE implementation was restricted to a single hypercube of the utilised features which best infer a class rather than a non-linear separation.

Sen and Clark [44] applied GP and GE to build IDS in a mobile ad hoc networks environment. Their research shows that both algorithms can be utilised to produce efficient programs automatically for known attacks, namely ad hoc flooding and route disruption against routing protocol. GP and GE evolved a program for each attack separately. These algorithms were assessed on a dataset gathered from simulated behaviours with different patterns of traffic and mobility. The experimental outcomes showed that GP and GE are good at exploring complex relations and GP offers greater efficacy compared to GE under their (approximation to) ideal parameter settings.

In contrast to GP, GE has seen very limited applications in intrusion detection and improvements are very likely possible. Furthermore, no single study exists that utilised CGP for IDS applications.

A GP framework has been adopted to form an ensemble IDS model before. Folino et al. [72] implemented a distributed IDS using a GP ensemble model. GP operates in a distributed hybrid multi-island model-based environment to examine events in a network.

Every island includes a cellular genetic program whose purpose is to produce a detector, trained using data stored on each node locally. In addition, these programs were enhanced with the use of a boosting algorithm AdaBoost.M2. Every genetic program operates cooperatively, yet independently of the others. After building these programs, they are integrated to produce the ensemble model by implementing a majority-voting concept. Folino and Pisani [62] proposed another ensemble using GP operating on distributed memory and environments, with GP composing several machine-learning techniques into an ensemble. GP evolved non-trainable combiners using a portion of the training set (i.e. validation set). Thus the ensemble built and used without the need for a further phase of training. The function set (i.e. non-trainable combiners) applied to classifiers that were detailed as following: average, weighted average, multiplication, maximum and median. These functions are duplicated using various arity usually from 2 to 3. The final evolved combiner function is utilised to categorise new data i.e. the test set. Both experiments over the KDD dataset have been performed. Both preliminary experiments showed that the proposed systems yielded better results compared to the efficacies of state-of-the-art algorithms. However, the prediction is worse for the underrepresented classes (i.e. unseen variations of U2R and R2L attacks) in the training dataset.

A more recent GP ensemble implementation for classifying low rate DoS attacks is proposed by Picek et al. [73]. In this study, a one-class GP algorithm is utilised to evolve a set of classifiers where the training phase included only normal behaviour. GP classifiers produce a single value compared with a target interval in order to decide instances type. The fitness function was calculated using the accuracy of normal classification where it was multiplied by the number of features used by the GP tree. This is important since it forced GP to use more features from the data which may be relevant to anomaly detection afterwords. Finally, the set of GP classifiers are stacked using the majority vote and simple average with ensemble members ranging from 3 to 15. These models were tested using three datasets KDD, NSL-KDD and a private one. The results showed that GP is able to produce more efficient anomaly detection solutions compared to state-of-the-art algorithms.

We note that utilising GP for evolving ensemble solutions was also studied before, however, the GP part was understandably different from ours. While some research has been carried out on GP ensembles for IDS, no studies have been found which utilised GE as an ensemble method.

MOEC based approaches have been applied to intrusion detection problems before.

Badran and Rockett [74] presented a multi-objective GP as a method for features extraction for multiple-category classifiers through producing multi-dimensional decision space from the input space. The aim of this work is to improve the classification performance among all classes. The two objectives that were set to be optimised using a Pareto-based multi-objective algorithm were the tree complexity (i.e. number of nodes) and the misclassification error. Complexity optimisation helps to deal with the bloat in GP. This approach works on the original format of datasets with no need for a pre-processing phase. In addition, feature selection and extraction are evolved simultaneously. In this experiment, the KDD dataset was used in which the proposed method achieved better performance in comparison to KDD Cup winner, but with a substantially simpler classification framework.

Gomez et al. [75] carried out a multi-objective technique based on a Pareto framework (MOEA) within the detection unit of Snort for automatic rules generation for network-based IDS. MOEA aims to simultaneously minimise the number of FP (normal traffic considered an attack); and FN (attack traffic not detected). Wide ranges of solutions were obtained using two optimisation methods: single objective (i.e. aggregate function) and the Pareto front. The optimisation mode is used to adjust FP and FN parameters (i.e. α and β) in minimising the objective function ($\alpha * FN + \beta * FP$). The experiment showed that multi-objective optimisation could improve results and provide more solutions that are more suitable for practical purposes. In addition, the authors showed that increasing the number of individuals and the backtracking rate (a quality measure) increases the overall achievement of IDS solutions. However, the authors employed a small number of individuals (i.e. only 3, 5 and 10) which is considered not enough for classifying millions of instances like those that exist in the dataset used (i.e. KDD).

Hoz et al. [76] proposed a network-based anomaly detection approach that includes a multi-objective paradigm for the application of feature selection. Using fewer features helps to reduce the complexity and improves the overall performance. The multi-objective optimisation algorithm implemented in this study is the Non-dominated Sorting Genetic Algorithm II (NSGA-II). NSGA-II evolved the non-dominated solutions (i.e. Pareto front) through the optimisation process. These solutions hold adequate information that helps to decide the features that allow a better labelling for each class. For the classification purpose, authors made use of the distinguishing characteristics of a clustering technique named Self-Organizing Maps (SOMs). They have examined Growing Hierarchical SOM (GHSOM) a clustering procedure that enhances solution efficiency through a relabelling technique. This clustering procedure benefits from clus-

ters that are previously trained via GHSOM to provide a better classification for the incoming events. The experiments were performed with the KDD-NSL dataset. In this implementation, the authors used the most effective set of features rather than the full feature set which helped to reduce the GHSOM solution size. This provided a more accurate detection rate with a lightweight processing that led to an increase in the computational efficiency of the IDS. Consequently, it becomes feasible to the security team to carry out other actions (e.g. IP blocking), in real time. Jaccard index, which has been used as an indication for the set of features selected for each activity type, is utilised as one of the objectives. The experiment showed that the feature set that optimises the Jaccard index along with the relabelling techniques improved the results obtained compared to other proposed methods.

Kumar and Kumar [77] proposed a multi-objective genetic algorithm, namely Archive based Micro Genetic Algorithm 2 (AMGA2) and an ensemble concept to model effective intrusion detection. The AMGA2 approach conducted a simultaneous consideration of a number of objectives such as the detection rate of each attack class, error rate, accuracy and diversity. A MultiLayer Perceptron (MLP) is utilised to generate a wide selection of base classifiers. A three-phase approach was utilised in which AMGA2 evolved individuals with a simple genome structure in the first phase. A Pareto front of non-dominated solutions is achieved during this phase. In the second phase, an entire solution group is further enhanced in order to identify the most effective ensemble solutions. This is performed by examining the interaction of the solutions and another Pareto front is produced. In the third phase, a majority voting technique is utilised to combine the predictions of individuals to produce the final prediction of an ensemble solution. To evaluate and validate the proposed model, two datasets namely KDD, KDD-NSL and ISCX 2012 [78] were used. The results showed that the proposed model could produce a high detection rate especially for minority attack classes when compared with bagged MLP and boosted MLP techniques. However, the method used a static means for choosing a suitable ensemble model, and the diversity between individuals and their ensembles is preserved implicitly.

Sen et al. [79] proposed utilising GP together with multi-objective evolutionary computation to build IDS in mobile ad hoc networks (MANETs) space. In this work, SPEA2 was used to evolve intrusion detection programs that show trade-offs amongst security elements and the power consumed by each program. The aim was to optimise three objectives simultaneously (i.e. detection rate, false positive rate and energy consumption). Both single objective and multiple objectives applications were implemented and compared. According to the outcomes, the multiple objective implementation, under

some circumstances, explored the trade-off space more effectively. Furthermore, SPEA2 counteracts the appearance of bloat in evolved individuals which offers an improvement since their goal is to evolve small-sized programs. The authors extended their work in [44] by investigating the energy usage of cooperative intrusion detection programs. In this implementation, besides optimising the previous objectives they included the number of neighbours (i.e. nodes) in a cooperative IDS. Each neighbour engages in the detection by forwarding their local knowledge to other nodes in the network. Reducing the number of nodes will help reduce energy/bandwidth consumption. The results show that EC techniques helped increase the effectiveness of the cooperative IDS.

There is no previous work to our knowledge in applications of the SPEA2 algorithm to IDS problem properties such as processing times, diversity and unknown attacks. In addition, multi-objective implementation was understandably different in terms of the features selection mechanism, and the examined environment for memory consumption.

The summary of the reviewed papers is presented in Table 3.3

3.4 Evolutionary Computation: Why?

The internet and computer network environments are complex, dynamic and fast growing. These environments are becoming more sophisticated, as are the threats against them with attackers using new attacks or modifying existing ones. It is challenging to differentiate threats from normal behaviours in these kinds of environments. To overcome such difficulties various artificial intelligence algorithms have been utilised. Algorithms that have been used to build IDSs possess strengths as well as weaknesses. It is difficult to say one algorithm is more superior than others are. Furthermore, varied algorithms are in many cases utilised with each other to increase effectiveness. Intrusion detection architectures incorporating various algorithms are also proposed. Machine learning can be the power behind the automation of security processes that enable security teams to keep up with the velocity and scale of what is being deployed in such environments. In order to make adequate IDS design, It is not quite clear whether the human perception can be the best possible option for these contexts compared to algorithms. In addition to that, the resources restriction needs a variety of trade-offs to be considered between multiple conflicting objectives. Humans are not usually able to make good choices when complex trade-offs need to be made.

In this research, EC algorithms are proposed to discover automatic solution designs for such complex space. Although techniques from different disciplines have been pro-

posed to solve IDS problems, EC is a standout among the most promising approaches. The EC framework can easily address the IDS problem. We can often see what the program is doing. EC approaches are very flexible where obtained IDS programs have some degree of manual analysis making them easy to understand. Using these approaches, researchers [80], [81] have been able to show that the evolved programs utilise far fewer features in comparison to other ML approaches. EC approaches have a number of attractive features, such as creating human-readable and lightweight detection rules, and offering a collection of solutions with different trade-offs amongst conflicting objectives [21]. These features are very significant for security teams [68]. These algorithms are considered distribution-free methods (i.e. no previous knowledge is required regarding the data statistical distribution), and there is no need for data pre-processing as it functions on the original format of the data directly [46]. Data relationships can be explored through modelling linear, non-linear or any application-specific operations. They include an intelligent self-configuration environment for solution selection/construction [74]. This helps to perform a dynamic method for selecting the most discriminative features of observed behaviours. The generated solutions can be used directly in the intended environment and can also result in a human-readable format that allows to easily understand the logic behind its evolved decision. These attributes are the main reasons for choosing EC-based approaches. Furthermore, multi-objective applications enable us to improve multiple conflicting objectives at the same time. Therefore, they can be employed to evolve detection programs that are both effective (i.e. detect attacks without a high false alarm rate) and efficient (i.e. take into consideration resources restriction). These characteristics render EC very attractive for synthesise of intrusion detection programs.

3.5 Conclusions

This chapter provides an overview of the learning algorithms that are applied throughout this thesis. It starts with a discussion about different stages of the evolutionary cycle with a focus on the three implementations: GP, GE and CGP. It then examines how EC approaches can be extended to address the multiple objectives optimisation problems. Next, it covered the concept of ensemble and its kinds, which will be applied with EC frameworks to increase the performance of the detectors. In addition, it presented a survey of the applications of the proposed algorithms to the field of intrusion detection systems. Finally, the reasons why EC algorithms are adopted in this research is also reviewed.

Year	Authors	Algorithm	Dataset	Findings
1995	Crosbie and Stafford [64]	GP	Simulated Data	GP was computationally efficient and powerful, in addition, it generated immediately usable programs.
2004	Lu and Traore [65]	GP	DARPA	GP able to identify novel intrusion signatures. The generated rule has low false rates as well as a high detection rate for unknown attacks.
2005	Yin et al [66]	GP	DARPA and Simulated Data	GP evolved new rules out of initially learned rules, which outperformed the initial rules used for detecting network attacks.
2005	Folino et al. [72]	GP	KDD99	GP based on the ensemble concept proved an effective method for the intrusion detection problem.
2006	Faraoun and Boukelif [67]	GP	KDD99	GP can provide competitive results without notably increasing the training and detection runtime
2007	Wilson and Kaur [71]	GE	KDD99	GE is proven to be effective in producing computer programs for the IDS
2009	Orfila et al. [68]	GP	LBNL	GP rules much simpler than C4.5 algorithm and performed better.
2010	Blasco et al. [70]	GP	KDD99	GP delivers a system that competes with state-of-the-art approaches when it comes to effectiveness, productivity and simplicity.
2011	Sen and Clark [44]	GP, GE and SPEA2	Simulated Data	GP and GE are good at exploring complex relations. The SPEA2 technique helped increase the effectiveness of the cooperative IDS.
2012	Badran and Rockett [74]	MOGP	KDD99	MOGP provided a simplicity in the generated classifier with comparable efficacy.
2013	Gomez et al. [75]	MOEA	KDD99	MOEA considered more suitable for practical purposes than the single objective technique since it produces a wide set of solutions with different trade-offs.
2014	Hoz et al. [76]	NSGA-II	KDD-NSL	NSGA-II evolved non-dominated solutions that hold adequate information helped to decide the features that provide a better labelling for every class.
2015	Kumar and Kumar [77]	AMGA2	KDD99, KDD-NSL and ISCX 2012	AMGA2 produced ensemble solutions with high detection rates especially for minority attack classes compared to state-of-the-art ensemble methods.
2015	Folino and Pisani [62]	GP	KDD99	GP composed several ML techniques into an ensemble classifier that yielded better results compared to the efficacies of state-of-the-art algorithms.
2018	Picek et al. [73]	GP	KDD99, NSL-KDD and a private one	The stacked of GP classifiers was able to produce more efficient anomaly detection solutions compared to state-of-the-art algorithms.

Table 3.3: Previous Works Summary

Chapter 4 | Datasets Acquisition for Building Intrusion Detection

Datasets have a critical role in training and testing intrusion systems. This chapter describes the datasets we utilised in our experiments. Collecting datasets and analysing behaviours should increase awareness and the ability to detect attacks in the future. However, one of the main challenges of today's research is obtaining representative data to derive meaningful and comprehensive results and reflect upon them. Unfortunately, generating a reliable benchmark dataset is not an easy task. Various studies argued that finding the right dataset is the most significant challenge and the results of analysing small environments might not hold for large systems [12], [14], [17], [42], [82]. The environments used to generate these datasets along with the feature set and existing attacks in them, are described in this chapter. Lastly, conclusions are given.

4.1 Intrusion Detection Datasets

System behaviours are diverse, complex and constantly evolving. This may affect the ability of protection systems to detect and mitigate attacks. The effectiveness of evolved IDSs may depend very much on the realism of the dataset(s) used to train the system. Limited studies were performed on the evaluation of the characteristics of datasets used for building an IDS. A recent study [82] outlined the major criteria in datasets such as including environment configurations, capture full packet, contain attack diversity, maintain the traffic payload, generate metadata, extract the feature set and labelled behaviours.

There are various methods used for gathering data such as using a simulation environment, honeypot, firewall alerts, web-server access logs and deploying a high-performance network monitor system with payload capture. A honeypot is considered one of the most useful and effective tools to collect and observe attack techniques and behaviours [83]–[85]. Deploying a honeypot to play the role of a router will provide researchers with a malicious traffic dataset and help especially with capturing routing

traffic [86]. Firewall logs or IDS alerts are another source of data and can be used for making passive measurements of live networks [87] and gathering different parameters of HTTP header files that contain GET/POST requests data that pass values to various web applications [88]. Web server access logs are datasets that store detailed information about various users' access and their sessions and are considered to be useful ways to monitor different behaviour on the web [89]. Another example, deploying a high-performance network monitor system with full-payload capture for different periods in time on a website is given in [90]. In [44], the authors have used a network simulation environment as the source of data to train and test their IDS. The collected data could be enriched by adding various elements of information, for instance, the geographical localisation of packets' source addresses, passive OS fingerprinting on tcpdump logs, and well-known blacklists of IP addresses. Moreover, an enriched dataset may augment an entry with the relevant working days, working hours, holidays and so on. Honeypot environment descriptions, analyses of observed sources' ports and honeypot machines attacked in parallel, or in sequence, may also be used to augment the dataset [84], [91], [92]. Datasets may also encompass cases of multiple attack simulations, penetration tests and/or vulnerabilities discovery tools to ensure the dataset contains a wide range of threats [93].

Public datasets such as the DARPA/Lincoln Labs 1998 [94] and the KDD 1999 [95] are used by different studies as ready collected data and contain various attacks. Tsai et al. [23] found that KDD and DARPA datasets were used by most studies with only a very few studies using their own datasets. However, KDD and DARPA are presently over a decade old and thus are considered inadequate for current research [14], [82].

4.2 Dataset Feature Extraction

Typically, after the dataset gathering is completed, researchers will start a pre-processing stage to extract a feature set. These features will be used in training and evaluating the IDS prior to deployment. Features vary in their significance for the purposes of intrusion detection. To be able to draw out a set of features that describe environment behaviours various techniques could be executed, such as Argus [96], which monitors and analyses network flow and also produces detailed network flow status reports, and Flowcalc [97], which measures flow statistics from network traffic data. The selection of methods utilised to examine the intended environment certainly depends on the sort of investigation conducted in the following stages [98]. The available information elements vary in their significance for the purposes of intrusion detection. Some 'features' of data may collectively discriminate between malicious and acceptable behaviours.

Other features may simply serve to act as 'noise' for detection purposes.

Before applying learning algorithms to the problem of creating intrusion detectors researchers tend to apply feature reduction to increase system performance and reduce redundancy. Most datasets suffer from irrelevant and redundant features so eliminating superfluous features will improve the processing time and accuracy. Tsai et al. [23] noted during their review of 55 studies between 2000 and 2007 that the level of IDS classification accuracy improved by applying feature selection. Some experience has shown that certain features could provide a good indication of the type of attacks. Stevanovic et al. [89] demonstrated how using 9 features including 2 novel ones with 7 classification algorithms improved their performance during the experiments they conducted. Various manual and automatic techniques for identifying high performing features are available [93].

One of the most critical steps in building an IDS is deciding which set of extracted features to include. With very large datasets and models that have a high computational cost, impressive efficiency can be realised by identifying the most (and least) useful features of a dataset prior to running a model. High dimensionality of the explanatory variables can cause both long computation times and a risk of overfitting during the training phase. In addition to that, it is challenging to understand models that use a large number of features. Generally, there are two main kinds of feature selection methods: filter and wrapper [29], [99]. Filter methods consider the relationship between features and the desired output to compute the relative importance of features. Meanwhile, wrapper methods create models with a subset of feature and measure their model performances. It would be preferable to have the capability to choose the important features prior to the proposed algorithm's learning stage. This shortens the training time in addition to rendering it easier to interpret the outcomes.

4.3 Feature Type used in our Experiments

Our proposed techniques have been evaluated using five widely different types of datasets (i.e. environments). These datasets are fully labelled containing realistic normal and malicious scenarios. Extracted features contain three types: numeric, symbolic and binary. Using EC approaches has noteworthy advantages since it operates on the original format of utilised data with no pre-processing and with no presumption of any prior distribution [74]. However, we did not employ all the available features. Firstly, we did not use the symbolic type to avoid increasing the complexity of the learning algorithm due to the requirement to process symbolic features such as protocol type,

packet type and flag. In the same vein, avoiding adding an extra stage for pre-processing symbolic features in the IDS system layout as in [29], [39], [67], [68]. Secondly, some attributes are changing constantly such as IP addresses and port numbers due to many reasons, for example, spoofing by attackers for malicious purposes or hiding the actual IP by proxies for the legitimate aim of protecting privacy. Thus, any detectors relying on these attributes may not generalise well in real world applications [30], [100]. However, features that describe flow statistics between IP addresses or to/from port numbers are included.

4.4 Dataset Splitting for Learning Techniques

Adopted datasets are publicly available and they are used by the security research community. Unfortunately, different studies have used different splitting percentages (i.e. training and testing) of these datasets for various reasons such as:

1. Choosing only a few kinds of attack.
2. Compiling a small version of the dataset.
3. Creating a new dataset by combining its training and testing parts.
4. Filtering a dataset to fit analyses concerning the distribution of normal and attack data.

These discrepancies are an issue since one can argue that utilising different proportions of the dataset will promote different results. There is a large computation cost to mimicking each and every dataset's settings in order to compare them. One of our aims is to use a common testbed where there is a wide variety of security threats under examination. In addition, it should be easy for future research to replicate the results for comparison. Therefore, this study adopts the original split of used datasets provided by the originators and indicates the splitting percentages if required.

4.5 Datasets Description

4.5.1 Kyoto 2006+

The Kyoto 2006+ dataset [101] contains 3 years of real traffic data, from November 2006 to August 2009, gathered from the two honeypots and legitimate servers that were implemented at Kyoto University. They employed various kinds of real and virtual machines as honeypots such as Windows machines (e.g., Windows XP SP2, fully

patched Windows XP, and Windows XP with no patches), Linux/Unix machines (e.g., Solaris 8, MacOS X), mail servers, network printers, home appliances (e.g. TV set, HDD Recorder) and so on. The main known cyber threats that researchers observed in their honeypots were Trojans, Worms, Phishing attacks, Email spam and Shellcode. In addition, there is a percentage of unknown threats that did not activate any IDS alarms, and yet included shellcodes. They extracted 24 important features from the raw traffic data collected by the honeypot systems. There were 14 statistical features inspired by the 41 features of the KDD99 dataset; they also extracted 10 additional features to help investigate what sorts of threats were encountered more effectively. Table 4.1 shows the final feature set that was used for this study :

Id	Feature Name	Description
1	Duration	The length of the connection in seconds
2	Service	The connection's service type. e.g., http, telnet
3	Source Bytes	The number of bytes sent by the source IP address
4	Destination Bytes	The number of bytes sent by destination IP address
5	Count	The number of connections between the same source and destination IP addresses in the past two seconds
6	Same_srv_rate	% of connections to the same service in Count
7	Serror_rate	% of connection that have "SYN" errors in Count
8	Srv_serror_rate	% of connections that have "SYN" errors in Same_srv_rate
9	Dst_host_count	The number of connections whose source IP address is also the same as that of the current connection
10	Dst_host_srv_count	The number of connections whose service type is also the same as that of the current connection
11	Dst_host_same_src_port_rate	% of connections with the same source port of the current connection in Dst_host_count
12	Dst_host_serror_rate	% of connections that have "SYN" error in Dst_host_count
13	Dst_host_srv_serror_rate	% of connections that have "SYN" errors in Dst_host_srv_count

Table 4.1: Kyoto 2006+ Dataset Features [101]

For the experiments on Kyoto 2006+ dataset, we chose to evaluate our frameworks on the dataset collected on 27, 28, 29, 30 and 31 August 2009. We selected 152,460 samples randomly. We split it into 70% for training which contains 58.19% attacks and 30% for testing which contains 57.96% attacks.

4.5.2 Phishing Websites Dataset

Phishing is known as the art of mimicking an internet site of a credible enterprise aiming to steal customer's credentials and other sensitive data, for example, accounts

details and social security numbers [102]. The percentage of URLs related to phishing activity in web traffic grew by 182.6% in 2017 compared to 2016 [7]. The well-known phishing websites dataset produced by Mohammad et al. published in UCI repository [103] is utilised. This dataset consists of 11,055 website samples. It has 4,898 examples labelled as phishing websites, whereas the remaining examples are labelled as genuine. The phishing websites were obtained from Phishtank [104], which is an anti-phishing community site that provides a phishing verification means by which users can publish, confirm, monitor and share phishing data. Furthermore, they used Millersmiles [105], which is regarded as a leading source of reports regarding spoof emails and phishing scams. The genuine websites were obtained from the yahoo directory [106] as well as from the starting point directory [107]. Most of this dataset's attributes are binary (0, 1) or ternary (0, 1, -1) where the values 1, 0, and -1 refer to legitimate, suspicious and phishing, respectively. This dataset contains 30 features that have proven its ability to make sound and effective predictions for phishing websites. These important features have been classified into 4 categories according to their impact on a website. The description of every categorise and its features are as follows [102], [103]:

Address Bar based Features: this refers to all features associated with the address bar that displays the current Uniform Resource Locator (URL) of the analysed internet site.

1. **having_IP_Address** refers to using an IP address instead of the domain name in the URL, and sometimes may even be changed to hexadecimal code.
2. **URL_Length** relates to hiding a suspicious part by using a long URL.
3. **Shortining_Service** refers to adopting shortening technique services which make the URL smaller in length.
4. **having_At_Symbol** checks whether the URL's has a "@" symbol.
5. **double_slash_redirecting** The presence of // in the URL indicates there is a redirection to another website.
6. **Prefix_Suffix** using dash symbol (-) makes the URL seem legitimate for its users.
7. **having_Sub_Domain** indicates the number of sub domains in the URL.
8. **SSLfinal_State** refers to checking the trust and age certificates attached to HTTPS (Hyper Text Transfer Protocol with Secure Sockets Layer) of a website.
9. **Domain_registeration_length** indicates the domain expiry date where normally phishing websites live for a short period of time.

10. **Favicon** refers to a visual reminder of the website identity and they checked whether it is likely to be a phishing website if loaded from an external domain.
11. **port** is used to indicate whether a particular service is open or closed and examine its favoured status.
12. **HTTPS_token** using a HTTPS token in the domain part of the URL to manipulate the users.

Abnormal Based Features: contains all features arising from recording the abnormal behaviours revealed in the internet site.

13. **Request_URL** indicates whether external objects embedded in a webpage are loaded from another domain.
14. **URL_of_Anchor** used to examine the hyperlink tag (i.e. <a>) status in the webpage source code.
15. **Links_in_tags** used to check the percentage of links in used tags such as <meta>, <script> and <link> in a website.
16. Server Form Handler **SFH** refers to the handler of submitted information and it is considered phishing if it contains an empty string or "about:blank" and suspicious if information is handled by external domains.
17. **Submitting_to_email** indicates whether the personal information submitted was redirected to a personal email.
18. **Abnormal_URL** refers to a phishing behaviour when the hostname is not included in the URL.

HTML and JavaScript based Features: includes all features associated with HTML and JavaScript source code of the pages incorporated in the analysed internet site.

19. **Redirect** indicates how many times a website has been redirected.
20. **on_mouseover** refers to using JavaScript code "onMouseOver" in the webpage source code to show a fake status bar to the users.
21. **RightClick** using JavaScript code to disable right-click function, so webpage source code becomes unavailable to the users.
22. **popUpWidnow** indicates inviting the users to fill out their personal information through pop-up windows.

23. **Iframe** is an HTML tag that is utilised by phishers to make the browser execute a visual delineation for misleading purposes.

Domain based Features: consists of all the features taken from the domain part in the URL of the analysed internet site.

24. **age_of_domain** used to indicate the minimum age of the domain where usually the legitimate ones live for 6 months or more.

25. **DNSRecord** checks the Domain Name System (DNS) record of the website where the record of phishing websites is usually empty or not found.

26. **web_traffic** indicates the number of visitors to the website which helps determine its popularity. The not popular or not recognised websites refer to phishing behaviour.

27. **Page_Rank** aims to estimate how important the webpage is on the internet. The more the better.

28. **Google_Index** shows whether the website is indexed by Google or not. The phishing websites are usually not indexed.

29. **Links_pointing_to_page** measures the number of links pointing to a website. The more the better.

30. **Statistical_report** uses various published statistical reports on phishing websites to examine whether the host of the website exists or not.

Phishing websites dataset is divided randomly into 80% for training which contains 44.39% phishing instances and 20% for testing where the percentage of phishing instances is 43.96%.

4.5.3 UNSW-NB15 Dataset

The dataset from the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) is used [38], [108]. In this dataset, a hybrid of real up-to-date normal and abnormal/malicious network traffic activities were generated in a synthetic environment. The UNSW-NB15 dataset is regarded as complex due to it matching the behaviours seen amongst some types of attacks and normal traffic. This dataset contains over 2 million sample data elements from two different simulation periods. In UNSW-NB15, there are nine categories of attacks [38]:

1. Fuzzers: a technique where the attacker tries to uncover new and unknown vulnerabilities in a program, operating system, or network by feeding it with the widest possible range of unexpected input of random data to make it crash.
2. Analysis: Variety of intrusions which aim to penetrate internet applications using ports (e.g., port scans), emails (e.g., spam), and web scripts (e.g., HTML files).
3. Backdoor: an intruder attempts to gain remote access to a device through bypassing authentication methods. As a result, he/she will be able to alter files, steal sensitive data and/or install malicious software.
4. Denial of Service (DoS): an intrusion that causes computer resources to be so heavily used as to prevent the authorised requests from accessing a device.
5. Exploit: a sequence of instructions that make use of a glitch, bug or vulnerability and generates an unintentional or unsuspected behaviour on a host or network.
6. Generic: an attack that is executed against all block-cipher to produce a collision with no consideration of the details of how that block-cipher is implemented.
7. Reconnaissance: can be defined as a probe; an attack that gathers information about a computer network to evade its security controls.
8. Shellcode: an attack in which the attacker penetrates a slight piece of code starting from a shell to control the compromised machine.
9. Worm: a type of computer malware which once it is installed by unaware users it can spread itself on the targeted system. It causes damage to the host network by exhausting its bandwidth and/or modifying and deleting system files.

There were 47 different reliable features that were extracted from the raw network packets. The final features that we adopted in our experiments are listed in Table 4.3 alongside their descriptions. The UNSW-NB15 dataset features are generated utilising tools such as Argus and Bro-IDS. In addition, authors wrote procedures to create new features based on relations among extracted features (e.g., is_sm_ips_ports and ct_state_ttl). They included a variety of packet-based features and flow-based features. These features are grouped into five sets:

(a) Flow features: involve the identifier attributes between hosts (i.e. client or server) for instance IP address, port number and protocol type.

(b) Basic features: include protocol connection properties.

(c) Content features: contain the attributes of TCP/IP; additionally they include some properties of http services.

(d) Time features: involve the attributes time, for instance, arrival time between packets, start/end packet time, and round-trip time of TCP protocol.

(e) Additional generated features: This category can be further divided into two groups: general-purpose features, whereby each feature has its own purpose, in order to protect the service of protocols, and connection features that are built from the flow of 100 recorded connections based on the sequential order of the last time feature.

Table 4.2 compares the experimental data of UNSW-NB15 dataset alongside its distribution of each category between training and testing datasets.

Category	**Training Set**	**Testing Set**
Normal	56,000	37,000
Analysis	2,000	677
Backdoor	1,746	583
DoS	12,264	4089
Exploits	33,393	11,132
Fuzzers	18,184	6,062
Generic	40,000	18,871
Reconnaissance	10,491	3,496
Shellcode	1,133	378
Worms	130	44
Total Records	175,341	82,332

Table 4.2: UNSW-NB15 Dataset Distribution [38]

The UNSW-NB15 dataset was split with an approximately 60%:40% proportion of the training and testing datasets sequentially with no redundant instances amongst the training and testing dataset. 68.06% of training instances are attacks whereas 55.06% of testing instances are attacks.

Id	Feature Name	Description
1	dur	Record total duration
2	sbytes	Source to destination bytes
3	dbytes	Destination to source bytes
4	rate	Number of packets per second
5	sttl	Source to destination time to live
6	dttl	Destination to source time to live
7	sloss	Source packets retransmitted or dropped
8	dloss	Destination packets retransmitted or dropped
9	sload	Source bits per second
10	dload	Destination bits per second
11	spkts	Source to destination packet count
12	dpkts	Destination to source packet count
13	swin	Source TCP window advertisement value
14	dwin	Destination TCP window advertisement value
15	Stcpb	Source TCP base sequence number
16	dtcpb	Destination TCP base sequence number
17	smeansz	Mean of the packet size transmitted by the srcip
18	dmeansz	Mean of the packet size transmitted by the dstip
19	trans_depth	The connection of http request/response transaction
20	response_body_len	The content size of the data transferred from http
21	sjit	Source jitter (mSec)
22	djit	Destination jitter (mSec)
23	sinpkt	Source inter-packet arrival time
24	dinpkt	Destination inter-packet arrival time
25	tcprtt	Setup round-trip time, the sum of 'synack' and 'ackdat'
26	synack	The time between the SYN and the SYN_ACK packets
27	ackdat	The time between the SYN_ACK and the ACK packets
28	is_sm_ips_ports	If srcip = dstip and sport = dsport, assign 1 else 0
29	ct_state_ttl	No. of each state according to values of sttl and dttl
30	ct_flw_http_mthd	No. of methods such as Get and Post in http service
31	is_ftp_login	If the ftp session is accessed by user and password then 1 else 0
32	ct_ftp_cmd	No of flows that has a command in ftp session
33	ct_srv_src	No. of rows of the same service and srcip in 100 rows
34	ct_srv_dst	No. of rows of the same service and dstip in 100 rows
35	ct_dst_ltm	No. of rows of the same dstip in 100 rows
36	ct_src_ltm	No. of rows of the srcip in 100 rows
37	ct_src_dport_ltm	No of rows of the same srcip and the dsport in 100 rows
38	ct_dst_sport_ltm	No of rows of the same dstip and the sport in 100 rows
39	ct_dst_src_ltm	No of rows of the same srcip and the dstip in 100 records

Table 4.3: UNSW-NB15 Dataset Features Description [38]

4.5.4 Modern DDoS Dataset

A new dataset that contains contemporary kinds of Distributed Denial of Service (DDoS) attack is used [40]. DDoS is an attempt to disrupt an online service and make it unavailable for its intended users by overwhelming it with traffic from multiple sources. Radware researchers observed in 2017 that the prevalence of DDoS attacks rose 10%. Moreover, they witnessed an increase in application layer attacks compared to network layer attacks [11]. A network simulator (NS2) was utilised to generate this dataset. It simulated different types of the attack targeting application and network layers. The generated dataset includes 4 kinds of DDoS attack as follows: (HTTP Flood, SIDDOS, UDP Flood, and Smurf). The five types of traffic in this dataset are described below [40]:

1. **Smurf** is malicious network traffic done by spoofing IP addresses in a network for sending a massive amount of Internet Control Message Protocol (ICMP) echo request packets to the target server. With enough ICMP responses forwarded, the victim machine is brought down.
2. **User Datagram Protocol (UDP) flood** is a type of network layer attack, where the UDP is a connectionless protocol. The intruders dispatch a large volume of UDP traffic to overwhelm the target server, the server then becomes unresponsive to other clients.
3. **SQL Injection DDOS** is an application layer attack, where intruders begin from the client side, a web browser for instance, by inserting a malicious code element (i.e. SQL statement) and forwarding it to the server-side database. This code will be executed indefinitely making the service unavailable for legitimate clients.
4. **HTTP flood** is a type of application attack. HTTP flood attacks are volumetric attacks, where attackers send what seem to be legitimate HTTP GET or POST requests for a web server or application. The attackers act as a legitimate user requesting services whereas in fact they exhaust server resources responding to every single request.
5. **Normal** transaction data.

Table 4.4 lists number of instances in each category.

Category	No. of Records
Smurf	12,590
UDP Flood	201,344
SIDDOS	6,665
HTTP Flood	4,110
Normal	1,935,959

Table 4.4: Distribution of Modern DDoS Dataset Classes

In this dataset, each simulated network traffic is converted to 27 different features. The final feature set that we used in our experiments is presented in Table 4.5 with their description.

Id	Feature Name	Description
1	pkt_id	Packet identifier
2	pkt_size	Total packet size in bytes
3	seq_number	Sequence number
4	fid	Flow identifier
5	number_of_pkt	Total number of packets
6	number_of_byte	Total number of bytes
7	pkt_in	Total time of packet inside queue
8	pkt_out	Total time of packet outside queue
9	pkt_r	Time of packet received
10	pkt_delay_node	Time packet delay within node
11	pkt_rate	Average packet rate
12	byte_rate	Average byte rate
13	pkt_avg_size	Average packet size
14	utilization	Bandwidth utilization
15	pkt_delay	Total time packet delay
16	pkt_send_time	Time of sending packet
17	pkt_reserved_time	Time of receiving packet
18	first_pkt_sent	Time of first packet sent
19	last_pkt_reseved	Time of last packet received

Table 4.5: Modern DDoS Dataset Features Description

This dataset was randomly partitioned, and its training and testing parts have no redundancy (i.e. there are no duplicate records). To obtain realistic results, the dataset was split into 34% testing and 66% training. 10.37% of training set instances are attack samples compared to 10.44% in the testing dataset.

4.5.5 CICIDS2017 Dataset

A more recent dataset named CICIDS2017 that was intended for cybersecurity and intrusion detection research is utilised [41]. This dataset covers a more diverse set of attack scenarios than the previous datasets. The dataset was created by the University of New Brunswick's and Canadian Institute for Cybersecurity over a period of five days. They used a comprehensive network infrastructure to generate normal and the most up-to-date common attack behaviours. This network included necessary equipment such as router, firewall, switches, various kinds of server and different versions of the common three operating systems, namely Windows, Linux and Macintosh. In this dataset, the authors created six attack profiles based on the last updated list of common attack families and executed them by using related tools and codes. The CICIDS2017 dataset attack scenarios are [41]:

1. **Brute Force Attack:** This type of attack is based on a trial and error approach targeting the victim system until it succeeds. The main usage of this attack is password cracking, however, it can be used for discovering hidden pages and content in a web application.
2. **Heartbleed Attack:** It is a bug in the popular OpenSSL cryptography library, which is widely utilised in the implementation for securing communications over networks. This attack is performed by sending a malformed heartbeat request with a small payload and large length field to the vulnerable system to leak memory contents.
3. **Botnet:** A number of compromised devices connected to the internet and used by an intruder to execute different tasks. For instance, stealing information, sending spam, besides getting access to the device and its connection.
4. **DoS Attack:** Beside traditional DoS attacks, the authors implemented low rate DoS attacks where a single machine keeps connections open with minimal bandwidth that consumes the server resources and takes it down.
5. **DDoS Attack and PortScan:** Authors executed the latest updated list of common DDoS attack tools. PortScan is an attack that is executed to check ports status in order to identify available services that are currently running on a server.
6. **Web Attack:** This type of attack is coming out daily targeting various web applications and can expose an organisation's valuable resources to the outside world. They implemented various web attacks such as SQL Injection, in which an attacker generates a string of SQL commands, and then utilises it to force the database to reply with sensitive information. Cross-Site Scripting (XSS) occurs

when developers do not examine their code properly to discover the possibility of script injection. In addition, Brute Force over HTTP, which involves trying a list of passwords to discover the administrator's password.

7. **Infiltration Attack:** It is an attempt to compromise the network from inside via the utilisation of a vulnerable software. In the case of success, a backdoor will be installed on the victim's system which leads to the performance of various attacks on the victim's network, for instance, IP sweep, full port scan and service enumerations.

In order to extract the network traffic features, authors utilised the CICFlowMeter [109], which is a bidirectional flows based feature extractor written in Java that can extract 83 statistical features. These features are calculated separately in the forward and reverse directions from the observation of traffic flows. In the second step, they find the most influential feature set for classifying each category in this dataset out of extracted features, as shown in Table 4.7. The CICIDS2017 was not divided by the provider into training and test datasets; therefore, we split it into training and test datasets using a ratio of 70% and 30% respectively. Both portions contain 20% attack instances. We observed that the CICIDS2017 dataset contains missing and duplicate information. This information has been removed. Table 4.6 shows the CICIDS2017 dataset classes distribution between the training and test portions.

Scenario Label	**Training**	**Testing**
Normal	1,589,806	681,514
FTP-Patator	5,574	2,361
SSH-Patator	4,124	1,773
Heartbleed	7	4
DoS slowloris	4,068	1,728
DoS Slowhttptest	3,887	1,612
DoS Hulk	161,067	69,057
DoS GoldenEye	7,233	3,060
Web Attack Brute Force	1,062	445
Web Attack XSS	462	190
Web Attack Sql Injection	16	5
Infiltration	26	10
DDoS	89,580	38,445
Bot	1,335	621
PortScan	111,266	47,538
Total Records	1,979,513	848,363

Table 4.6: CICIDS2017 Dataset Scenarios Distribution

Id	Feature Name	Description
1	Bwd_Packet_Length_Min	Minimum size of packet in backward direction
2	Subflow_Fwd_Bytes	The average number of bytes in a sub flow in the forward direction
3	Total_Length_of_Fwd_Packets	The total number of bytes sent in an initial window in the forward direction
4	Fwd_Packet_Length_Mean	Mean length of a flow
5	Bwd_Packet_Length_Std	Standard deviation length of a flow
6	Flow_IAT_Min	Minimum inter-arrival time of packet
7	Fwd_IAT_Min	Minimum time between two packets sent in the forward direction
8	Flow_IAT_Mean	Mean inter-arrival time of packet
9	Flow_Duration	Duration of the flow in Microsecond
10	Fwd_IAT_Std	Standard deviation time between two packets sent in the forward direction
11	Active_Min	Minimum time a flow was active before becoming idle
12	Active_Mean	Mean time a flow was active before becoming idle
13	Bwd_IAT_Mean	Mean time between two packets sent in the backward direction
14	Fwd_IAT_Mean	Mean time between two packets sent in the forward direction
15	Init_Win_bytes_forward	The total number of bytes sent in an initial window in the forward direction
16	ACK_Flag_Count	Number of packets with ACK
17	Fwd_PSH_Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
18	SYN_Flag_Count	Number of packets with SYN
19	Flow_Packets/s	Number of flow packets per second
20	Init_Win_bytes_backward	The total number of bytes sent in an initial window in the backward direction
21	Bwd_Packets/s	Number of backward packets per second
22	PSH_Flag_Count	Number of packets with PUSH
23	Average_Packet_Size	Average size of packet

Table 4.7: CICIDS2017 Dataset Features Description [109]

4.6 Conclusions

IDS has been the subject of a great deal of research but remains a challenging problem. Various approaches were used for collecting data for the purpose of evaluating IDSs. Many evaluations have been carried out using a limited number of very old datasets. Furthermore, the research often investigates a very limited number of attacks and makes assumptions about the architectures of solutions. Thus it is difficult to compare the various implementations proposed. A common testbed that contains various datasets, or better yet, testing in genuine environments utilising various scenarios would be especially desirable for the intrusion detection community [25]. This chapter has summarised various issues regarding datasets used for building and evaluating IDSs. As a result of the investigative work performed, we have identified five datasets to be used in our experiments in this thesis. We can observe the following:

1. These are five radically different kinds of environment that were utilised to generate these datasets.

2. These are labelled datasets and are in current use by the scientific community.

3. These datasets contain at least two classes: normal and attack, with a variety of up-to-date attacks and their variations.

Chapter 5 | Performance Evaluation of Evolutionary Computation on Intrusion Detection

This chapter discusses the details of the implementation of evolutionary computation techniques in order to discover intrusion detection programs. Firstly, an overview of the framework that was utilised is introduced, then the application of each algorithm (GP, GE and CGP) to intrusion detection is outlined. The evolved programs are presented and explained. The results are reported and a comparison of the performance of the evolved programs is made. Lastly, the efficacy of the proposed techniques for evolving intrusion detection programs to detect unknown attacks are reported.

5.1 Methodology Framework

Our approaches use a supervised learning module based on evolutionary computation algorithms for detecting cyber threats. Our programs will be expressed over features extracted from available datasets. Other researchers have already extracted features from these datasets as part of their own attempts to derive intrusion detectors. Hence, we will start at the learning phase and measure the performance of each features relationship generated (i.e. behavioural signatures). During the training phase, we use three forms of the implementations: GP, GE and CGP. The training phase finishes when either all instances are categorised correctly, or else a maximum number of generations has been reached. The best-evolved programs then feed into the testing phase to be evaluated. The basic architecture of our approach for producing programs is shown in Figure 5.1.

Initially, an EC framework was constructed and it began with the process function set (i.e. operators) selection, since the function set affects the search for the decision boundary that is sought. We tested 6 different frameworks. The main differences between them are the utilised function set and how the final decision was made. The first

framework utilised a small set of mathematical operators (+, −, * and protected (/)) and the evolved output is compared with 0 as the threshold to make the final decision. The second used an enriched function set with more complex mathematical functions such as (exp, log, sqrt and ln). The Ephemeral Random Constants (ERC) concept with a range from 1 to -1 was introduced in the third framework. In the fourth framework, we evolved the discrimination threshold alongside the program instead of using 0. The inclusion of relational functions (>, >=, <, <=, == and !=) which return Booleans values (i.e. true: attack, false: normal) used in the fifth framework. The final and currently used framework which incorporated various mathematical, relational and logical operators is described in this chapter. However, due to applying proposed methods to five widely different environments. There is no ultimate winner from these frameworks as each framework performed differently on each environment (i.e. dataset). However, the currently employed methodology was the best-performing one collectively. In other words, the method used showed a steady performance across all the environments that were tested.

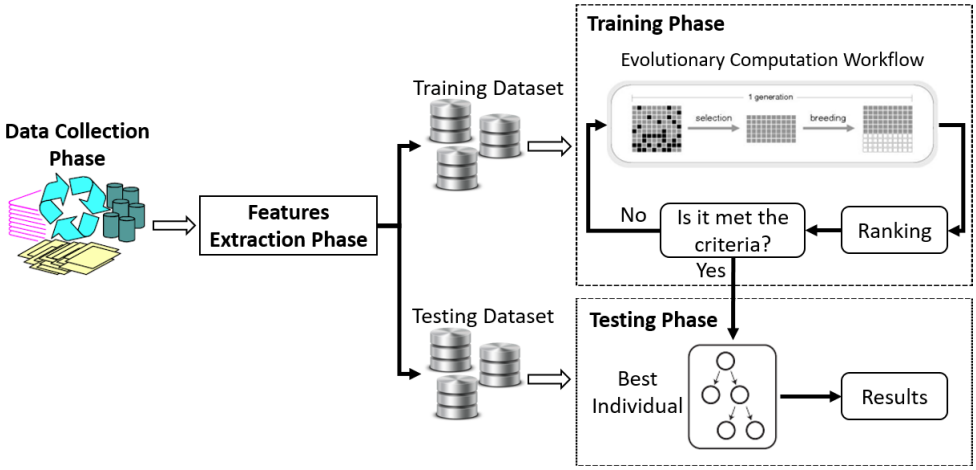

Figure 5.1: Methodology Framework

During each generation every individual in the population is subjected to the process of natural selection. The cost function used in our experiments is defined in equation (5.1) where we aim to maximise both TP and TN. Both are important since misclassifying normal behaviours wastes a great deal of time and can lead to loss of confidence. Low TP is an example of the IDS system not performing its primary tasks (i.e. to detect attacks).

$$Fitness = (\frac{TP - Anomaly\ Count}{Anomaly\ Count})^2 + (\frac{TN - Normal\ Count}{Normal\ Count})^2 \tag{5.1}$$

If all instances are classified correctly (the ideal) then the cost is clearly 0. In our experiments, both TP and TN are equally important (i.e. having the same weight

in the equation). We compared the cost function used in our experiments against two traditional metrics utilised to evaluate an IDS (i.e. classification accuracy and misclassification error). Figure 5.2 shows the comparison taken from the experiments of each proposed method using the UNSW-NB15 dataset. In comparison, Equation(5.1) achieved better performance than the other metrics in this case. However, no notable differences were found between these fitness functions when applied to other datasets.

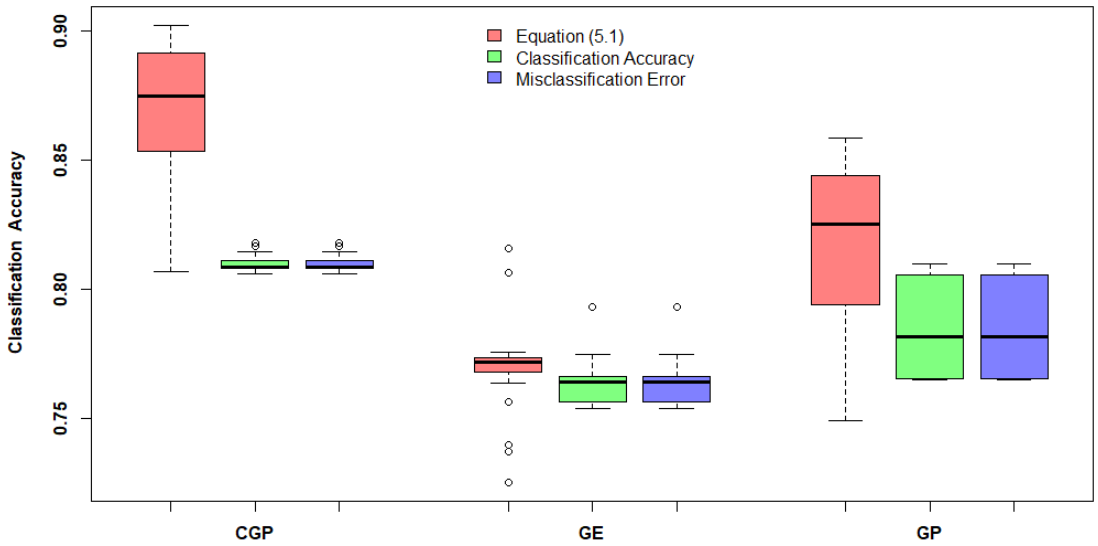

Figure 5.2: Fitness Functions Comparison

5.2 Application of Genetic Programming to Intrusion Detection

The method employed here is that GP enforces data type constraints (as input and output) in the evolved programs [49]. Normally GP uses trees as data structures. Inputs to the GP evolution process (i.e. instances to be classified) will be sets of feature values extracted from a dataset's instances. These features may occur directly in the dataset's instance values or else be derived from such values. For example, a dataset instance may comprise the fields of a network packet. An average of the most recent 10 packet lengths may also be synthesised as an attribute. Function nodes consist of a set of mathematical, relational and logical operators. These operators are of two types: a binary which takes 2 inputs and a unary which takes one input only. Relational nodes take 2 inputs and return an output of a Boolean type whereas logical operators take 2 Boolean inputs and return an output of a Boolean type. Function and terminal sets together with GP parameters settings utilised for individual representation are given

in Table 5.1. Using this set of functions, evolved individuals will cover both linear and non-linear separation. We provided a rich set of functions and expect our techniques to select judiciously from them. These functions have been used before to address IDS problems [44], [67], [79], [110]–[112]. The protected (/) is used to examine whether the second argument value is 0 and if so, it returns 1 as an output value.

Objective	Evolve intrusion detection programs
Function Set	The binary operators: +,−,∗,protected(/),power,max,min,percent The unary operators: sqrt, abs, ceil, exp, floor, log, ln, sin, cos, tan, tanh The relational operators: >, >=, <, <=, ==, != The logical operators: AND, OR
Terminal Set	The feature set given by utilised datasets
Population Size	1,000
Generations	50
Crossover Probability	0.9
Reproduction Probability	0.1
Tournament Size	7

Table 5.1: GP Settings

The ECJ [113] toolkit is used for the GP implementation. The GP parameters deployed in this research are given in Table 5.1. In addition, there is no elitism operation performed in the implementation. The rest of the parameters were determined automatically by the package (i.e. the defaults were assumed). The parameter selections could certainly influence the system performance. Nevertheless, predefined settings of these tools are utilised in order to make future comparisons easier. We can render no claim to having optimised these settings. In fact, identifying the best parameter settings is a very difficult task faced when EC algorithms are applied to nearly any problem. Our selection of parameters is intended to demonstrate that good results can be accomplished with these parameters. Better results could potentially be achieved by using greater computational resources. If IDS developers were to employ our methods to produce IDS rules for their specified environments then a degree of parametric examination would be advised. Once more, no claim to optimality can be made, but experimentation might proceed until the developers were satisfied with the evolved programs' abilities, and that also applies to the various other techniques.

This syntax produces an 'if' statement. The root node in GP tree either contains comparison or relational operators and returns a Boolean value. The following example is of an evolved program produced by the GP algorithm applied on the modern DDoS dataset:

```
(fid * number_of_pkt) <= Tanh[byte_rate]) &&
        (pkt_id <= number_of_byte)
```

Given that the output of proposed algorithms are algebraic expressions, we used the FullSimplify function, which "tries a wide range of transformations on an expression involving elementary and special functions and returns the simplest form it finds" [114], from Mathematica to simplify the outputs. Despite the promising results achieved by this implementation, it might be better to take variables values (i.e. dataset features) used by the algebraic expression into account in future investigations.

The GP algorithm is run 20 times on the training dataset and the best evolved program from each run is evaluated on the testing dataset. The relationship amongst classification accuracy and the number of generations in each dataset from the best-evolved program produced by the GP algorithm is shown in Figure 5.3. In most cases, the maximum fitness value is achieved at the 50th generation and it is steady thereafter. As can be seen from this figure, the best program shows a performance on the testing which is nearly as effective as on the training, with the exception of the UNSW-NB15 dataset due to its complexity. In addition, bloat has also been observed in our GP experiments. This problem could be resolved by punishing the cost function, for instance, by also including tree size optimising, or adapting multiple fitness functions. In a subsequent chapter, a multi-objective GP method that aims to optimise multiple objectives simultaneously is employed.

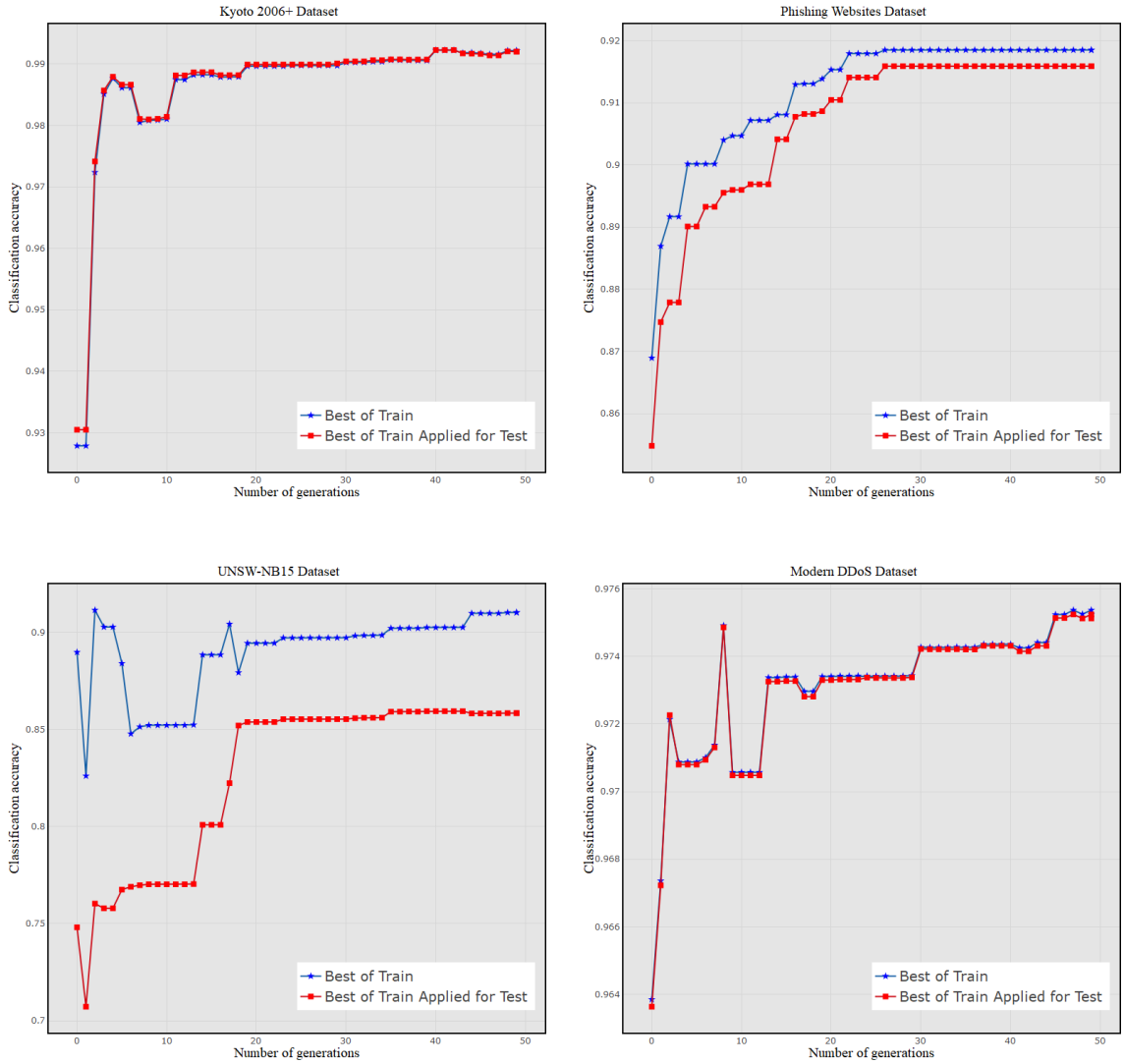

Figure 5.3: GP: Relationship between Accuracy and Number of Generations

5.3 Application of Grammatical Evolution to Intrusion Detection

For GE, the objective is to evolve computer programs to detect attacks. The problem is outlined using a grammar and a fitness function. Table 5.2 shows BNF grammars utilised by GE in order to evolve the solutions. This grammar produces an 'if' statement. This allows detection rules of the form of "if <condition> then raise alarm" to be generated. The input features and the functions utilised in this grammar are similar to those used in GP. The cost function described by equation 5.1 is also utilised in the GE algorithm also.

S	::=	<rule>
<rule>	::=	ifelse (<condition>) { raise alarm ( ), no alarm ( ) }
<condition>	::=	<condition>AND<condition>, <condition>OR<condition>, <rel.op>
<rel.op>	::=	<op>(<expr>, <expr>)
<op>	::=	>, >=, <, <=, ==, !=
<expr>	::=	<binary.op>(<expr>, <expr>), <unary.op>(<expr>), <var>
<unary.op>	::=	sqrt , abs , ceil , exp , floor , log , ln , sin , cos , tan , tanh
<binary.op>	::=	+, −,∗, protected (/) , power, max, min, percent
<var>	::=	The feature set given by utilised datasets

Table 5.2: The BNF Grammar Used for the Problem

The following is an example of an evolved expression (i.e. condition) produced by the GE algorithm taken from the UNSW-NB15 dataset:

```
(sload > sttl) && (sttl > Max[smean, Min[ct_dst_ltm,
    Log[ct_flw_http_mthd ^ Tanh[Ceiling[sttl]]]]])
```

GE was implemented using the package gramEvol [115] in this research. The GE parameters used in the experiments had a population size of 1000, the number of generations was set to 50 and without elitism. The rest of the parameters determined were automatically set by the package. The maximum search depth in the case of cyclic grammar and was limited to the number of production rules in the grammar. The number of integer codons in the chromosome is determined by sequence length per expression multiplied by the number of expressions. Random points are selected and a classic single-point crossover is performed. Each chromosome may mutate with a probability of a mutation occurring which ranges between 0 and 1. It is set to 1/(1+chromosome size)) for genetic algorithm and 10/(1+chromosome size)) for evolution strategy [115].

In Figure 5.4, the performance of the best-evolved individual generated by the GE algorithm is demonstrated for each dataset. It shows the relationship amongst the classification accuracy of the best-evolved programs and the number of generations. Although for GE based experiments we observed fewer evolved programs suffering from bloat than in their counterparts evolved from GP experiments. However, in contrast to the best individuals formed utilising GP with the same parameter settings and search space, the GE overall performance is less. An examination of the outcomes shows that the candidate solutions in several executions converge too early and become stuck at local optima in a rather early phase as shown in Figure 5.4. This issue could be overcome through the adaptation of the ensemble technique since it explores the search space from many different starting points [61].

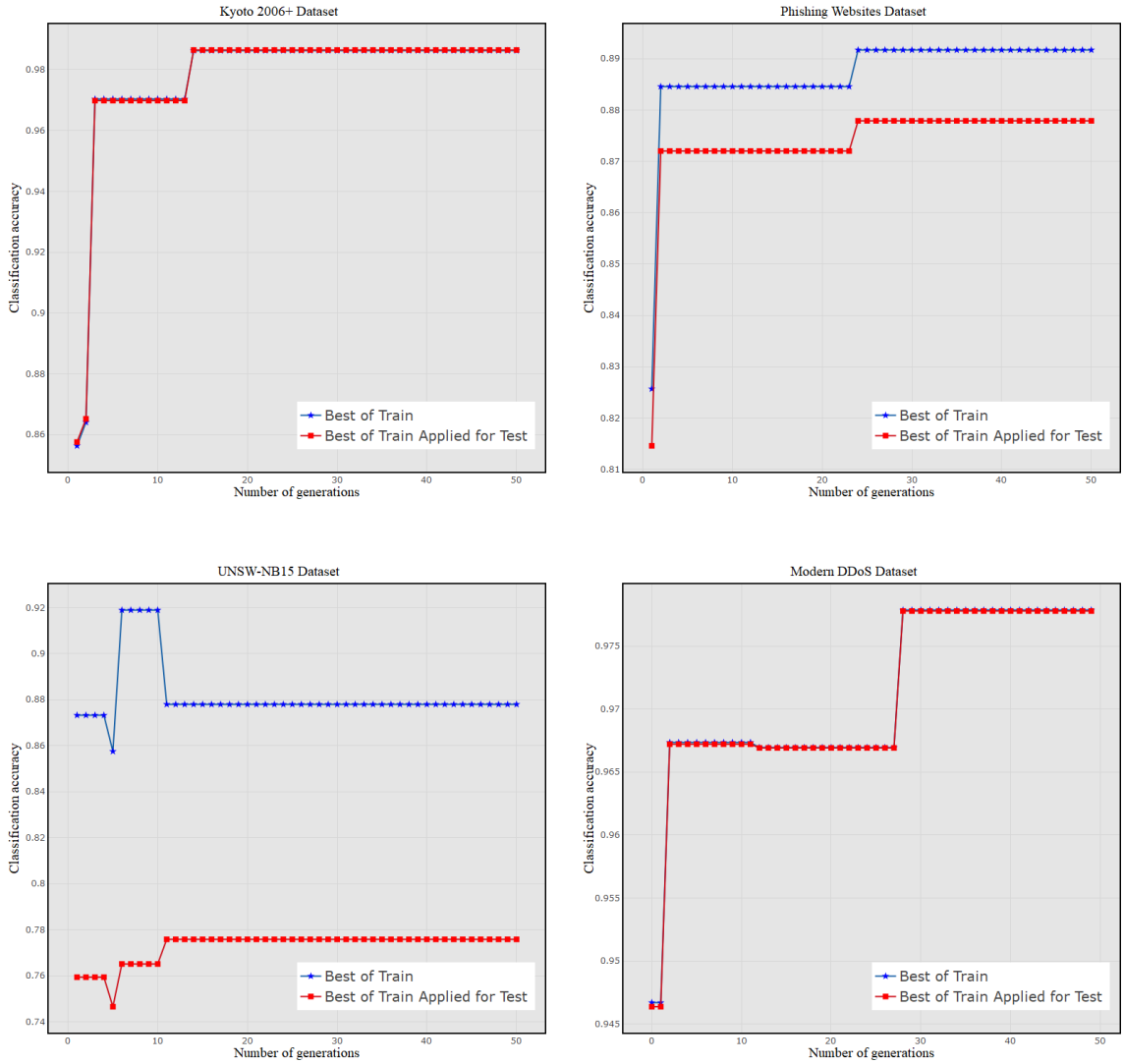

Figure 5.4: GE: Relationship between Accuracy and Number of Generations

5.4 Application of Cartesian Genetic Programming to Intrusion Detection

The same objective was set for CGP. A problem is outlined using function nodes along with input nodes (i.e. dataset features) which are parts of a CGP graph, and the cost function. Function nodes consist of a set of mathematical, relational and logical operators which are provided in Table 5.3. Equation (5.1) is used to evaluate a CGP individual. CGP parameter settings were determined empirically via preliminary experimentation. The rest of the parameters were decided automatically by the packages.

Objective	Evolve intrusion detection programs
Function nodes	Binary input arity: +, −,∗, protected (/), power, max, min, percent, >= , <, <= , == , ! =, AND, OR Single input arity: sqrt, abs, ceil, exp, floor, log, ln, sin, cos, tan, tanh
Input nodes	The feature set given by utilised datasets
Population Size	5
Generations	10,000 (except modern DDoS dataset where it is set to 2,000)
Mutation Probability	0.01
Number of Nodes	Maximum is 500 (not including input and output nodes)

Table 5.3: CGP Settings

The ECJ [113] toolkit is used to implement CGP. The number of output nodes for CGP experiments was set to 2. To obtain a binary classification decision (normal vs. anomaly), evolved output nodes compared their values using greater than operator (i.e. $o0 > o1$). The following is an example of CGP evolved outputs taken from the phishing websites dataset experiment:

```
Cos[Exp[Links_pointing_to_page <= Prefix_Suffix]] >
(URL_of_Anchor + (Min[Prefix_Suffix, Sin[Shortining_Service]]
|| Ln[SSLfinal_State]))
```

Figure 5.5 shows the relationship between accuracy and the number of generations for the best evolved solution over each dataset by CGP. For CGP experiments, 20 runs are performed. The best solution showed a similar overall performance in the testing and in the training with the exception of the UNSW-NB15 dataset. For the experiment on the modern DDoS dataset, the number of generations was reduced from 10,000 to 2,000 due to stagnation of the fitness value after reaching 2,000 generations for all runs. The results of the best CGP programs are better than those of their GP and GE counterparts.

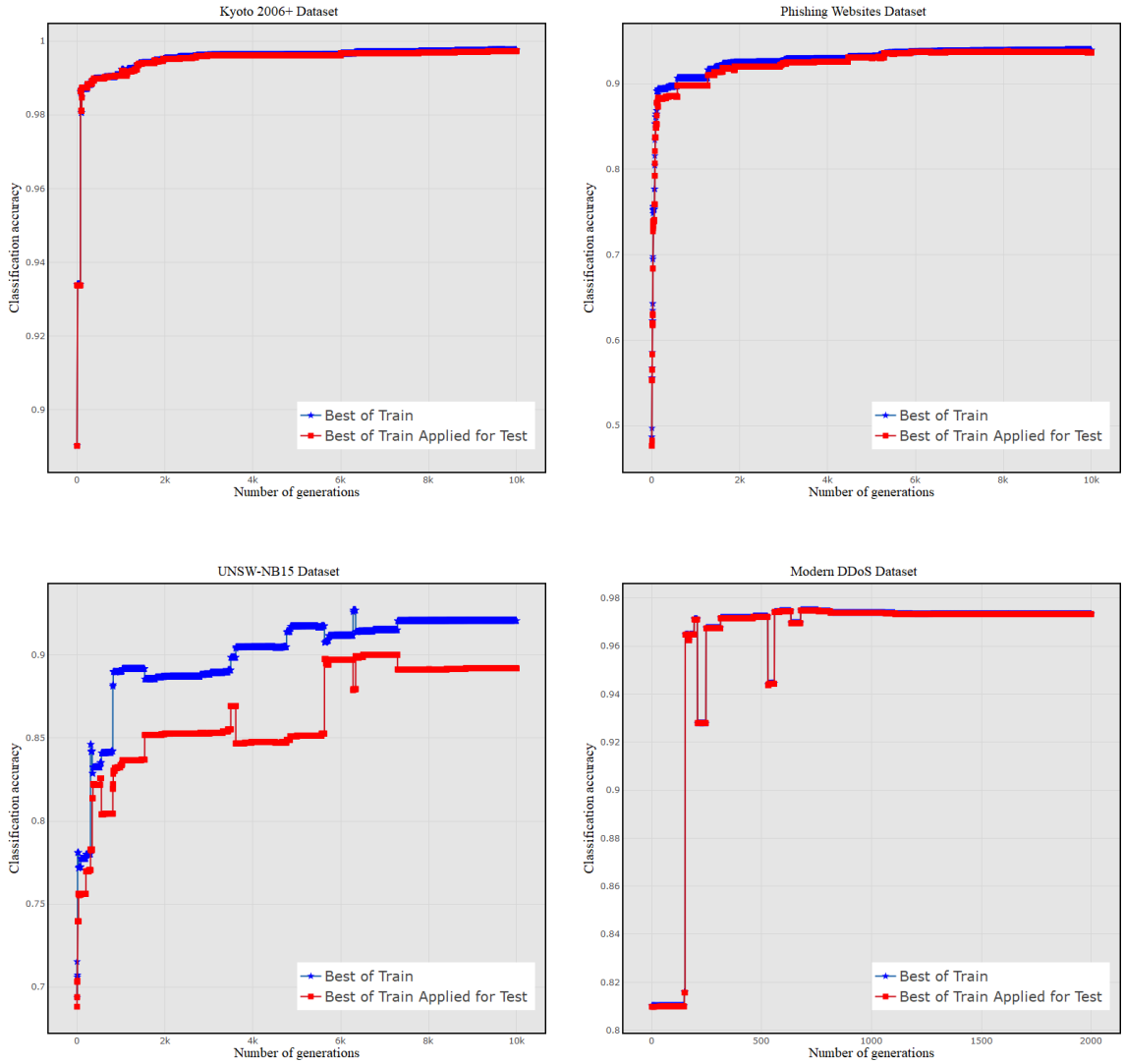

Figure 5.5: CGP: Relationship between Accuracy and Number of Generations

The functions utilised to represent the problem in all the proposed techniques are identical. In this thesis, employed GP enforces the rules (i.e. strongly typed) which is similar to the GE grammar. This helps the generation of conditionally structured rules (i.e. IDS programs). For instance, only relational or logical operators which return a Boolean value can be positioned at the root node of the GP tree. Thus, GP and GE outcomes are a Boolean decision (i.e. true: attack, false: normal). The CGP output node can be connected to the output from any previous nodes [53]. Thus, we evolved two CGP output nodes and experimented with various relational and logical functions (i.e. $>$, $<$, OR and AND). These functions showed no statistically significant difference in the performance achieved among them. However, greater than operator (i.e. $o0 > o1$) evolved programs with shorter execution times (i.e. the speed of processing). For example, Figure 5.6 shows a comparison of the processing times between these functions

from experiments conducted on phishing websites and modern DDoS datasets. Each barplot indicates the average of 20 runs with an error bar on top which shows the standard error around the average. The Standard Error (SE) of the average indicates the change in average with different experiments conducted each time (i.e. the precision of averages). It can be calculated using $(Standard\ Deviation/\sqrt{sample\ size})$. The sample size in our case refers to the number of runs.

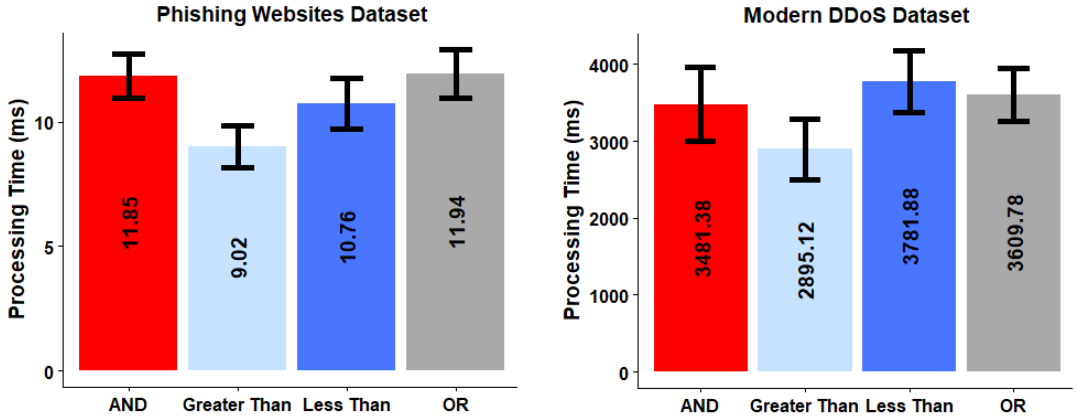

Figure 5.6: Barplot for CGP Algorithm Decision Making Functions with SE Error Bars

5.5 The Performance of Evolutionary Computation Techniques

In order to evaluate the classification performance of our detectors, we examine the detection rate, false alarm rate and classification accuracy achieved by the best-evolved program at the end of each run. We find that every time we ran the proposed algorithms to evolve programs, the generated programs are different and thus the results obtained from each program are also different. Therefore, to obtain a statistically significant measure that does not depend on the initial random seed, the results were expressed as the average and the standard error of the average for 20 independent runs. The results from the testing phase are presented in Table 5.4. A few conclusions can be drawn from these results. For instance, CGP outperformed GP and GE especially in the UNSW-NB15 and the CICIDS2017 datasets which were the most complex. Whereas, GP showed a better performance compared to GE in almost all cases.

Dataset	GP			GE			CGP		
	DR	**FAR**	**Acc**	**DR**	**FAR**	**Acc**	**DR**	**FAR**	**Acc**
Kyoto 2006+	98.78 ± 0.44	1.64 ± 0.27	98.42 ± 0.29	98.79 ± 0.13	1.88 ± 0.04	98.22 ± 0.05	**99.65 ± 0.06**	**0.70 ± 0.07**	**99.35 ± 0.04**
Phishing Website	85.63 ± 0.61	11.47 ± 0.35	88.87 ± 0.32	83.64 ± 0.00	12.65 ± 0.00	87.78 ± 0.00	**89.99 ± 0.35**	**8.31 ± 0.14**	**91.88 ± 0.13**
UNSW-NB15	90.67 ± 0.84	19.19 ± 0.80	81.80 ± 0.75	88.50 ± 1.17	24.34 ± 0.51	76.95 ± 0.45	**94.20 ± 0.73**	**13.46 ± 0.52**	**87.31 ± 0.53**
Modern DDoS	87.39 ± 0.00	7.15 ± 0.02	97.15 ± 0.04	86.76 ± 0.28	8.14 ± 0.28	95.88 ± 0.55	87.38 ± 0.00	7.13 ± 0.01	97.19 ± 0.03
CICIDS2017	88.86 ± 0.92	10.20 ± 0.77	90.36 ± 0.88	75.91 ± 1.08	21.10 ± 0.42	80.69 ± 0.68	**92.16 ± 0.64**	**6.91 ± 0.46**	**93.16 ± 0.49**

Table 5.4: The Performance of GP, GE and CGP on Testing Datasets (%). Average from 20 Runs, ± Value Indicates the SE. Bold Text Indicates a Better Results.

The proposed algorithms showed encouraging performance when tested using the Kyoto 2006+ dataset. However, they varied in their efficacy with the rest of the testbed. The DRs achieved by the GE algorithm were the lowest, especially for the CICIDS2017 dataset. Whereas the GP system recorded its lowest DR in the phishing website dataset. The highest FARs and lowest Accs were produced by the proposed methods in the UNSW-NB15 dataset experiments. In the modern DDoS dataset, the proposed methods have shown comparable performances. A possible explanation for this variation in the results may be the difference between datasets (i.e. various environments) and the nature of the threats in them. However, this is essential since we wanted to test our systems using different integration testing environments. A possible way to increase algorithm performance effectiveness is by adopting the ensemble technique which would help boost knowledge extraction from the search space. In the next chapter, EC algorithms are used as a stacking approach to evolve IDS programs.

It is important to sustain the diversity of EC algorithms in order to guarantee we have examined a large area of the search space and not limited their search to an unprofitable area. Table 5.5 presents the average number of terminal and nonterminal nodes in the best-evolved programs over the 20 runs from each dataset. Terminals refer to dataset features whereas nonterminal corresponds to functions. In this table, it can be seen that GP and CGP paradigms the diversity of their solutions compared to the

GE paradigm.

Dataset	GP		GE		CGP	
	Terminal	**Nonterminal**	**Terminal**	**Nonterminal**	**Terminal**	**Nonterminal**
Kyoto 2006+	27	56	4	6	35	58
Phishing Website	7	12	3	4	19	27
UNSW-NB15	36	48	4	6	19	25
Modern DDoS	36	58	5	7	22	32
CICIDS2017	55	67	3	4	29	32

Table 5.5: Average Number of Terminal and Nonterminal Nodes in Best-Evolved Rules Learned with Proposed Approaches

Based on the experience of numerous researchers over many years, it seems that EC methods have been particularly fruitful in areas having unknown or poorly understood interrelationships among the relevant features [48]. They have proved successful where the application is new or otherwise not well understood. Furthermore, EC methods provided novel solutions, unveil unforeseen relationships among features; and, sometimes discover new concepts that can then be employed in a wide variety of circumstances. The analysis of best programs was evolved by the proposed methods for the modern DDoS dataset. We want to illustrate how often each feature in this dataset contributed to the construction of the evolved programs. From Figure 5.7, last-pkt_reseved was the most frequently used feature for both GP and CGP. Whereas pkt_delay holds the highest count, where it appeared in 17 out of 20 programs evolved by the GE paradigm. This can help security teams to examine each feature and potentially find out the reasons for the classification errors. Thus, introducing a new variant of the features used (e.g. capture it based on different time windows) and/or aggregate multiple features information to produce more useful ones. Furthermore, the proposed methods did not employ the whole feature set available while constructing the solutions. Hence, any reduction in the solution size that is able to provide a given overall performance is necessary for the computational efficiency of the IDS. Consequently, if the activities are accurately detected using lightweight processing, it becomes feasible to the security team to carry out other actions (e.g. IP blocking) in real time [76].

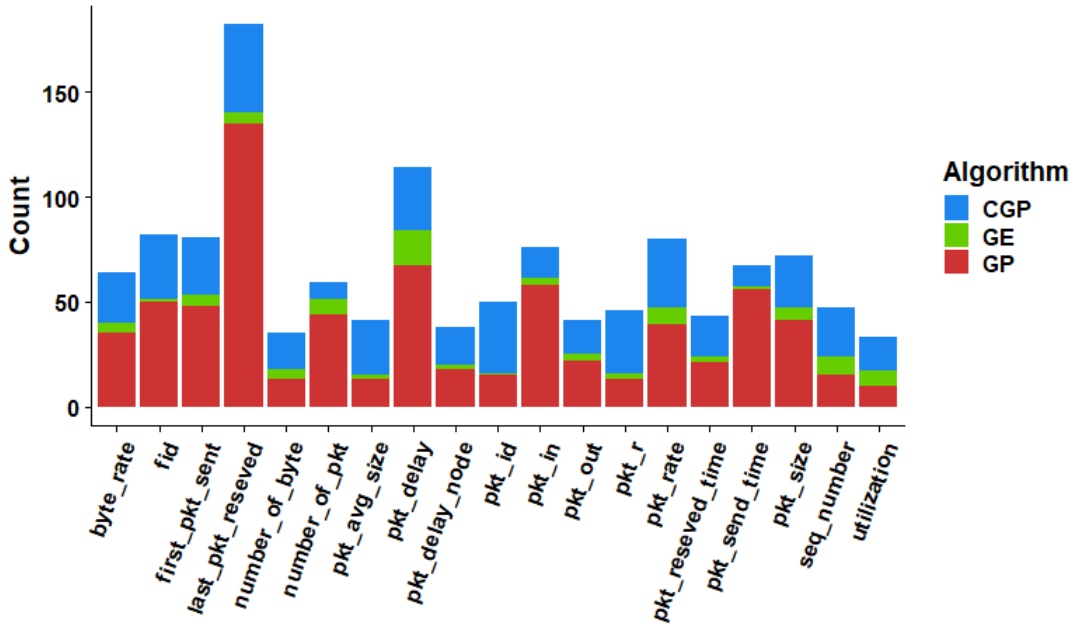

Figure 5.7: Frequent Features Selection (Modern DDoS)

5.6 Evolutionary Computation Techniques for Detecting Unknown Attacks

Not only has the volume of attacks dramatically increased nowadays, but also the landscape of these attacks has become more assorted, with attackers operating harder to find new methods of attack [7]. To test the robustness of our proposed approaches, we have ensured that attack traffic in both training and testing datasets was significantly different. This means that the attack types in the training dataset are different from the testing dataset. This will show the EC frameworks capability to learn features of known attacks which may hold the symptoms of unknown attacks. In this implementation, the detection of previously unseen variations of known attacks (i.e. subfamilies) and unseen attacks are both tested. The same GP, GE and CGP frameworks settings from the previous investigation were adopted here. Twenty independent runs were performed and the number of instances of unknown attacks that were detected is reported. The detection of attacks existing only in the testing stage and absent from the training stage indicates the potential capability to detect novel attacks.

To evaluate the capability of the proposed methods for detecting previously unseen variations of known attacks, we use the training and testing portions of the modern DDoS dataset. This dataset includes 4 subfamilies of DDoS attack. Table 5.6 presents the new distribution of this dataset where only network layer DDoS attacks appear in

the training (i.e. Smurf and UDP Flood). Whereas application layer DDoS attacks (i.e. SIDDOS and HTTP Flood) exist in the testing portion only. The network layer attack instances were removed from the training portion and added to the testing portion.

Attack Name	**Attack Traffic in Training Set**	**Attack Traffic in Testing Set**
Smurf	8,341	4,249
UDP Flood	132,441	68,903
SIDDOS	**0**	6,665
HTTP Flood	**0**	4,110
Total Records	140,782	83,927

Table 5.6: The New Attack Distribution of The Modern DDoS Dataset

Table 5.7 displays the overall achievements of the best-evolved programs and the algorithms which provide similar performance results.

Approach	**Modern DDoS**		
	DR	**FAR**	**Acc**
GP	88.15	6.60	97.44
GE	88.10	6.44	97.76
CGP	88.16	6.65	97.35

Table 5.7: Overall Performance of Best-Evolved Program (%)

The number of attacks detected in the testing dataset is presented in Table 5.8. Although the Smurf attack was included in the training dataset, it can be noted that it is the most challenging attack for all classifiers. This is due to the nature of the attack which is very similar to normal traffic [40]. However, other algorithms such as MLP, Random Forest and Naïve Bayes employed in [40] detected 1396, 1414 and 140 instances only from the Smurf attack, respectively. The proof of concept implementation shows the proposed approaches can detect application layer attacks especially the HTTP Flood, which were absent from the training dataset.

Table 5.9 shows the outputs from the best-evolved programs using the proposed methods in this experiment. An instant observation is that the size (i.e. number of nodes) of the solutions utilising GE is smaller and easier to interpret in comparison to the ones evolved by GP and CGP. This can be justified by the fact that GE evolution operators, which randomly select and change various portions of the solutions and generating solutions complying to the outlined grammar, are much more difficult than the operators in GP and CGP. An advantage of the proposed algorithms is the representation of the evolved solution. Examining the evolved output can provide useful insights into how

the algorithm learns to solve a given problem. From Table 5.9, it is apparent that the GP algorithm selected 3 out of 19 features to construct its best individual. However, it shows that the GP needed control over the growth of the program (i.e. number of nodes) as that affects the understandability of the program. This could be achieved through the adaptation of the multi-objective GP concept [79]. In contrast, the GE algorithm utilised 4 features. Finally, CGP used 8 features to build its graph and the graph shows the implicit reuse of the node (Ln[pkt_rate]/pkt_rate).

Attack Name	**GP**		**GE**		**CGP**	
	No. of Attack Traffic Detected	**DR (%)**	**No. of Attack Traffic Detected**	**DR (%)**	**No. of Attack Traffic Detected**	**DR (%)**
Smurf	1,552	36.52	1,539	36.22	1,552	36.52
UDP Flood	62,015	90	61,981	89.95	62,023	90.01
SIDDOS	6,310	94.67	6,315	94.74	6,313	94.71
HTTP Flood	4,110	100	4,110	100	4,110	100
Total Records	73,987	88.15	73,945	88.10	73,998	88.16

Table 5.8: Detection Results by Proposed Approaches (Testing Dataset)

Algorithm	Best-evolved Program
GP	(Sqrt[Tan[Tan[Log[last_pkt_reseved]]]] >= (Sin[Floor[Min[Floor[Sqrt[seq_number + Tan[Log[last_pkt_reseved]]]], last_pkt_reseved]] / Log[(seq_number + Tan[Sqrt[Min[Sqrt[seq_number + Tan[Log[last_pkt_reseved]]], last_pkt_reseved]]]) + Floor[Min[Sqrt[seq_number + Tan[Log[last_pkt_reseved]]], Sqrt[seq_number + Tan[Log[Ln[(seq_number + pkt_size)]]]]] / Ln[(seq_number + Tan[Log[last_pkt_reseved]]) + pkt_size]]])
GE	pkt_delay > ((last_pkt_reseved + seq_number) / byte_rate)
CGP	((fid % (Cos[Tan[utilization]] + (fid == Exp[seq_number]))) < (Ln[pkt_rate]/pkt_rate)) > (((Ln[pkt_rate]/pkt_rate) * Log[Abs[pkt_in]]) >= (Ln[(Tan[last_pkt_reseved]) && (Sin[seq_number] <= Abs[Log[pkt_id/fid]])]/(Ln[Ceiling[Cos[first_pkt_sent] && first_pkt_sent]])))

Table 5.9: Intrusion Detection Programs- Best Run

The UNSW-NB15 dataset was adopted for testing the capability of the proposed methods at recognising an unseen attack (i.e. the attack is missing while training the algorithm). As presented in Table 4.2, there were 9 types of attack in this dataset. We investigated each attack separately by removing its training instances and adding them

to the testing portion. For each investigation, we ran each proposed algorithm 20 times, which brings the total of the independent runs for all attacks to (20 * 9 = 180). Figure 5.8 compares the overall performance from each algorithm. Each barplot indicates the average of 180 runs and the error bar on top refers to the SE around the average. Overall, we can see from the presented results that the CGP approach managed to achieve better performance than the other techniques and was followed by GP.

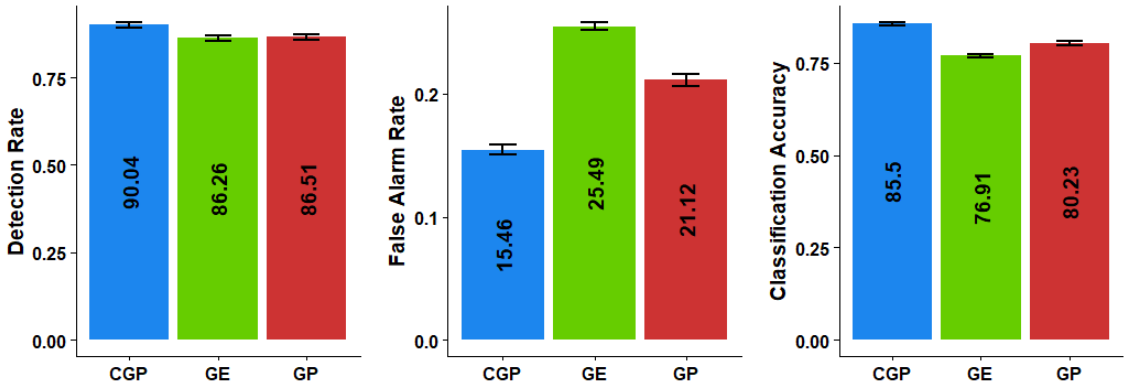

Figure 5.8: Proposed Approaches Performance

Figure 6.5 shows the distributions of the responses and the average achieved by the best-evolved program from each algorithm for each tested unknown attack (i.e. missing during the training). Each boxplot represents the investigation of an attack. The CGP average scores were better than GP and GE for 4 attacks namely: Backdoor, DoS, Exploits and Worms. On the other hand, GE achieved higher average scores than the other algorithms for 4 attacks namely: Fuzzers, Generic, Reconnaissance and Shellcode. In contrast, GP provided a better average score for the Analysis attack, and better average scores than GE for DoS and Exploits attacks. Regarding the distributions of the responses, CGP outcomes are less spread out than GP and GE in most cases. Ultimately, these variations in the behaviour towards attacks make it possible to combine these signatures and store them together in an IDS database.

The unknown attacks detection percentages are summarised in Table 5.10, they were achieved by the best individual from each unknown attack experiment. It can be seen that the proposed algorithms were able to recognise unseen attacks. However, the DR of the Fuzzers attack was low compared to others. And also the DR of the Exploits attack was low with the GE algorithm. This is due to lower variances among some malicious records and normal records in the UNSW-NB15 dataset [39]. However, the proposed techniques have achieved a higher DRs percentage for the attacks in the UNSW-NB15 dataset than [39] but with a lower classification accuracy percentage than their system

accuracy (92.8%).

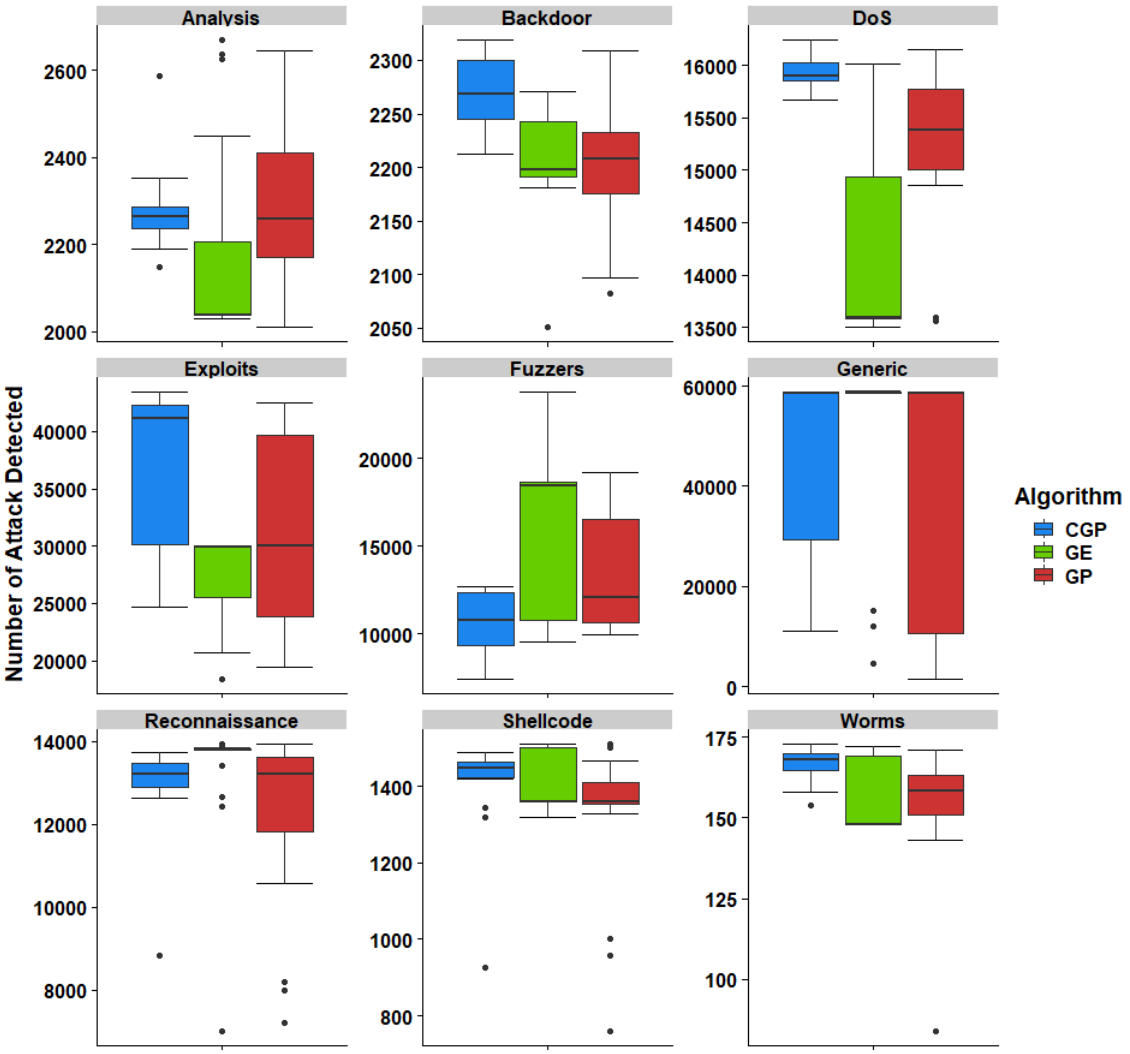

Figure 5.9: Boxplots Illustrating The Distribution of Each Attack That was Detected by Each Algorithm (Each Boxplot Represents an Experiment of 20 Independent Runs)

In this investigation, both unknown attacks experiments showed that the proposed algorithms do not require prior information about an attack in the training to be able to classify it. However, there is still a need to have attack behaviours present in the training phase which might mirror the symptoms of unknown attacks.

Attack Name	GP		GE		CGP	
	No. of Attack traffic Detected	DR (%)	No. of Attack traffic Detected	DR (%)	No. of Attack traffic Detected	DR (%)
Analysis	2,253	84.16	2,147	80.20	2,229	83.29
Backdoor	2,309	99.14	2,226	95.57	2,260	97.03
DoS	15,872	97.05	15,623	95.53	16,050	98.14
Exploits	42,544	95.55	29,873	67.09	42,891	96.33
Fuzzers	12,957	53.43	12,476	51.45	8,482	34.98
Generic	58,797	99.87	58,593	99.52	58,836	99.94
Reconnaissance	13,161	94.09	12,658	90.49	13,689	97.86
Shellcode	1,387	91.79	1,319	87.29	1,488	98.47
Worms	169	97.12	149	85.63	168	96.55

Table 5.10: Comparison of DR on Attack Categories of UNSW-NB15 Dataset

5.7 Conclusions

In this chapter, it was revealed that evolutionary computation techniques uncover the complex relationships between various tested environments. The first hypothesis of this research is supported by the evaluation outcomes. The proposed approaches are capable of mapping the input vector space (i.e. features) into a decision space to discriminate between classes (i.e. normal vs. anomaly). One interesting feature of these approaches is the ability to evolve a detector (i.e. behavioural signatures) that can detect a wide range of attacks. In addition, the experimental results demonstrated that our proposed approaches showed a robust performance in detecting unknown attacks that the systems were not trained on. These results suggest that the proposed methods recognised new attack methods well based on their similarity to known attacks. GP, GE and CGP emerged as reliable predictors for finding previously unseen attacks/subfamilies of known attacks. In general, CGP was the best performing paradigm compared to GP and GE.

Chapter 6 | Evolving Ensemble Model using Evolutionary Computation for Intrusion Detection

In the previous chapters, the standard EC algorithms for evolving IDS rules were discussed. In this chapter, we propose an EC classifier design scheme that incorporates multiple learners to provide superior predictive performance over a single algorithm. The proposed techniques (GP, GE and CGP) combine multiple learning algorithms into a single and powerful prediction function. The proposed techniques will work as a secondary learning process called meta-learner. The evolved programs are demonstrated and explained. The results are reported and the performance of evolved programs are compared. Lastly, adopting ensemble paradigms to evolve intrusion detection programs for detecting unknown attacks are described.

6.1 Stacking

Stacking is the informal term used to reference Stacked Generalization [116]. The main idea behind stacking is finding the optimal combination from a collection of (heterogeneous or homogeneous) learning algorithms. Stacking is a type of ensemble that comprises of training a second level, known as a meta-learner, to find the optimal combination of the first level (i.e. base learners). Each base learner employs diverse kinds of knowledge representation and learning biases, thus the input space will be investigated differently and a variety of learners will be acquired [117]. Unlike other types of ensemble, the goal in stacking is to combine strong, diverse sets of learners [118]. In general, stacking is often used to produce heterogeneous ensembles. Obtaining the right combination of base learners is one of the issues that stacking implementation faces [117]. In this work, a heterogeneous stacking scheme is employed for constructing IDS programs which hopefully will outperform the single classifiers. There are several issues that emerge when utilising an ensemble of the stacking type, for instance, the algorithms that are utilised to create the base learners, their parameters settings and their

number [63]. These issues have been addressed through the adoption of the automatic machine learning concept and EC algorithms described in the subsequent sections.

Figure 6.1 illustrates the steps normally used in order to construct the stacked model. Firstly, we generate the first level classifiers by training K learning algorithms using the training dataset. Then, the K base classifiers that are used to create the meta-date form both training and testing datasets (Figure 6.1a). The meta-data have K attributes driven by the response of the K classifiers to each instance (i.e. class probability or predicted class) associated with the actual class. Once meta-data have been built, a second level algorithm (i.e. meta-learner) is trained on collating the meta-data to build the meta-classifier (Figure 6.1b). The main purpose here is to find the best combination of the base classifiers. Finally, the meta-classifier is then used to predict the class of the new instances in the testing meta-data (Figure 6.1c).

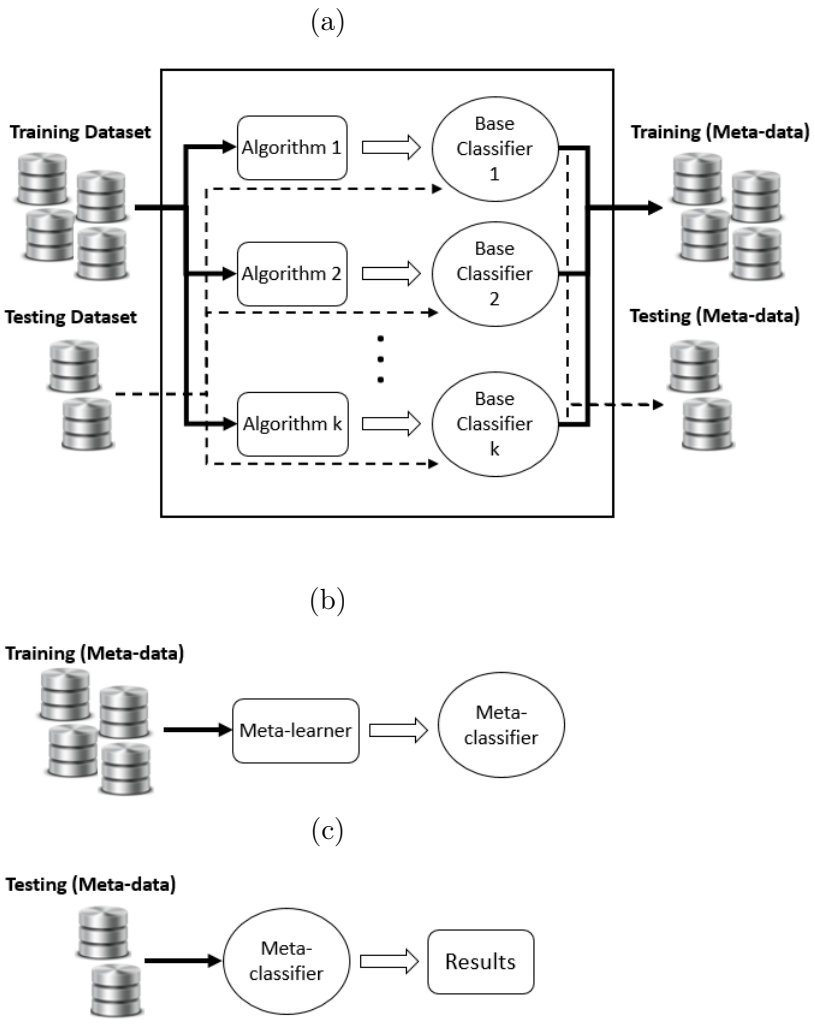

Figure 6.1: Generating an Ensemble of Classifiers using Stacking [63]

6.2 Generating Base Learners

The algorithms used to generate the pool of classifiers for this experiment are implemented in the H2O.ai platform [118]. H2O.ai includes many common machine-learning algorithms and implements the best in class algorithms at scale. The Automatic Machine Learning (*AutoML*) function in the platform is used to build base learning models. This function automates the supervised machine learning model training process. AutoML performs basic data pre-processing if required, such as imputation, one-hot encoding and standardisation. During the model generation stage, it implements random grid search and model parameters tuning using the validation set to produce the best performing models. AutoML trains and cross-validates a distributed random forest, an extremely randomised forest, a random grid of gradient boosting machines, a fixed grid of generalised linear model, and a random grid of deep neural nets. In AutoML, the max models parameter is set to 10 (i.e. number of models generated) and stopping rounds (i.e. early stopping according to convergence of stopping metric) was set to 6. Stopping rounds were utilised to stop training the model when there is no improvement in the performance. The rest of the parameters were determined automatically by the platform. In practice, the default cross-validation folds in AutoML is set to 5 while training the base learners. Note that it is possible to produce more than one model from the same family. However, these models vary in their parameters settings. The implementation was in R using the following command (where x assigned feature vector and y assigned actual outcome):

```
R> aml <- h2o.automl(x = x, y = y,
                     training_frame = train,
                     validation_frame = valid,
                     max_models = 10,
                     stopping_rounds = 6,
                     seed=1)
```

The following supervised learning algorithms (i.e. base learners) in AutoML were considered for generating a pool of individuals as an input vector space for the proposed algorithms [118] [119][120]:

6.2.1 Distributed Random Forest (DRF)

This is a tree based technique and it is considered to be a powerful paradigm for classification and regression problems. When implemented over a dataset, DRF generates multiple decision trees. It takes the average of the classification trees, each created on different random samples from the dataset. Each of these trees is a weak learner

built on a subset of rows and columns. Increasing the number of trees will decrease the variance. The average prediction over all of the constructed trees will generate the end outcome, whether it is predicting a class type or a probability. Trees in the random forest model are independent. It is easy to use, non-linear, and provides feedback on the importance of each predictor in the model, making it one of the most robust algorithms for noisy data. DRF it is an ensemble method of type bagging.

6.2.2 Extremely Randomized Tree (XRT)

This is a tree based ensemble model similar to the random forest. Like random forests, an arbitrary subset of candidate features is utilized. However, the randomness in XRT goes one step further in terms of how splits are performed. It consists of randomly chosen candidate features and a threshold is picked to determine when to split a tree node. Furthermore, XRT does not implement the bagging technique to create a set of training patterns for each tree. The whole learning sample is utilised to train all the trees. This model usually leads to more diversified trees.

6.2.3 Gradient Boosting Machines (GBMs)

This is a decision tree based model where a model in the shape of an ensemble of weakly predicted trees is created. GBM builds consecutive classification trees on all the features of the dataset in a completely distributed manner. These trees are dependent on one another. Thereafter, it fits those consecutive trees as one model. The ensemble of type boosting is the central idea here. GBM gives a higher weight to the instances that are difficult to classify with a tree (i.e. model), so the subsequent tree pays more attention to these instances. Finally, using this technique helps to decrease bias.

6.2.4 Generalized Linear Model (GLM)

GLM is a statistical tool that generates a linear model over the inputs in a dataset to predict the response variable. H2O.ai fits this model according to the maximum likelihood estimation using repeatedly reweighted least squares. The flexibility of the model structure unifies the typical regression methods (such as linear regression and logistic regression for binary classification). GLM generates linear regression by permitting the linear model to become linked to the outcome variable by using a link function and by permitting the magnitude of the variance of each measurement to be a function of its expected value.

6.2.5 Deep Neural Nets (DNNs)

"H2O.ai DNN is based on a multi-layer feedforward artificial neural network that is trained with stochastic gradient descent using back-propagation" [118]. A neuron is a function that takes multiple numeric inputs and gives out one numeric output. The network can incorporate a large number of hidden layers composed of neurons besides activation functions (i.e. tanh, rectifier and maxout). Each compute node trains a copy of the global model parameters on its local data and supplies regularly to the global model via model averaging across the network. The layers amongst the input layer and the output layer are known as the hidden layers. Each neuron in each hidden layer has a weight for each of its inputs, and modifying those weights is how the network learns.

6.3 Evolutionary Computation as a Meta-learner

In this technique, various algorithms are used on the same dataset and feature set in order to build different learner models. After a number of models are built using part of the dataset i.e. training, the predictions of these models are combined and a common decision is taken. In phase two, GP, GE and CGP are used to create the actual stacked model. When using each technique the ensembles are coded as genetic programs, each individual representing a possible aggregation of the available base learners. The main difference is the representation of individuals. Learning the stacking procedure via proposed methods leads to possibly non-linear combination mechanisms that could better exploit the outputs of multiple learners. This is unlike a stacking scheme led by weighting or majority voting strategies that linearly combine multiple learners, which is not the best ensemble building option [121]. During the combination process (i.e. stacking), we considered the continuous outputs (i.e. probability) as meta-data, rather than the discrete outputs (i.e. label) of the base classifiers. The main reason for that is the type of operations performed by our proposed algorithms on this dataset. There are two types of actions that proposed algorithms executed while creating the stacked model: selecting ensemble members from a pool of base models and evolving the combination among selected base models (i.e. base models fusion). This also leads to automatic determination of the size of the ensemble instead of predefined size (i.e. fixed).

The stacking approach has proven successful across a large number and types of datasets; however, it is considered to be time-consuming due to the fact that it needs to train a second level learner (i.e. meta-learning) [62]. To avoid a further phase of training, we split the training portion into 80% training, used to train base models, and 20% validation, used by the proposed techniques for building ensemble models (see Figure

6.2). Thus, recomputed the stacked model require with a moderate computational effort in the event of any changes in the data.

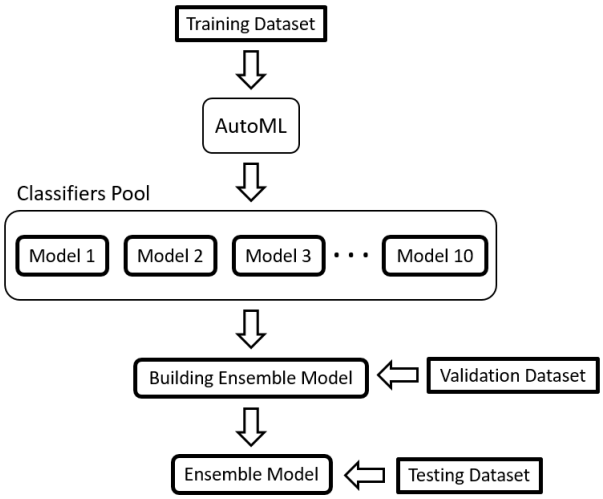

Figure 6.2: General Scheme used for Building our Ensemble Model

The proposed paradigms offer an intelligent self-configuration environment to perform models selection and integration for constructing the ensemble. This helped them to perform a dynamic method instead of a static one for selecting stacked solution members. In addition, the challenges related to determining the size of the ensemble have been addressed. Each ensemble is either represented as a tree for GP, a grammar for GE and as a graph while running CGP. In this implementation, the input vector elements are in the form of prediction probabilities generated by stacked members. Equation 5.1 is used as a fitness function for the stacking solutions. The GP framework from the previous implementation in Table 5.1 and its parameters settings were adopted here as well. The grammar introduced in Table 5.2 and the standard GE parameters settings were extended to evolve the ensemble model. Finally, standard CGP parameters settings (see Table 5.3) are utilised here with the exception that the number of generations was modified to 2,000. This is due to the stagnation of the fitness value after reaching 2,000 generation for all runs.

6.4 The Performance of Evolutionary Computation Ensemble Techniques

The average values from 20 runs and the SE around the average are depicted in Table 6.1. It can be observed from the results that the stacking approach performed better than standard implementations in almost all experiments. The purpose of the

stacking learning phase was to improve the effectiveness of the security performance of IDS evolved programs. GE stacking demonstrated a superior performance for all datasets compared to GE standard implementation. On the other hand, both GP and CGP stacking programs improved achievement compared to standard programs in most cases. However, both methods displayed slightly declined efficacy on the modern DDoS dataset in terms of false alarms and accuracy. The results showed that the aggregated knowledge of used ML techniques was effective, leading to better results compared with the single base classifier. The proposed stacking techniques were almost identical in their performance with marginal differences. The main reason for that is the new search space (i.e. meta-data), which is generated by the base models, provides complementary information. Thus, the proposed methods search through similar decisions boundaries reached by the base models.

Dataset	**GP Stacking**			**GE Stacking**			**CGP Stacking**		
	DR	**FAR**	**Acc**	**DR**	**FAR**	**Acc**	**DR**	**FAR**	**Acc**
Kyoto 2006+	99.94 ± 0.004	0.14 ± 0.001	99.87 ± 0.001	99.93 ± 0.002	0.14 ± 0.001	99.87 ± 0.001	99.94 ± 0.002	0.14 ± 0.001	99.87 ± 0.001
Phishing Website	97.52 ± 0.084	3.25 ± 0.034	96.65 ± 0.044	97.63 ± 0.085	3.28 ± 0.033	96.60 ± 0.042	97.33 ± 0.076	3.22 ± 0.045	96.70 ± 0.051
UNSW-NB15	96.28 ± 0.07	11.30 ± 0.049	89.46 ± 0.044	96.19 ± 0.097	11.05 ± 0.103	89.67 ± 0.083	96.29 ± 0.048	11.29 ± 0.048	89.47 ± 0.041
Modern DDoS	87.44 ± 0.007	7.29 ± 0.025	96.86 ± 0.051	87.37 ± 0.013	7.04 ± 0.05	97.37 ± 0.1	87.40 ± 0.004	7.18 ± 0.017	97.10 ± 0.033
CICIDS2017	99.91 ± 0	0.1 ± 0	99.87 ± 0	99.91 ± 0.001	0.11 ± 0	99.87 ± 0	99.92 ± 0.001	0.10 ± 0	99.87 ± 0

Table 6.1: The Performance of EC Algorithms as a Meta-classifier (%)

In this testbed, no differences in the performance were observed between standard, stacking and the prior study (see Table 2.1) experimented on the modern DDoS dataset. A possible explanation for this might be that there is an underrepresentation of some of the behavioural types and/or the extracted feature set does not accurately describe the search space. In future investigations, it might be possible to evolve a specialist IDS program (i.e. distinguish one class of instances from the other, instead of normal vs. attacks). In addition, it could obtain a more meaningful feature set that is able to represent the types of traffic effectively.

The type of programs that can learn with the proposed approaches are described below. Figure 6.3 provides the average frequency (Kyoto 2006+ dataset, 20 runs) of base models selection during ensemble construction (i.e. the best individual from each run). From this figure, it can be seen that GMB_5 was the most frequently used model; however, the proposed approaches vary in their selection. Looking at this figure, the CGP average frequency for selecting each base model was the highest, followed by GP. The least chosen model for GE was GLM whereas DNN was the least picked one by GP and CGP. From Figure 6.3 it can be seen that there is not a conclusive trend regarding utilisation of base models. Furthermore, some of the best individuals (i.e. stacking programs) output evolved by the proposed methods were relatively small but with high performance, for example:

$$\text{GP} \;:\; \text{GBM\_0} \;\hat{}\; \text{GBM\_3} <= \text{Cos}\,[\text{GBM\_3} + 2 * \text{GBM\_5}]$$

$$\text{GE} \;:\; \text{Cos}\,[\,\text{Exp}\,[\text{GBM\_4}]\,] > \text{GBM\_3}$$

$$\text{CGP} \;:\; \text{Abs}\,[\,\text{Log}\,[\,\text{Max}\,[\text{DRF},\ \text{GBM\_2}]\,]\,] > \text{Log}\,[\,\text{Tan}\,[\,(\,\text{Log}\,[\text{GLM}]\,) <= \text{GBM\_3}]\,]$$

This variety of results reveals the adaptive properties that the proposed methods possess for producing ad hoc stacking programs.

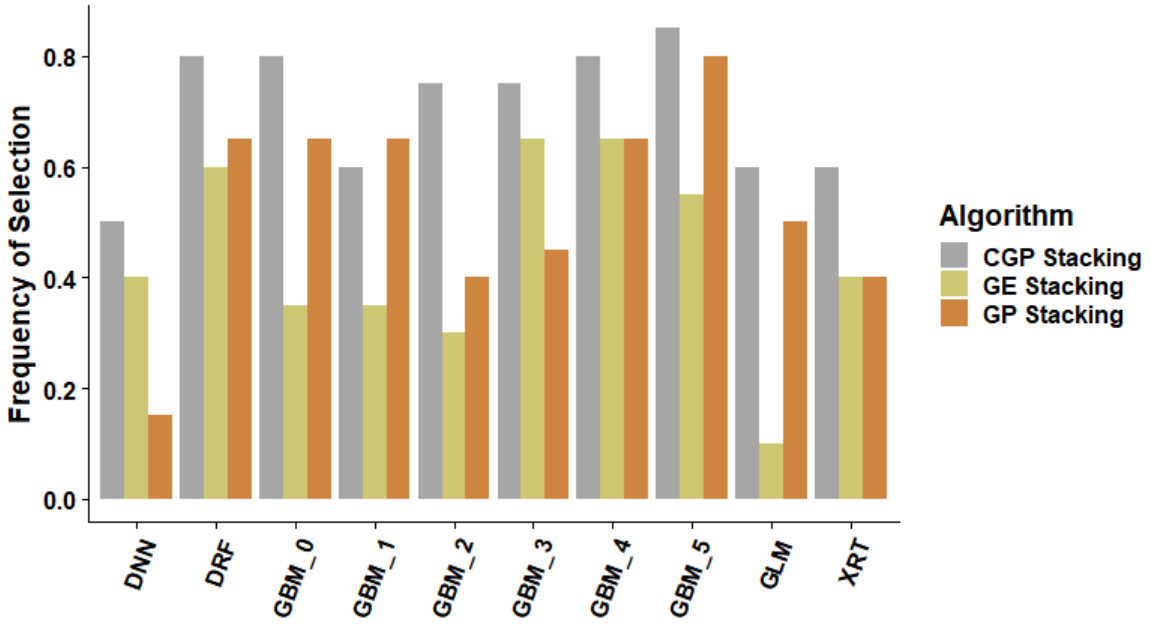

Figure 6.3: Average Frequency of Base Models Selection in Stacked Programs (Kyoto 2006+ dataset, 20 runs)

6.5 Evolutionary Computation Ensemble for Detecting Unknown Attacks

Unknown attacks are emerging as one of the most serious threats to a system [13]. An experimental assessment to evaluate the suitability of the proposed stacking mechanisms to investigate detecting unseen attacks was performed. The same settings of the proposed algorithms previously used were adopted. Table 6.2 compares the performance of the best-evolved stacking program out of 20 runs evaluated using the modern DDoS dataset given in Table 5.6 with each algorithm. The aim is to recognise unseen variations of known attacks. In comparison with the best-evolved standard program implementation, the techniques performances were similar. The results in Table 6.3 demonstrate that the proposed stacking techniques can detect the majority records of application layer DDoS attacks (i.e. unseen variations of known attacks). However, like standard implementations, the Smurf attack detection percentage remained very low.

Approach	**Modern DDoS Table (5.6)**		
	DR	**FAR**	**Acc**
GP Stacking	88.22	6.80	97.04
GE Stacking	88.17	6.62	97.40
CGP Stacking	88.19	6.74	97.16

Table 6.2: Overall Performance of The Best Stacked Programs (%)

Attack Name	**GP Stacking**		**GE Stacking**		**CGP Stacking**	
	No. of Attack Traffic Detected	**DR (%)**	**No. of Attack Traffic Detected**	**DR (%)**	**No. of Attack Traffic Detected**	**DR (%)**
Smurf	1,551	36.50	1,540	36.24	1,554	36.57
UDP Flood	62,062	90.07	62,031	90.02	62,041	90.04
SIDDOS	6,323	94.86	6,319	94.80	6,320	94.82
HTTP Flood	4,110	100	4,110	100	4,105	99.87
Total Records	74,046	88.22	74,000	88.17	74,020	88.19

Table 6.3: Detection Results by Stacking Approaches

The best-evolved program outputs are shown in Table 6.4. As can be seen from the table, the GP stacking individual suffers from redundancy in the code. Again, this could be overcome through the utilisation of the multi-objective GP implementation. Although, the best individual from GE stacking did not suffer from redundancy. However, we did notice best individuals from other runs with this issue. In the CGP stacked program, the node (GBM_4 * Sin[GBM_5]) was implicitly reused 5 times whereas

nodes such (Exp[GBM_5]) and (Ceiling[XRT]) were reused 3 and 2 times respectively. This is one of the advantages of using a graph representation. In addition, there were various base models selected in the stacked model generation where GP used 4 models. Whereas GE and CGP used 6 and 7 models, respectively. GBM models and the DNN model were picked by all proposed algorithms. This shows how the proposed methods automatically determine the members of the ensemble model. Overall, 9 out of 10 base models were used.

Algorithm	**Best Meta-classifier**
GP Stacking	(GBM_4 < Max[GBM_3 / Floor[Tanh[Max[Exp[Sin[Max[Sqrt[GBM_5] / GBM_3, Tanh[Exp[DNN]]] / GBM_3]/GBM_3 ^ 2, Tanh[Max[Sqrt[GBM_5] / GBM_3, Tanh[Exp[GBM_3]], Tanh[Exp[Sin[Sin[Abs[DNN]]]]]/GBM_3]]]] / GBM_3] / Max[Sqrt[GBM_5] / GBM_3, DNN], Tanh[Exp[Sin[Abs[DNN]]]]]) || (DNN <= Tanh[Max[Exp[-Sin[Sin[Exp[DNN]]] Sqrt[GBM_5], GBM_5]] / GBM_3])Tanh[Max[Exp[-Sin[Sin[Exp[DNN]]] Sqrt[GBM_5], GBM_5]] / GBM_3])
GE Stacking	(GBM_3 < Ln[GBM_3]) || (-((-1 + GBM_4^2)/((1 + GBM_4 ^ 2) (-1 + GLM + 2 / (1 + DNN ^ 2) + GBM_0))) >= DRF)
CGP Stacking	Tan[Cot[GBM_4 * Sin[GBM_5] > Ceiling[XRT]] / ((Tan[Sqrt[Exp[GBM_5] ^ (GBM_3/GLM)]] <= Abs[DNN / Min[GBM_4, GBM_4 * Sin[GBM_5]]]) >= (GBM_1 <= (GBM_5 <= XRT)))] > (Floor[Abs[Max[Max[DNN, GBM_4 * Sin[GBM_5]] ^ Exp[GBM_5] - Cos[GBM_3 / GLM] + Min[GBM_4, GBM_4 * Sin[GBM_5]]]] ^ (GBM_3 / 2) * ((Max[DNN, GBM_4 * Sin[GBM_5]] ^ Exp[GBM_5] != Ceiling[XRT])/(GBM_4 + XRT)) ^ GBM_3] * Log[GBM_1])

Table 6.4: Intrusion Detection Programs (Meta-classifier) - Best Run

To examine our stacking frameworks capability in detecting unseen attacks, we used the UNSW-NB15 dataset for this purpose. There are 9 categories of attack in this dataset and we tested our stacking implementations capability for each attack separately. We executed each framework 20 times for each attack which brings the total of runs in this experiment to 180. The barplots, in Figure 6.4, represent the averages of DR, FAR and Acc measurements for 180 runs. The error bars on top show the SE around the averages. This figure depicts two investigations, standard and ensemble. Overall, the ensemble implementations showed a better performance, however, the standard CGP has achieved relatively close performances to the ensemble ones.

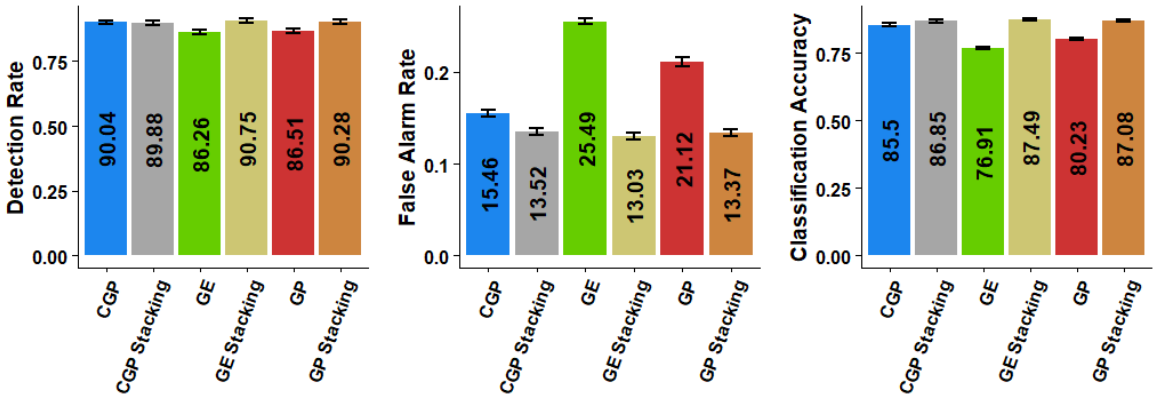

Figure 6.4: Proposed Approaches Performance

For each attack class, the boxplot corresponding to each method (standard and stacking) of 20 runs is plotted in Figure 6.5. These boxplots show the spread of responses (i.e. the number of times the unknown attack was detected) alongside the average. Compared with Chapter 5 standard implementations, stacking approaches provided better performance for 6 attacks out of 9 namely: Backdoor, DoS, Exploits, Reconnaissance, Shellcode and Worms. In addition, the stacking boxplots appear to have smaller variability than standard ones, which indicates a constant in their performance. Although, in this investigation, the overall performance of stacking techniques were better than standard techniques. However, standard techniques produced preferable outcomes for 3 attacks namely: Analysis, Fuzzers and Generic. The Fuzzers attack was noticeably worse in the stacking implementations. Looking at the DRs of the base models for these attacks may indicate the lack of satisfactory results. The DRs vary from 80.05% to 92.04% for the Analysis attack and from 21.29% to 67.16% for the Fuzzers attack and from 56.26% to 99.94% for the Generic attacks. The detection range of the Fuzzers attack is the lowest which may contribute to the decline in the capability of the meta-classifiers.

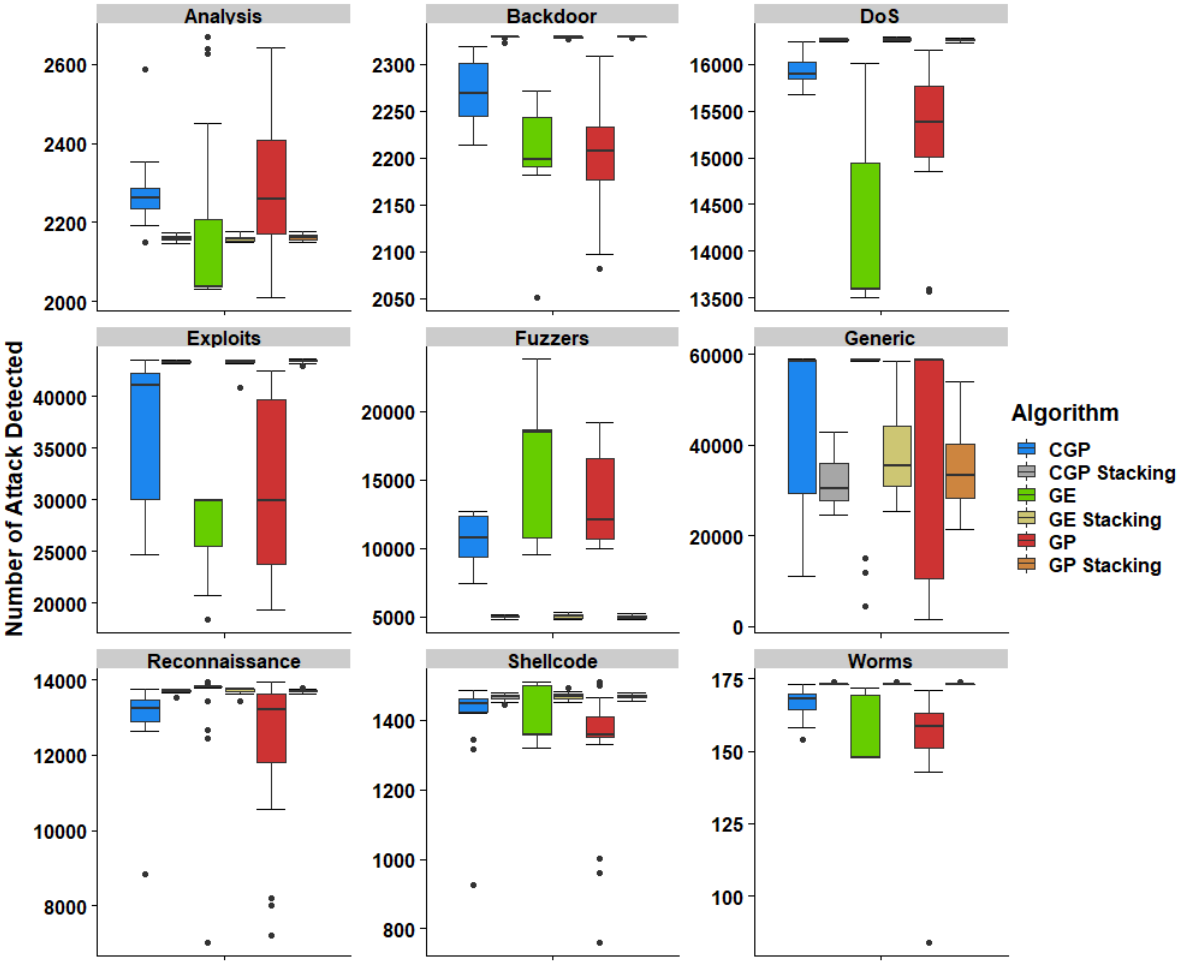

Figure 6.5: Boxplots Illustrating The Distribution of Each Attack That was Detected by Each Algorithm (Each Boxplot Represent an Experiments of 20 Independent Runs)

6.6 Conclusions

In this chapter, we extend our GP, GE and CGP frameworks to learn ensemble solutions for solving the problem of detecting cyber threats. The proposed approaches are beneficial since they provided security teams with better-performing IDS programs. The ensemble construction through the proposed techniques allows for more complex stacked solutions spaces to arise. Empirical results on benchmark datasets were reported. The assessment exposed the effectiveness of the proposed approaches. In general, Stacked solutions outperformed the standard models. However, a clear benefit of stacked approaches applied to the modern DDoS dataset could not be identified in this analysis, to the best of our knowledge this is due to underrepresentation of some classes.

An analysis of the evolved solutions showed how each proposed method forms the

meta-classifier from the base models. We also evolved stacked models for detecting unknown attacks. However, comparing standard and ensemble methods showed that they achieved differently on unseen attacks, but with lower false alarms in favour of ensemble methods. Finally, there is no significant difference observed in the performance between the stacked approaches for both investigations (known-known and known-unknown).

In the next chapter, we intend to show how the multi-objective EC algorithm could be applied to synthesise intrusion detection programs that generate potentially the best (or near best) trade-offs among security and non-security factors.

Chapter 7 | Trade-offs in Intrusion Detection

In the previous two chapters we have detailed experiments that focus on the effectiveness (i.e. security performance) of evolved IDS programs. This chapter provides a new means to develop intrusion detection system for resource constrained environments. The programs evolved using GP are analysed, and it was found that a variety of trade-offs can be produced among the classification accuracy and other criteria of these programs. To be able to identify these trade-offs, multi-objective evolutionary computation described in Section 3.1.4 is utilised. The experimental outcomes are presented and discussed. Finally, conclusions are outlined.

7.1 Multi-objective Evolutionary Genetic Programming for Learning

In this research, evolutionary algorithms, which enable us to enhance several objectives at the same time, are employed to find programs that are both effective (i.e. perform well against the standard detection and alarm rates) and efficient (i.e. are fast and generally consume little resource). This is a dual-objective problem and we use GP combined with a suitable MOEC method. MOEC enables system designers to discover a set of solutions instead of a single solution. In general, standard EC algorithms are able to simultaneously find a optimal solution for all objectives. In order to do that a classical weighted sum method is used to combine individual fitness functions into a single objective. However, assigning a proper weight vector also counts on the scaling of each individual fitness function which is often a difficult task. Consequently, the solution acquired by using this method mostly counts on the predefined weight vector. In addition, there is no guarantee in the case of a single objective (aggregated objectives) that the solution which has the best performance on the second objective will be selected, which would not be the case in a multiple objectives search. On the other hand, combining MOEC with GP avoids the emergence of bloat [44], [48], [122] and reduces overfitting effectively [123]. This also will present opportunities to improve the comprehensibility of the output and encourage generalisation.

An example of the multiple objectives optimisation technique is given here performed on the phishing websites dataset. We evolved programs using SPEA2, an extension to ECJ combined with GP to explore trade-offs among two conflicting security objectives in an intrusion detection system: detection rate described in equation (2.1) and false alarm rate calculated using equation (2.5). The fitness of an individual (evolved program) is depicted by the two objectives. A set of optimal solutions is acquired by utilising MOGP method. Figure 7.1 displays the Pareto fronts, which show the optimal evolved programs at the end of 50 generations of 10 independent runs. The chart represents detection rate (maximised) versus false alarm rate (minimised) when their metrics are optimised concurrently. As we can see, there is an obvious trade-off amongst these two objectives: while the detection rate increases, false alarms increase too.

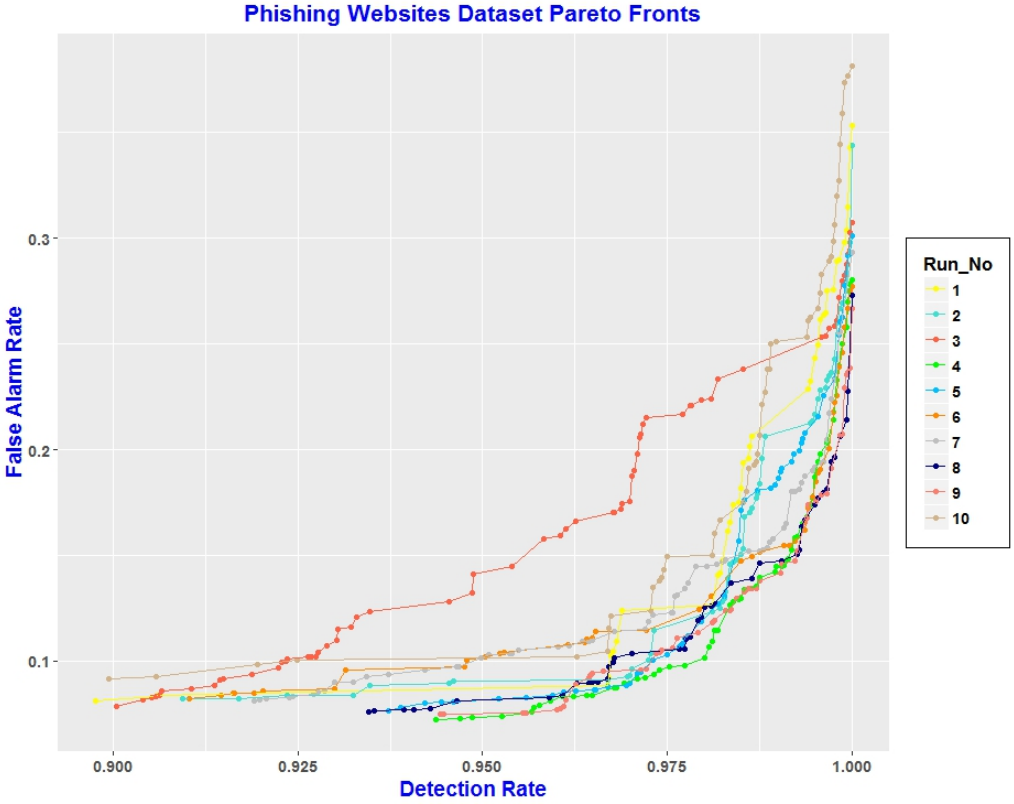

Figure 7.1: Trade-offs Between DR vs. FAR (Phishing Websites)

Each run generates a set of non-dominated solutions (i.e. Pareto front). We can see from the above figure that a solution in one Pareto front may dominate or be dominated by a solution from another Pareto front. That is why we decided to unite these fronts and produce a global Pareto front that contains all non-dominated solutions across these 10 Pareto fronts. To achieve that we utilise R package called "rPref" described in [124]. This package generates a *Skyline* of a dataset by selecting tuples which are Pareto-optimal with respect to given optimisation goals. Only those tuples are returned

which are not dominated by any other tuple. A tuple dominates another tuple if it is better in all relevant dimensions and strictly better in at least one dimension. We passed the obtained Pareto fronts from MOGP runs to the Skyline and generated the global one. Figure 7.2 presents the global Pareto front associated with the run number that contributes the non-dominated solutions to it. This is clearly indicates the need to perform multiple runs.

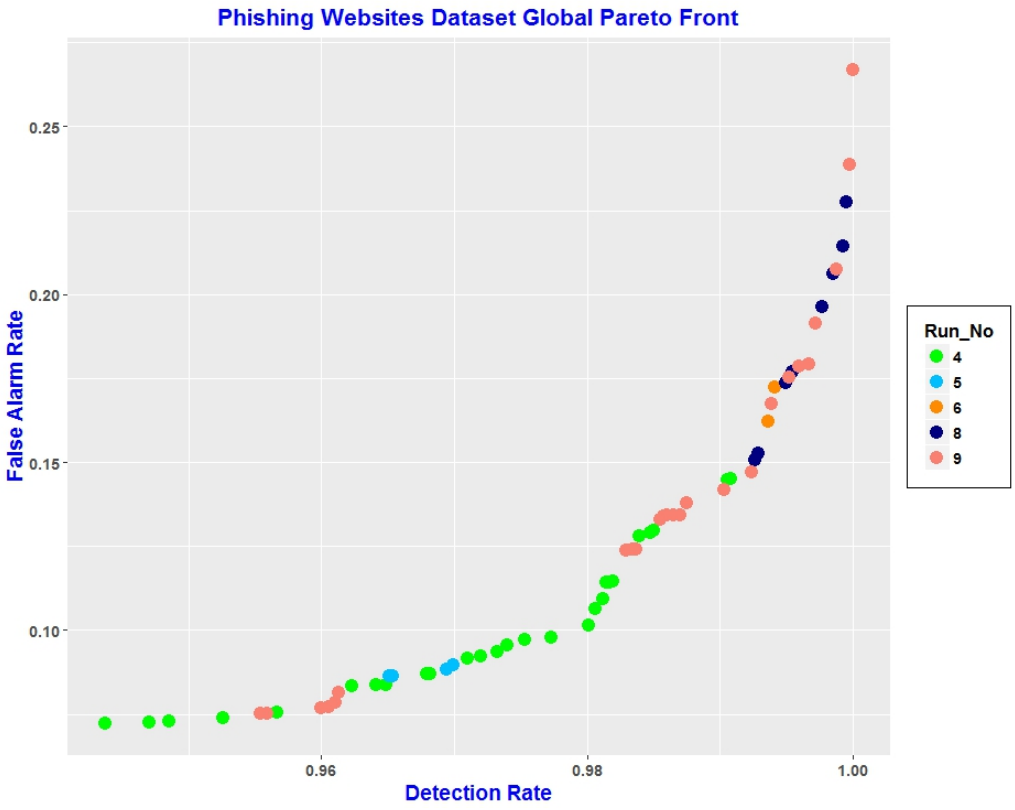

Figure 7.2: Global Pareto Front Trade-offs between DR vs. FAR

We extended our implementation to check the effect of increasing MOGP tree depth since MOEC will help control the size of the evolved solutions. We ran a second experiment with tree depth = 17 under the same objectives and compared its global Pareto front with that obtained when depth = 6. Figure 7.3 demonstrates the comparison which clearly shows that the tree depth (17) experiment contributes more solutions to the global Pareto front.

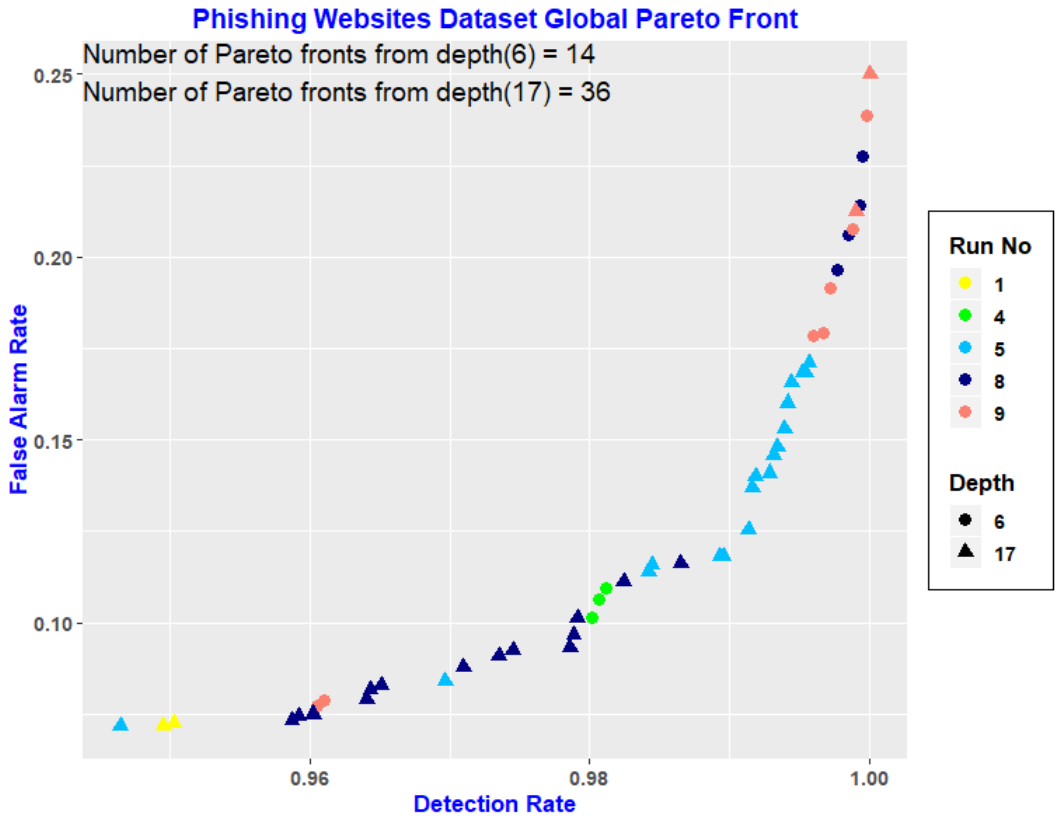

Figure 7.3: Global Pareto Front Trade-offs with Different Depths

Some optimal programs from the phishing websites dataset experiment, when their tree depth is set to 17, are shown in Figure 7.4 and Table 7.2. The overall performance of these programs applied on the testing portion is demonstrated in Table 7.1. In comparison with the standard GP execution (depth = 6), the best-evolved program achieved 89.09%, 8.47%, 91.81% of DR, FAR and Acc respectively with the tree size equal to 79. It clearly can be seen that MOGP not only maintains the solutions size but improved the performance as well compared to standard implementation. The Pareto front contains many other evolved solutions which have a variety of trade-offs.

Program No.	**MOGP (depth = 17)**			**Tree Size**
	DR (%)	**FAR (%)**	**Acc (%)**	
1	94.23	7.72	92.03	49
2	93.20	10.78	88.73	27
3	96.91	9.93	89.23	29

Table 7.1: The Performance of Some Programs Evolved by MOGP (Depth = 17)

Program No.	Evolved Program
2	Abs[Abnormal_URL] == Cos[SSLfinal_State] || Exp[Exp[Links_pointing_to _page] + Prefix_Suffix] <= Links_in_tags / Prefix_Suffix || SSLfinal_State <= Log[Log[URL_of_Anchor * Log[URL_of_Anchor]]]
3	URL_of_Anchor != Ceiling[Sqrt[URL_of_Anchor]] || (Prefix_Suffix < Cos[web_traffic] && Max[Prefix_Suffix, Log[Cos[web_traffic]]] >= URL_of_Anchor) || SSLfinal_State < Max[Prefix_Suffix, Tanh[Cos[Abnormal_URL]]] || RightClick > SSLfinal_State

Table 7.2: Example Programs Output Evolved by MOGP (Depth = 17)

In this research, a steady state MOGP technique is employed to discover more complex relationships than a standard GP could. The default parameters defined in koza.params and spea2.params that are included in the ECJ package are utilised, with the exception of the following:

- Number of generations: 50.
- Number of runs: 10 (with different seeds).
- Population size: 1000.
- SPEA2 archive size: 100.
- Evolutionary operators (probability): crossover (0.9) and mutation (0.1).
- Selection: Tournament Selection, size 7.

Furthermore, the same function set used in standard GP experiments is adopted here as well. These settings were chosen based on producing the best results during extensive experimental work. The experiments were conducted on Intel Core i7 CPU @ 3.40 GHz with 16 GB RAM running the 64-bit Windows 10 operating system.

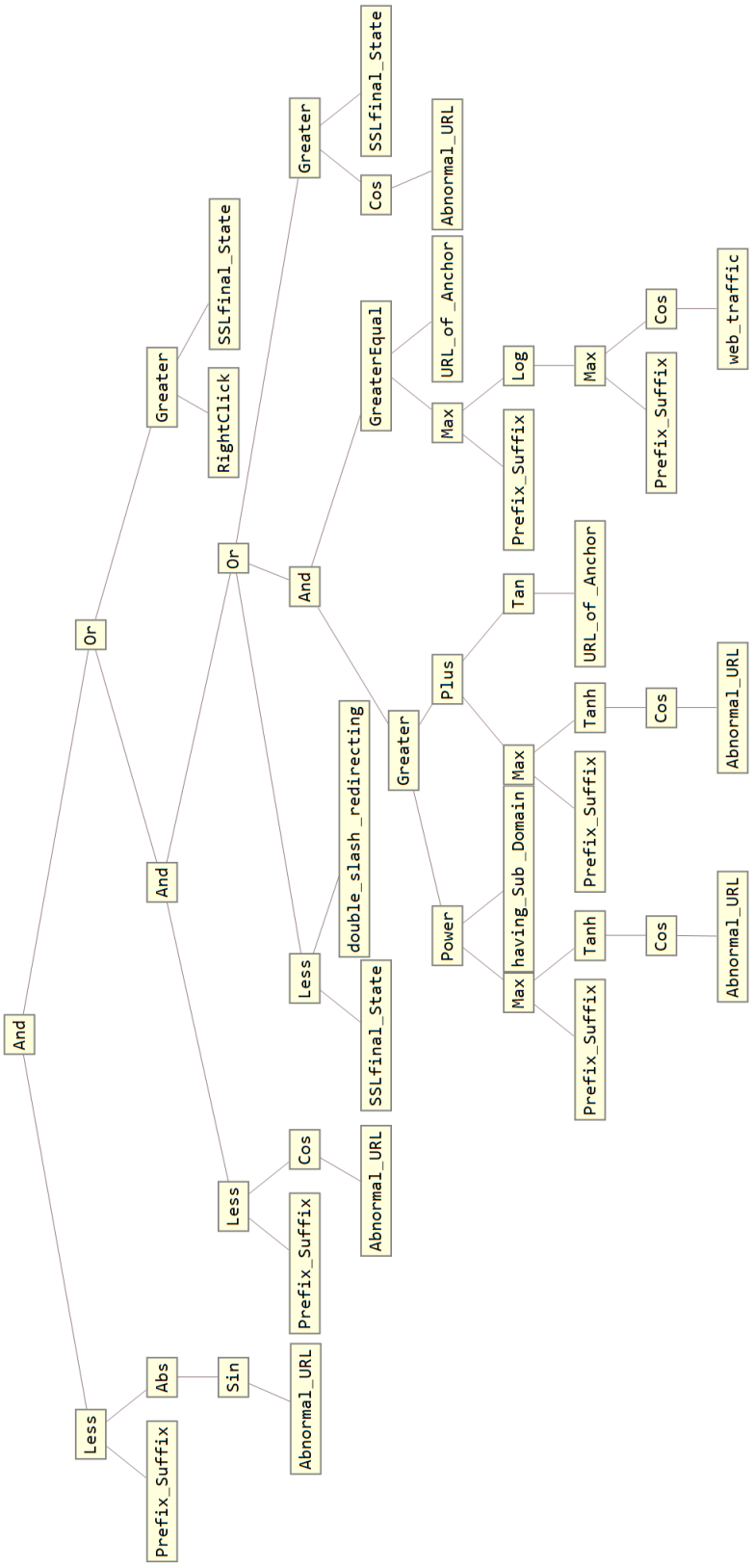

Figure 7.4: The Visualisation of The Evolved Program 1 (Depth = 17) using The ExpressionTreePlot Function From Mathematica [114]

7.2 Discovering Trade-offs in Intrusion Detection Programs

Nowadays, the types of intrusions are changing continuously and becoming increasingly complex. This made IDS an ongoing and challenging task for network administrators and security professionals. IDS system designers are concerned with the development of both effective and efficient solutions. In the context of constructing IDSs there would appear to be very little in the way of methodology to handle both aspects simultaneously. There is untapped potential in the use of evolution search techniques to discover IDS rules considering both functional and non-functional performance criteria. There have been few resource-aware applications in the IDS domain that have been implemented in operational "real world" settings. We are engaged in developing multi criteria trade-offs among the detection ability of intrusion detection programs and their resource consumption. The main reason is that humans are not especially skilled at making optimal decisions when it comes to complex trade-off choices. Almost all research concentrates on the major performance criteria (i.e. the various detection and alarm rates); very few research papers address the issues of trade-offs that involve resource consumption.

IDS is usually implemented in software, however, a significant part of IDS performance depends on the capability of the devices running these programs. For instance, if the CPU has already been specified for the device, then it is essential that the software solution be efficient enough so that scheduling the given set of tasks on the processor is possible. Furthermore, due to the possibility of deploying IDS programs into various devices that vary in their resources, IDS programs requirements (i.e. software and hardware) must take into account limited resources. The multi-objective method offers the ability to satisfy such needs. In this research IDS rules demands, such as reduced complexity, lower memory consumption and faster processing time, are investigated. In addition, the capabilities for detecting known and unknown attacks are discovered using multi-objective optimisation techniques.

7.2.1 Experiment 1: Feature Selection

Currently, network environments are transmitting ever-increasing amounts of data and it is very difficult for IDSs to examine all extracted features of the data flow. The main idea is to create the detection programs based on the input features and the operations performed upon them. That is why having redundant features slows down the detection process, and makes it difficult to process a large amount of traffic in real

time [27]. Ideally, we would be able to feed the most discriminating features to the algorithm before the learning stage (i.e. building IDS programs) began. This reduces training time and makes it easier to interpret the outputs. Feature selection is the process of identifying the feature subset that actually has an influence on the outcome. It can improve computational efficiency due to the smaller number of features, avoid the curse of dimensionality and visualising the data becomes more intuitive [125] [76].

We now aim to investigate 3 conflicting objectives: security (detection rate and false alarm rate) and non-security (number of features) using our MOGP approach. Figure 7.5 illustrates Pareto fronts of the three objectives and the united Pareto front obtained after 10 runs. However, the number of runs, which provided solutions that are non-dominated by any other solutions, is 3. The colours in these 3 dimensional figures refer to the solution achievement on the z-axis (in this case the number of features).

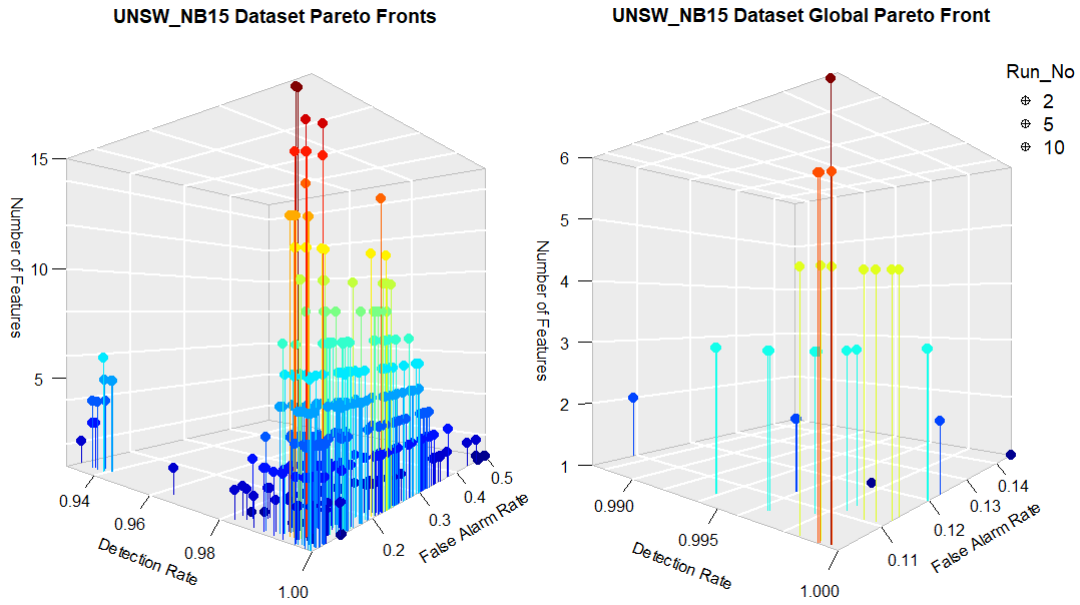

Figure 7.5: Feature Selection Pareto Fronts and Global Pareto Front Trade-off

The achievements of some of these evolved programs are presented in Table 7.3 and the programs output are demonstrated in 7.4. Program 1 used 2 features only (i.e. ct_state_ttl and sttl) out of 39 available in the UNSW-NB15 dataset. Whereas, program 2 adopted 6 features (i.e. ct_state_ttl, ct_dst_src_ltm, res_bdy_len, dur, sloss and dinpkt) compared to 3 features utilised by program 3 (i.e. ct_state_ttl, is_sm_ips_ports and sttl). Programs 1 and 3 achieved the highest detection rate recorded by standard GP experiments on this dataset. However, they had a relatively high false alarm rate. Programs with almost the same security performance as acquired by standard GP, but with a fewer number of features are evolved by using the MOGP

technique. More importantly, the tree (i.e. program) size was reduced dramatically using this implementation compared with the best-evolved program by standard GP with tree size equal to 80 that contains 16 features.

Program No.	**MOGP**			**No. of Features**
	DR (%)	**FAR (%)**	**Acc (%)**	
1	99.07	21.62	80.47	2
2	97.56	16.23	85.16	6
3	98.88	21.63	80.44	3

Table 7.3: Feature Selection Experimental Results using the Testing Dataset

Program No.	**Evolved Program**
1	Cos[Sqrt[Cos[Sqrt[ct_state_ttl]]]] > Cos[sttl + Min[Sqrt[ct_state_ttl], Tan[sttl]]]
2	(Log[Sin[ct_state_ttl]] >= Sin[Log[Sin[Log[ct_dst_src_ltm]]]]) || (Sin[Log[Log[Max[res_bdy_len, Log[Log[Sin[Log[Max[res_bdy_len, Log[Log[dur]]]]]]]]]] >= Max[sloss, Sin[Log[Sin[Log[dinpkt]]]]])
3	Max[Sin[ct_state_ttl], Sin[Sin[ct_state_ttl]]] > Max[Abs[is_sm_ips_ports], Abs[Sin[Sin[Sin[sttl]]]]]

Table 7.4: Feature Selection Experimental Programs Output

To further test the robustness of MOGP as a features selection technique, we adopted standard GE and CGP implementations from Chapter 5. We run each algorithm using the selected features 20 times and compared their outcomes with the outcomes using the full set of features (i.e. Table 5.4). It is shown from the results in Table 7.5 that the GE algorithm improved the performance of all the measurements when adopting features from MOGP programs. On the other hand, the CGP algorithm when using features selected by programs 1 and 3 managed to enhance detection rate only compared to using the full set. Whereas features selected by program 2 helped CGP achieve the best performance on the UNSW-NB15 dataset. This clearly indicates that the MOGP technique helped to provide an insight into which features are powering the predictions the most.

Features from	**GE**			**CGP**		
	DR	**FAR**	**Acc**	**DR**	**FAR**	**Acc**
Program 1	98.79 ± 0.02	21.77 ± 0.1	80.31 ± 0.08	98.79 ± 0.01	21.65 ± 0	80.41 ± 0
Program 2	92.44 ± 0.56	18.73 ± 1.06	82.39 ± 0.93	96.40 ± 0.15	12.42 ± 0.17	88.47 ± 0.14
Program 3	98.93 ± 0.02	21.83 ± 0.13	80.26 ± 0.12	98.96 ± 0.02	21.59 ± 0.01	80.48 ± 0.01

Table 7.5: The Performance of Standard GE and CGP with Selection Features (%)

7.2.2 Experiment 2: Memory Consumption

The trade-offs among intrusion detection strength and memory consumption of IDS programs are also taken into consideration. Researchers suggest enhancing the system throughput with the use of embedded memory applications, however, there is a necessity to reduce the amount of required memory since embedded memory is very expensive [126]. There are various proposed IDSs that have considered reducing memory usage either by improving string matching [127] [126] or optimising the memory requirements of each sensor node in a wireless network [128]. In contrast, our framework focuses on the rule itself by forcing the evolving process to consider the least memory hungry rules. In this investigation, we will show how security performance is also affected by memory saving. Normally the greater the number of features employed, the higher the memory consumption. For memory measurement, we used the Java Agent for Memory Measurements (JAMM) [129]. JAMM computes Java object memory use at runtime. MOGP tries to evolve memory-efficient IDS programs.

Figure 7.6 shows the optimal solutions and global optimal solutions of 10 runs found for the phishing websites dataset. It presents the plots of detection rate and false alarms which are evolved conditionally on the memory usage of the programs. These charts show that the programs with higher security performances (detection rate and low false alarm rate) values use more memory, as expected.

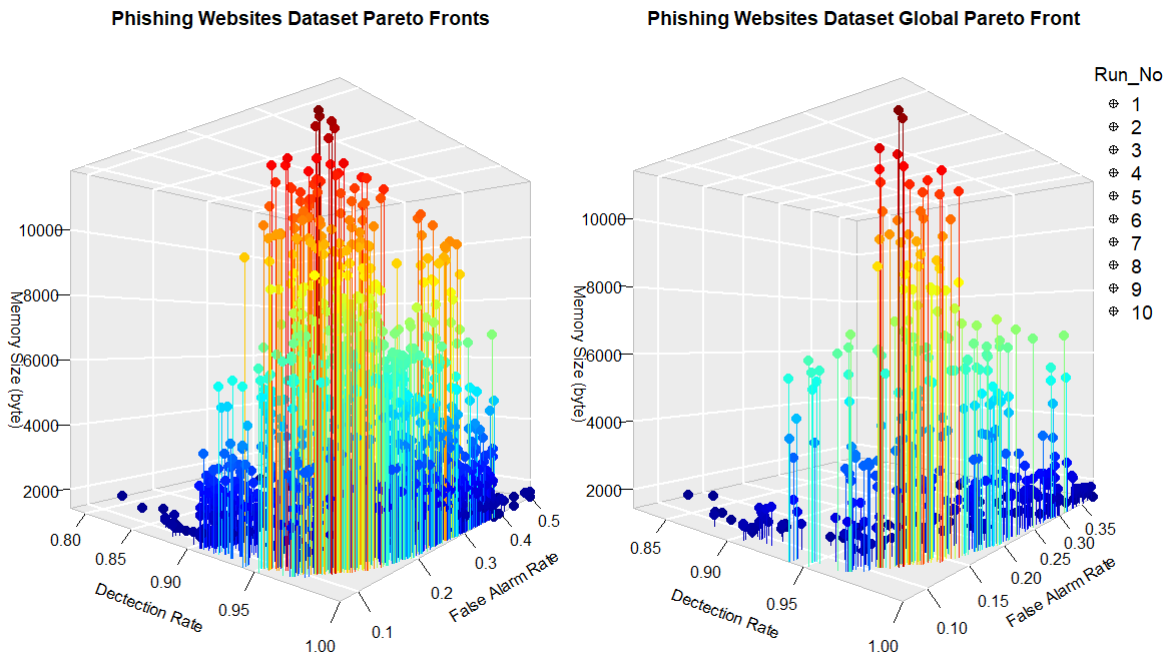

Figure 7.6: Memory Consumption Pareto Fronts and Global Pareto Front Trade-off

Table 7.6 compares the performance of some programs to their memory consumption. These programs have a better detection accuracy compared to standard GP and provide a reduction in memory usage. The best-evolved standard GP program consumes memory of 4,600 bytes tested on the same dataset. Table 7.7 shows the evolved programs output, where it clearly can be seen that the more complex the program is, the more memory it requires. Note that the memory computation is based on the memory usage required by an evolved program over the whole testing dataset and not based on a single instance in the dataset.

Program No.	**MOGP**			**Memory Size (bytes)**
	DR (%)	**FAR (%)**	**Acc (%)**	
1	89.40	9.37	90.77	1,920
2	95.57	9.27	90.14	2,784
3	96.39	8.90	90.45	3,456

Table 7.6: Memory Consumption Experimental Results using the Testing Dataset

Program No.	**Evolved Program**
1	Exp[Min[age_of_domain, Prefix_Suffix]] + SSLfinal_State <= Floor[Cos[Statistical_report] ^ Min[SSLfinal_State, Tan[URL_of_Anchor]]]
2	Min[Exp[URL_of_Anchor], Tanh[Tan[SSLfinal_State]]] < Cos[web_traffic] && Prefix_Suffix <= Redirect && ((Tanh[Max[double_slash_redirecting, HTTPS_token]] >= URL_of_Anchor && Tanh[Tanh[web_traffic]] >= URL_of_Anchor && URL_Length <= port) || RightClick > SSLfinal_State)
3	Prefix_Suffix <= Redirect && ((Cos[web_traffic] >= URL_of_Anchor && Tanh[Max[double_slash_redirecting, HTTPS_token]] >= URL_of_Anchor && Min[Exp[URL_of_Anchor], Tanh[Tan[SSLfinal_State]]] < Cos[web_traffic] && (Tanh[Tanh[web_traffic]] >= URL_of_Anchor || Tan[SSLfinal_State] <= Redirect)) || RightClick > SSLfinal_State)

Table 7.7: Memory Consumption Experimental Programs Output

7.2.3 Experiment 3: Processing Time

The steady rise in link speeds and the need for more sophisticated packet processing impose challenges towards IDSs [130]. Thus, one of the crucial parts of an IDS is how fast its programs can classify an event. Many incorporate IDSs with different techniques (software/hardware) in order to speed up the processing of the data and be able to process in real time. For instance, they used feature extraction [27], Hadoop [9] and graphics processing units [130]. In this experiment, we optimised the IDS programs

overall processing time which includes both data reading time and time spent in detection. Note time for the pre-processing steps is not included in this experiment as they are already performed for the utilised datasets. However, a study [131] concluded that 75% of the total processing time in IDSs is accounted for by rules checking. MOGP tries to optimise evolved programs processing times while maintaining detection rates and false alarms. The calling of the Java runtime library System.nanoTime() before and after the program evaluation is assigned as a non-security objective is to be reduced as much as is practical.

MOGP was run 10 times. Every run generated a Pareto front of non-dominated programs. Figure 7.7 illustrates the Pareto fronts and its union Pareto front (only 9 runs contributed to it) taken from the Kyoto 2006+ dataset. Programs with excellent security performance have a tendency to require more processing time as displayed.

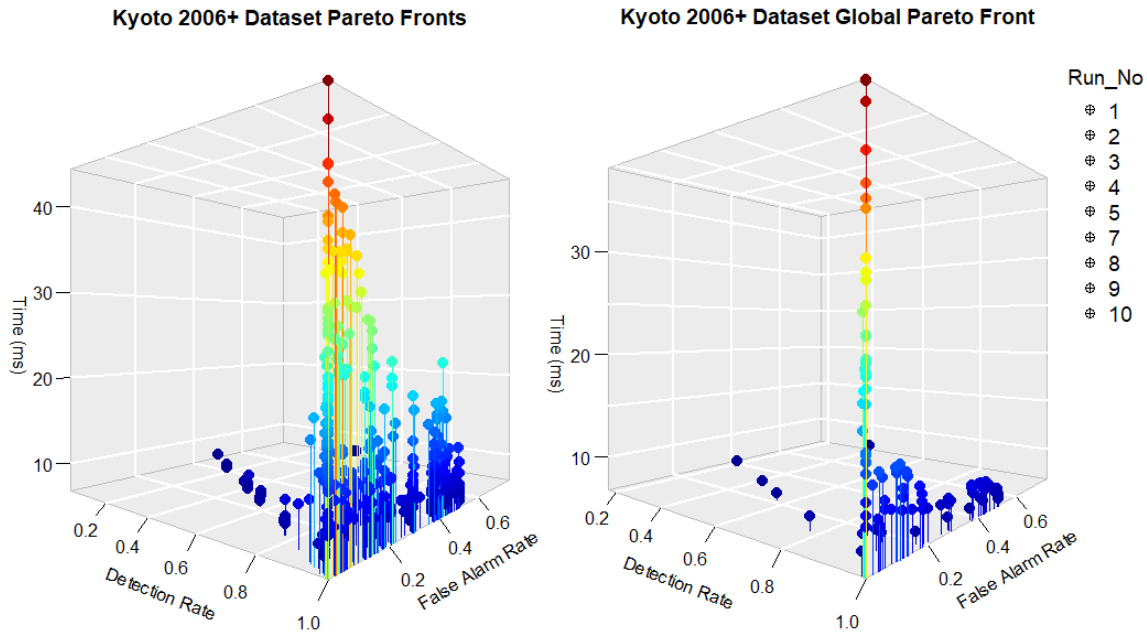

Figure 7.7: Processing Time Pareto Fronts and Global Pareto Front Trade-off

The performance of evolved programs is demonstrated on the testing dataset, including their processing times and memory consumption are presented in Table 7.8. In this implementation, only the detection rate showed an improvement compared to standard GP programs. However, MOGP returns a set of programs with far shorter processing times than the best-evolved standard GP program with (138.76 ms). Table 7.9 shows the output of the programs evolved using MOGP.

Program No.	MOGP			Time (ms)	Memory Size (bytes)
	DR (%)	FAR (%)	Acc (%)		
1	99.72	1.55	98.64	15.99	1,896
2	100	1.42	98.80	7.80	1,728
3	99.99	1.40	98.81	13.60	1,872

Table 7.8: Processing Time Experimental Results using the Testing Dataset

Program No.	Evolved Program
1	Ceiling[Cos[Abs[serror_rate]]] >= Sqrt[Log[dst_host_srv_serror_rate] / Sqrt[Floor[Cos[service]]]]
2	service > dst_host_srv_count || Sqrt[Floor[Cos[service]]] <= dst_host_count
3	dst_host_srv_count <= service || count < Cos[service] || Tanh[dst_host _same_src_port_rate] <= Cos[service]

Table 7.9: Processing Time Experimental Programs Output

7.2.4 Experiment 4: Ensemble Diversity

Ensembles essentially aim to overcome individual model weaknesses by appealing to the capabilities of multiple models. The weaknesses of any individual model should be outweighed by the classification strength of the others. This generally means that classification errors are "smeared out" across the classifiers, or put another way, their strengths and weaknesses are complementary. In many cases, all models may directly agree but in others we only need the ensemble to be right. Diversity is a measure that defines the degree of disagreement in the output of the base models utilised to build an ensemble. The diversity concept in the context of classifier combination does not just indicate that the base models should be different; it also indicates that the base models make errors in different instances [121]. In order to analyse the ensemble diversity effect on evolved programs, MOGP will be used to guide the building of the ensemble systems. This could be achieved by incorporating an ensemble diversity measure as a penalty into the MOGP fitness measurement. To measure diversity among ensemble members, the Kohavi-Wolpert (KW) [132] measurement was adopted. The KW is defined in equation (7.1).

$$Kohavi - Wolpert\ (KW) = \frac{1}{(NL)^2} + \sum_{j=1}^{N} l(zj)(L - l(zj)) \tag{7.1}$$

Where N refers to the number of instances, L denotes to the number of the base models in the ensemble. Finally, $l(zj)$ contains the number of base models that correctly classified the instance zj. KW values range from 0 and 1, where 0 indicates no diversity

and 1 indicates the highest possible diversity.

The MOGP algorithm is run to evolve functional properties (detection rate and false alarm rate) and a non-functional property (stacked model diversity) in intrusion detection programs. The algorithm aims to maximise both detection rate and ensemble diversity while minimising the false alarm rate. The aim of this experiment is the selection of optimal sets of base models using the MOGP approach. This would help to produce accurate, diverse and small ensembles. Figure 7.8 shows the plot diagrams for the three objectives of both Pareto fronts and the union of 10 runs. These results were obtained from the experiment on the modern DDoS dataset.

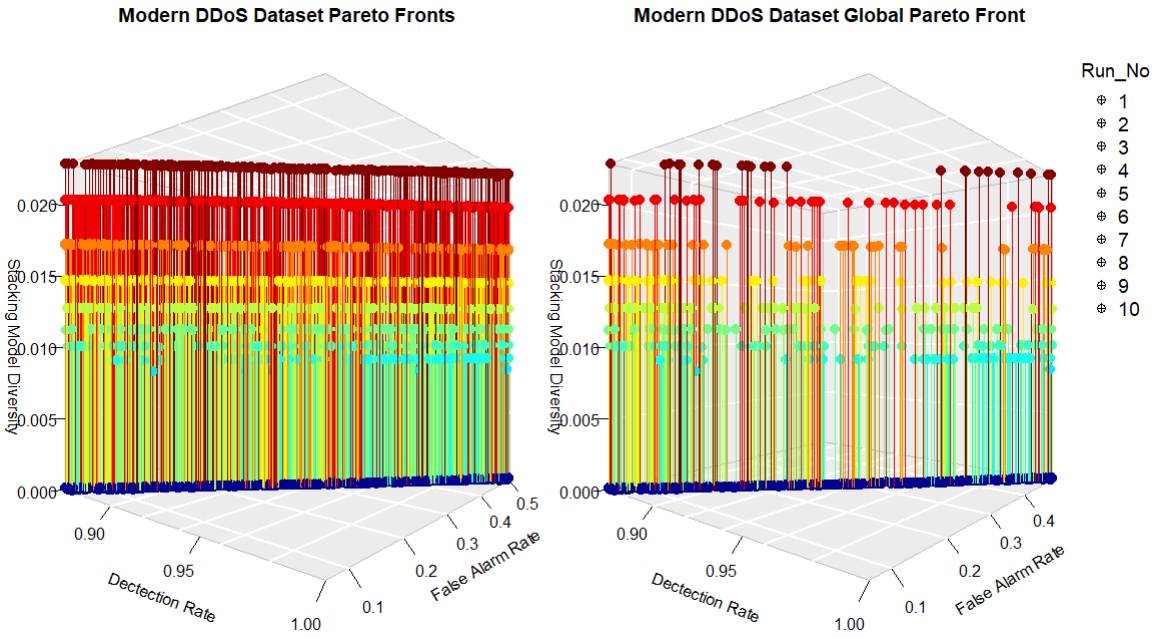

Figure 7.8: Ensemble Diversity Pareto Fronts and Global Pareto Front Trade-off

The pool of individuals consisted of only 10 members. However, there is a trade-off made based on the diversity measure used. The performance of some of these stacked programs evolved based on the testing dataset reported in Table 7.10. The MOGP evolved programs output are shown in Table 7.11. The results clearly demonstrate how MOGP helps to enhance stacked programs compared to the GP stacking method. Program 1 has the highest diversity measure. This program uses 2 base models with tree size = 3 compared to the best-evolved GP stacking program which uses 7 models with tree size = 114. It is important to bear in mind that the enhanced property is the solutions complexity rather than the performance. The main reason for that is the utilised base models have produced relatively similar outcomes.

Program No.	MOGP Stacking			Diversity
	DR (%)	FAR (%)	Acc (%)	
1	87.19	6.41	98.64	0.022
2	87.19	6.42	98.62	0.014
3	87.30	6.84	97.78	0

Table 7.10: Stacking Experimental Results using the Testing Dataset

Program No.	Evolved Program
1	GBM_4 < GBM_5
2	GBM_2 < GBM_5 || ((Sin[Exp[Sin[GBM_0 / (DNN / Log[GBM_4])] + Sin[Sin[Exp[DNN]]]]] + Sin[Sin[Abs[GBM_5]] / (DNN / Log[GBM_4])] > DNN || GBM_4 <= GBM_3) && Tanh[GBM_4] <= Cos[DNN])
3	GBM_4 ^ Tan[Sqrt[GBM_0]] > DNN || DRF <= GBM_4 ^ Tan[DRF]

Table 7.11: Ensemble Diversity Experimental Evolved Programs Output

7.2.5 Experiment 5: Detecting Unknown Attacks

Cybersecurity attacks can originate from new and unexpected sources [7]. A comprehensive attacks signature database is an essential component of modern IDS. However, signatures generally only detect known attacks [36]. To help with the problem, these maintained signatures (i.e. programs) need to make their capability to recognise unknown attacks more robust. Every evolved program during the training phase sees various instances of normal and attack behaviour. Later, the best-evolved individual tries to predict the instances of normal and attacks in the testing dataset. In this experiment, however, during the training phase, MOGP programs are evolved without seeing all the attacks and later its performance was tested on the complete set of attacks. The MOGP approach optimises the security objectives of evolved programs by increasing the detection rate and decreasing false alarms.

Firstly, the evaluation of the performances of the MOGP technique is conducted for unseen subfamilies by the known attack experiment. The experiment was executed on the modern DDoS dataset described in Table 5.6. Despite MOGP being able to avoid complex solutions compared to the standard GP, however, the evolved programs did not produce any improvements in the security performance. Some good results are demonstrated in Table 7.12. The tree representation of program no. 2 is illustrated in Figure 7.9. As can be seen from this figure, MOGP produced a far less complex solution than the best-evolved solution by standard GP (see Table 5.9). Table 7.13 shows the percentages of detection for each attack, with good outcomes for application layer

DDoS attacks especially. In this experiment, the Smurf attack continues to be the least identified attack.

Program No.	MOGP Modern DDoS Table (5.6)		
	DR (%)	FAR (%)	Acc (%)
1	88.22	6.68	97.24
2	88.17	6.59	97.44
3	88.10	6.28	98.05

Table 7.12: The Performance of Some Programs (Unknown Variations of Known Attack Experiment)

Attack Name	Program 1		Program 2		Program 3	
	No. of Attack Traffic Detected	DR (%)	No. of Attack Traffic Detected	DR (%)	No. of Attack Traffic Detected	DR (%)
Smurf	1,559	36.69	1,555	36.59	1,523	35.84
UDP Flood	62,061	90.07	62,022	90.01	61,997	89.97
SIDDOS	6,318	94.79	6,315	94.74	6,316	94.76
HTTP Flood	4,110	100	4,110	100	4,110	100
Total Records	74,048	88.22	74,002	88.17	73,946	88.10

Table 7.13: Detection Results of MOGP Evolved Programs

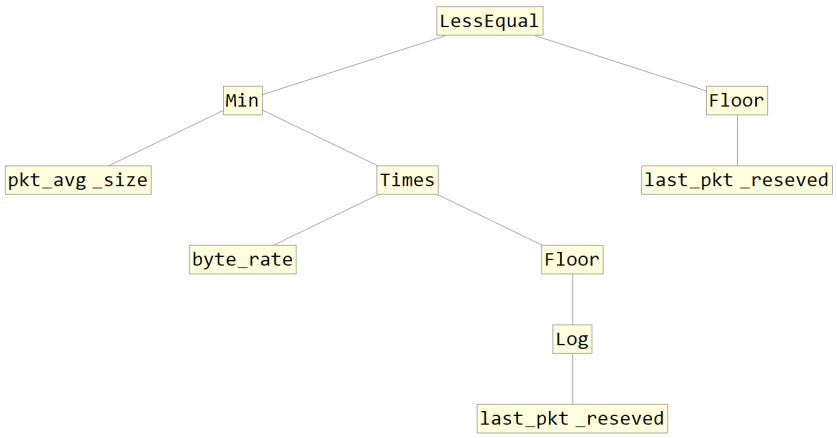

Figure 7.9: The Visualisation of Program 2

Secondly, we aim to test the MOGP implementation capability in detecting unseen attacks. The MOGP framework for optimising security objectives (i.e. detection rate and false alarm rate) run 10 times with each attack present in the UNSW-NB15 dataset. There are 9 attacks in this dataset. Figure 7.10 and Figure 7.11 show MOGP Pareto fronts and global front obtained from the Analysis attack and the Fuzzers attack experiment, respectively. There were 5 different runs contributing to the global front of

the Analysis experiment, whereas only 3 runs contributed in the case of the Fuzzers experiment. These figures illustrate the trade-offs made between DR and FAR by the Pareto front programs.

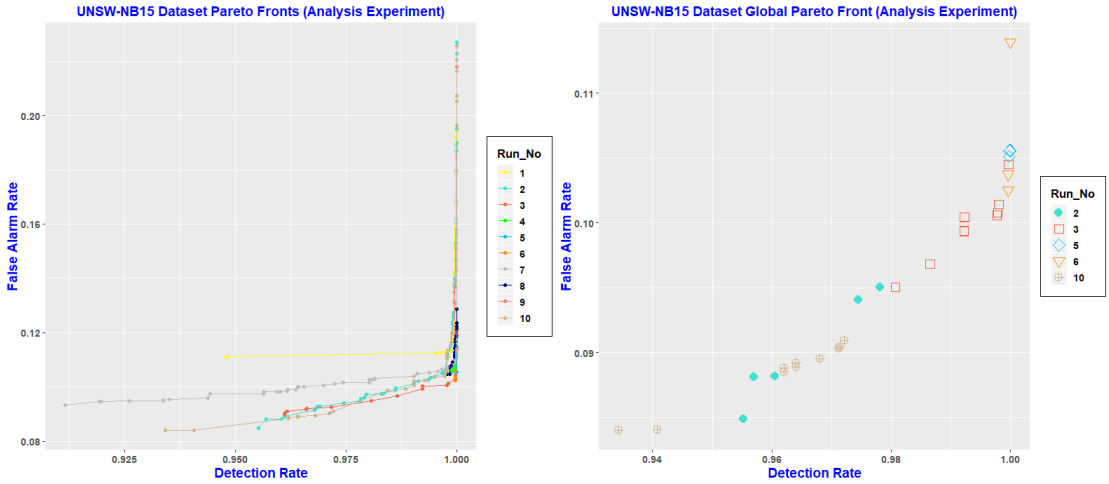

Figure 7.10: Pareto Fronts and Global Pareto Front Trade-off (Analysis Experiment)

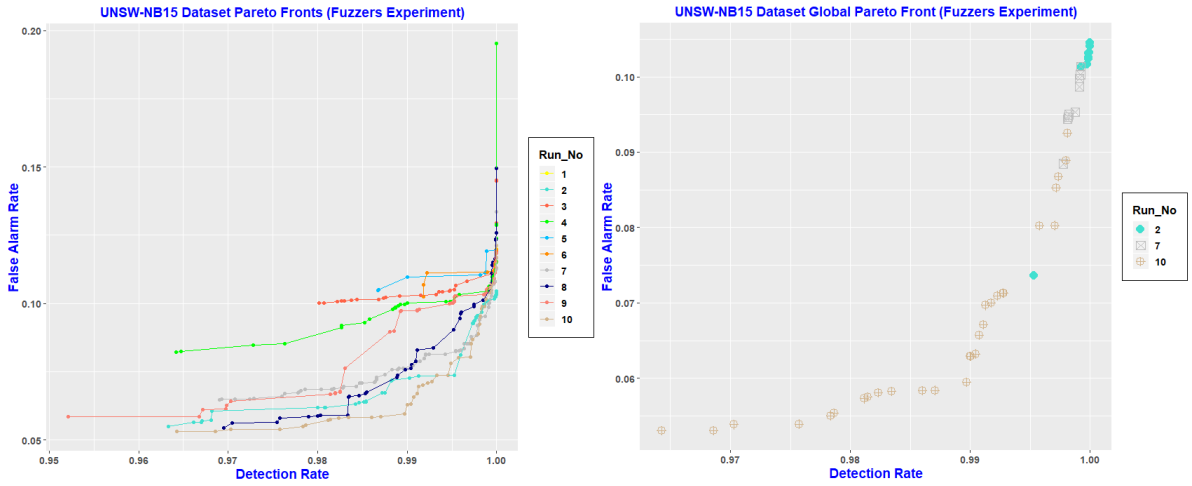

Figure 7.11: Pareto Fronts and Global Pareto Front Trade-off (Fuzzers Experiment)

The DRs of known and unknown attacks accomplished by some of the MOGP evolved programs from every experiment are presented in Table 7.14. There are many other non-dominated programs which have different performances. The highlighted cell refers to the missing attack during the training. It is apparent from this table that DRs of the attack records varied from one MOGP program to another. The lowest detected attack was Fuzzers which ranged from (47.11%) to (83.63%). Whereas the Generic attack was fairly easy to recognise with DRs ranging from (99.36%) to (99.78%). Overall, these programs were able to detect unknown attacks.

	Analysis	**Backdoor**	**DoS**	**Exploits**	**Fuzzers**	**Generic**	**Reconnaissance**	**Shellcode**	**Worms**
Analysis Experiment	2,200 (82.18%)	566 (97.08%)	3,839 (93.88%)	9,981 (89.66%)	2,856 (47.11%)	18,752 (99.36%)	3,105 (88.81%)	329 (87.03%)	37 (84.09%)
Backdoor Experiment	660 (98.95%)	2,237 (96.04%)	3,907 (95.54%)	9,440 (84.80%)	4,254 (70.17%)	18,801 (99.62%)	3,372 (96.45%)	365 (96.56%)	43 (97.72%)
DoS Experiment	656 (96.89%)	568 (97.42%)	15,773 (96.45%)	9,968 (89.54%)	4,887 (80.61%)	18,812 (99.68%)	3,350 (95.82%)	363 (96.03%)	43 (97.72%)
Exploits Experiment	659 (97.34%)	581 (99.65%)	3,961 (96.86%)	41,755 (93.77%)	5,070 (83.63%)	18,827 (99.76%)	3,369 (96.36%)	364 (96.29%)	43 (97.72%)
Fuzzers Experiment	675 (99.70%)	579 (99.31%)	4,000 (97.82%)	10,603 (95.24%)	14,250 (58.77%)	18,804 (99.64%)	3,390 (96.96%)	366 (96.82%)	44 (100%)
Generic Experiment	623 (92.02%)	576 (98.79%)	3,902 (95.42%)	10,314 (92.65%)	3,631 (59.89%)	58,744 (99.78%)	3,282 (93.87%)	335 (88.62%)	39 (88.63%)
Reconnaissance Experiment	656 (96.89%)	568 (97.42%)	3,910 (95.62%)	9,928 (89.18%)	4,715 (77.77%)	18,808 (99.66%)	13,189 (94.29%)	363 (96.03%)	43 (97.72%)
Shellcode Experiment	656 (96.89%)	568 (97.42%)	3,952 (96.64%)	10,025 (90.05%)	4,888 (80.63%)	18,818 (99.71%)	3,352 (95.88%)	1,447 (95.76%)	43 (97.72%)
Worms Experiment	659 (97.34%)	580 (99.45%)	3,993 (97.65%)	10,635 (95.53%)	4,677 (77.15%)	18,828 (99.77%)	3,380 (96.68%)	364 (96.29%)	165 (94.82%)

Table 7.14: No. of Attack Traffic Detected (DR) by MOGP Evolved Programs

7.3 Conclusions

In this chapter we extended our optimisation framework to employ MOEC to design robust solutions for complex environments. The multi-objective approach is beneficial since it can provide a set of optimised IDS programs. A security analyst can choose a solution in line with the purpose of the examined environment and modify the IDS settings according to need. We showed that the knowledge of the Pareto optimal solutions presents a principled means to differentiate the trade-offs involved.

This work showed that various trade-offs could be produced among classification accuracy and other properties. These trade-offs are recognised by utilising a MOGP approach. A multiple-objective approach offers a more efficient way of examining the trade-off space, especially when conflicts exist. Evidence suggests that for these environments the potential to produce good trade-offs will be especially necessary. Programs with equal (or better) classification accuracy than acquired by GP but with lower resource consumption are evolved by utilising our MOGP approach. Better still, a collection of solutions offering a variety of trade-offs is produced. It is left to the engineers to choose which particular solutions are suitable for a specific deployment.

These experiments are unusual since they trade off security abilities (detection rate and false alarms) against non-security attributes (complexity, memory and time). The inherent complexity of an environment's infrastructure and functions renders it challenging to see how IDS programs, with the best possible trade-offs, could be acquired via the means of standard system development. To summarise, an approach based on optimisation emerges as a natural and effective option for the challenge.

Chapter 8 | Summary and Conclusions

In this chapter we summarise the research and provide the conclusions to our work. The thesis hypotheses introduced in Chapter 1 are also reviewed to demonstrate how the work carried out in this research supports them. To conclude, open issues and future work are presented.

8.1 Summary of Experimentation

The security of the internet and computer networks are crucial problems. A great deal of collaboration has occurred to produce standards to help, but new attacks emerge every day and current mechanisms are not sufficient in themselves. The major problems for existing mechanisms for intrusion detection systems can be summarised as follows:

1. The lack of a common testbed.
2. Limited IDS capability for detecting known and unknown attacks.
3. Significant computational overheads incurred by IDS programs.

The first two are concerned with the trend functional capabilities of an IDS and the efficiency with which it can carry out its task. The third concerns a problem with whole are, namely the difficulty of providing a rigorous and meaningful evaluation of the capability. Many evaluations have been carried out using a limited number of datasets. Researchers often investigate a very limited number of attacks and make assumptions about the architectures of the solutions. In this thesis, four widely different deployment environments containing a contemporary set of threats and their variants are investigated. In the context of intelligent IDSs, we are not only concerned with the power of classifying known attack behaviours, but the potential for detecting novel attacks. Extant research in the use of ML for IDS has rarely considered non-functional properties. It usually targets only the typical detection and alarm rates. Then again, the problem of resource constraints is crucial. The restricted resources of IDS rules are factors to consider as well. The main objective of this thesis is developing intrusion detection systems that are able to accurately detect the internet and computer network threats

and do so with a low resource consumption, high speed and a simple design. In order to accomplish this goal, evolutionary computation algorithms have been used to evolve such intrusion detection programs. These approaches work directly on the datasets without any pre-processing. The thesis also presents the first application of Cartesian genetic programming synthesis of an intrusion detection system.

This thesis contains detailed descriptions regarding the use of evolutionary computation algorithms to the detection of threats targeting the internet and computer networks. The performance of programs evolved using GP, GE and CGP are tested in various environments that contain different kinds of threats (and their variations). The standard implementation of these paradigms showed a good capability to transform the features given as input into a decision (normal vs. anomaly). These programs were able to profile the symptoms of normal and anomalous behaviours. CGP evolved programs with better performance than those generated by GP and GE. These paradigms provided some degree of manual analysis where we were able to explore feature vectors and operators relations appearing in the evolved programs. The implementations were able to show a robust capability in detecting previously unseen attack behaviours. The frameworks and parameter settings used for GP, GE and CGP were also investigated and the paradigms were compared. As presented in *hypothesis 1: evolutionary computation are able to discover intrusion detection programs for the internet and computer networks based environments*. To assess this hypothesis we conducted different experiments.

The heterogeneous ensemble exploits the benefit of having multiple expert systems. Ensemble solutions evolved by GP, GE and CGP outperformed those evolved by their standard individual implementations, especially in achieving a higher detection rate. The number of false alarms decreased dramatically. However, experiments in the modern DDoS dataset did not show much improvement in terms of false alarms. The proposed paradigms worked as a meta-learner which provided a means to choose base models dynamically instead of stacking all of them. In this implementation, the proposed paradigms performed relatively similarly to each other. Also, we explored the effectiveness of the stacking paradigm to address the unseen attacks issue. The experimental results showed that the detection rate increased for some attacks (seen or unseen) and decreased for others compared to standard implementations. However, stacking programs produced lower false alarms and better accuracy. *Hypothesis 2: evolutionary computation can act as a meta-learner to evolve a function for generating stacking models (intrusion detection programs) that are more effective than previously demonstrated* is supported with the results in Chapter 6.

In Chapter 7, non-functional objectives like solution complexity, memory consumption, processing times and unknown attacks are investigated. Multi-objective evolutionary computation technique offers the designer a collection of solutions which demonstrate a variety of trade-offs between these objectives.

MOGP is utilised to explore the correlations amongst detection rate, false alarm rate and non-functional properties of evolved programs. In the results, there are programs which are made less complex by considering features reduction while evolving. In the same vein, ensemble solutions were optimised through increasing the diversity of the stacked base models. The trade-offs amongst the IDS capability, memory consumption and processing time needed are examined utilising a multi-objective optimisation approach, and interesting outcomes have been achieved. Finally, MOGP is extended to deal with unseen attacks. The evolved programs showed a higher detection rate than do standard and stacking programs. However, these programs had a higher false alarm rate compared to stacking programs. *Hypothesis 3: multi-objective evolutionary computation will be able to explore the trade-offs between functional (intrusion detection ability) and non-functional (complexity, memory usage and processing time) properties of evolved programs* is supported by these results.

8.2 Thesis Contributions

The contributions of this thesis are outlined as follows:

Demonstration of the use of evolutionary computation techniques for the synthesis of an intrusion detection system: This thesis shows that GP, GE and CGP can be utilised to evolve effective detectors for known and unknown attacks. To the best of our knowledge, it is the first application of CGP technique to the intrusion detection problem and the work provides a comparison with the other EC approaches.

Demonstration of the effectiveness of evolutionary computation techniques-based ensemble for synthesising intrusion detection systems: This research extended learning techniques from a single classifier system framework into a multiple classifiers system using the ensemble concept. An ensemble classifier has the ability to incorporate the strengths of various classifiers in a way that means their weaknesses are compensated for. This thesis showed that GP, GE and CGP techniques can be used to build stacked models capable of differentiating environment behaviours more effectively. In these systems the percentage of false alarms reduced dramatically in most

cases. In addition, stacked models showed an ability to detect unknown attacks. There has been no detailed investigation of using GE and CGP based ensembles for intrusion detection problems before.

Efficiency: This research proposes an approach that explores various trade-offs between security and non-security properties of evolved programs. Our technique can be applied to create a collection of solutions with the optimal trade-offs that could be obtained. These sets of programs provide a variety of trade-offs amongst intrusion detection capability and its utilised features (needed inputs), memory consumption and processing times. These needs are discovered by using the multi-objective optimisation technique, which provides a more efficient means of building IDS programs. Furthermore, MOGP investigates security properties of evolving programs to detect both known and unknown attacks. Finally, these present an implementation of a new perspective on the selection of base learners for stacked IDS programs, using the concepts of diversity and Pareto optimality. This helped clarify why an ensemble obtains particular learners. This research represents a novel application of SPEA2 to multi-objective IDS considering properties such as processing times, diversity and unknown attacks. Even though feature selection and memory consumption implementations have been investigated before in other approaches, SPEA2 was understandably different in terms of the features selection mechanism and the investigated environment for memory consumption. A final selection between programs offering different trade-offs rests with the system designer.

8.3 Future Research

The design of robust defence techniques appears to be a never-ending task. In this thesis, we have presented contributions to prevent contemporary threats. These contributions leave space for extended work and open new research challenges. This section will present possible areas for future research:

Applying evolutionary computation techniques to new environments: In this thesis, we present how to utilise evolutionary computation algorithms to the problem of intrusion detection and how to examine a variety of trade-offs amongst properties in resource-restricted environments. The work introduced in this research could be adjusted with ease to a specific environment, for instance, IoT. It is anticipated that for IoT the capability to obtain potentially the best (or near best) trade-offs will be especially significant given that these networks may often be even more resource-restricted.

Improving our approach: In this work, we have primarily concentrated on IDS detection capabilities. However, our approaches could be improved in the following two ways: incorporation of active responses and improving how quickly defenders recover from an attack. The second would be helped by our evolved programs identifying the type of the attack (and not just whether the system is under attack). Further improvements may also be sought by using complementary base learners, i.e. we would aim to synthesise classifiers with complementary weaknesses.

Adaptive systems: The works presented have been designed as an off-line learning approach based on available data and circumstances. However, when the protected environment experiences changes, the algorithms need to be retrained. To overcome this drawback, adaptive IDSs readjust automatically. In order to do that, the idea is that rather than stopping learning immediately after the training phase, the algorithms continue gaining knowledge throughout the operation of the system. Hence, the system learns from wrongly categorised cases. This procedure may deal with various issues linked with IDSs: it decreases high false positive rates, avoids retraining and the systems adjust to changing environments. This is an important issue for future research.

Application to other contexts: Aside from purely security-related matters, EC algorithms have been widely used to address various real-world applications. The authors are wondering how current implementations could be used to investigate non-security contexts, for instance, medicine, biology and bioinformatics applications. In addition, the MOGP implementation could be utilised for other domains in which there are conflicting objectives.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[2] A. Whitmore, A. Agarwal, and L. Da Xu, "The internet of things—a survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[3] *Internet world stats*. [Online]. Available: `https://www.internetworldstats.com/emarketing.htm` (visited on 10/01/2019).

[4] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[5] T. Jordan and P. Taylor, "A sociology of hackers," *The Sociological Review*, vol. 46, no. 4, pp. 757–780, 1998.

[6] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network security*, vol. 2011, no. 8, pp. 16–19, 2011.

[7] "Symantec internet security threat report," 2018. [Online]. Available: `https://www.symantec.com/`.

[8] *Cve security vulnerability database*. [Online]. Available: `https://www.cvedetails.com/browse-by-date.php` (visited on 10/02/2019).

[9] M. M. Rathore, A. Ahmad, and A. Paul, "Real time intrusion detection system for ultra-high-speed big data environments," *The Journal of Supercomputing*, vol. 72, no. 9, pp. 3489–3510, 2016.

[10] E. Cambiaso, I. Vaccari, E. Punta, S. Scaglione, S. Bianchi, A. Zarca, R. Trapero, P. Sobonski, and D. Rivera, "Attacks threats analysis and contingency actions," 2018. [Online]. Available: `http://www.anastacia-h2020.eu/`.

[11] "Radware's global application and network security report," 2018. [Online]. Available: `https://www.radware.com/`.

[12] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[13] "Cisco 2018 annual cybersecurity report." [Online]. Available: `https://www.cisco.com/c/en_uk/products/security/security-reports.html`.

[14] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy (SP)*, IEEE, 2010, pp. 305–316.

[15] H. He, C. Maple, T. Watson, A. Tiwari, J. Mehnen, Y. Jin, and B. Gabrys, "The security challenges in the iot enabled cyber-physical systems and opportunities for evolutionary computing & other computational intelligence," in *IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2016, pp. 1015–1021.

[16] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST special publication*, vol. 800, p. 94, 2007.

[17] G. Kumar and K. Kumar, "The use of artificial-intelligence-based ensembles for intrusion detection: A review," *Applied Computational Intelligence and Soft Computing*, vol. 2012, p. 21, 2012.

[18] J. P. Anderson, "Computer security threat monitoring and surveillance," *Technical Report, James P. Anderson Company*, 1980.

[19] L. Teresa, R. Jagannathan, L. Rosanna, L. Sherry, D. L. Edwards, P. G. Neumann, H. S. Javitz, and A. Valdes, "Ides: The enhanced prototype, a real-time intrusion detection system," *Technical Report SRI Project 4185–010, SRI-CSL-88–12, CSL SRI International, Computer Science Laboratory, SRI Intl.*, 1988.

[20] D. Denning and P. G. Neumann, *Requirements and model for IDES-a real-time intrusion-detection expert system*. SRI International, 1985.

[21] S. Sen, "A survey of intrusion detection systems using evolutionary computation," *Bio-Inspired Computation in Telecommunications*, pp. 73–94, 2015.

[22] S. hafez Amer and J. A. hamilton Jr, "Intrusion detection systems (ids) taxonomy-a short review," *This is a paid advertisement. STN 13-2 June 2010: Defensive Cyber Security: Policies and Procedures 2*, p. 23,

[23] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[24] A. A. Aburomman and M. B. I. Reaz, "A survey of intrusion detection systems based on ensemble and hybrid classifiers," *Computers & Security*, vol. 65, pp. 135–152, 2017.

[25] G. Folino and P. Sabatino, "Ensemble based collaborative and distributed intrusion detection systems: A survey," *Journal of Network and Computer Applications*, vol. 66, pp. 1–16, 2016.

[26] Y. Bouzida and F. Cuppens, "Neural networks vs. decision trees for intrusion detection," in *IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM)*, 2006, pp. 81–88.

[27] Z. Banković, D. Stepanović, S. Bojanić, and O. Nieto-Taladriz, "Improving network security using genetic algorithm approach," *Computers & Electrical Engineering*, vol. 33, no. 5-6, pp. 438–451, 2007.

[28] D. Stevanovic, N. Vlajic, and A. An, "Detection of malicious and non-malicious website visitors using unsupervised neural network learning," *Applied Soft Computing*, vol. 13, no. 1, pp. 698–708, 2013.

[29] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," *IEEE transactions on computers*, vol. 65, no. 10, pp. 2986–2998, 2016.

[30] M. M. Najafabadi, T. M. Khoshgoftaar, and N. Seliya, "Evaluating feature selection methods for network intrusion detection with kyoto data," *International Journal of Reliability, Quality and Safety Engineering*, vol. 23, no. 01, p. 1 650 001, 2016.

[31] S. Chebrolu, A. Abraham, and J. P. Thomas, "Feature deduction and ensemble design of intrusion detection systems," *Computers & security*, vol. 24, no. 4, pp. 295–307, 2005.

[32] C. Xiang, P. C. Yong, and L. S. Meng, "Design of multiple-level hybrid classifier for intrusion detection system using bayesian clustering and decision trees," *Pattern Recognition Letters*, vol. 29, no. 7, pp. 918–924, 2008.

[33] A. Abraham and J. Thomas, "Distributed intrusion detection systems: A computational intelligence approach," in *Information Security and Ethics: Concepts, Methodologies, Tools, and Applications*, IGI Global, 2008, pp. 1639–1659.

[34] A. Zainal, M. A. Maarof, and S. M. Shamsuddin, "Ensemble classifiers for network intrusion detection system," *Journal of Information Assurance and Security*, vol. 4, no. 3, pp. 217–225, 2009.

[35] I. Syarif, E. Zaluska, A. Prugel-Bennett, and G. Wills, "Application of bagging, boosting and stacking to intrusion detection," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, Springer, 2012, pp. 593–602.

[36] M. H. Kamarudin, C. Maple, T. Watson, and N. S. Safa, "A logitboost-based algorithm for detecting known and unknown web attacks," *IEEE Access*, vol. 5, pp. 26 190–26 200, 2017.

[37] F. Thabtah, R. M. Mohammad, and L. McCluskey, "A dynamic self-structuring neural network model to combat phishing," in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2016, pp. 4221–4226.

[38] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016.

[39] N. Moustafa, J. Slay, and G. Creech, "Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks," *IEEE Transactions on Big Data*, 2017.

[40] M. Alkasassbeh, G. Al-Naymat, A. B. Hassanat, and M. Almseidin, "Detecting distributed denial of service attacks using data mining techniques," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 1, pp. 436–445, 2016.

[41] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," in *ICISSP*, 2018, pp. 108–116.

[42] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[43] G. Kumar and K. Kumar, "Ai based supervised classifiers: An analysis for intrusion detection," in *Proceedings of the International Conference on Advances in Computing and Artificial Intelligence*, ACM, 2011, pp. 170–174.

[44] S. Sen and J. A. Clark, "Evolutionary computation techniques for intrusion detection in mobile ad hoc networks," *Computer Networks*, vol. 55, no. 15, pp. 3441–3457, 2011.

[45] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015, vol. 53.

[46] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1, ISBN: 0262111705.

[47] W. Banzhaf, P. Nordin, R. Keller, and F. Francone, *Genetic Programming-An Introduction: On the Automatic Programming of Computer Programs and its Applications*. Morgan Kaufman, San Francisco, dpunkt. verlag, Heidelberg, 1998.

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

[49] D. J. Montana, "Strongly typed genetic programming," *Evolutionary computation*, vol. 3, no. 2, pp. 199–230, 1995.

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

[51] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[52] F. Noorian, A. M. de Silva, and P. H. Leong, "Gramevol: Grammatical evolution in r," *Journal of Statistical Software*, vol. 71, no. i01, 2016.

[53] J. F. Miller, "Cartesian genetic programming," in *Cartesian Genetic Programming*, Springer, 2011, pp. 17–34.

[54] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *European Conference on Genetic Programming*, Springer, 2000, pp. 121–132.

[55] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[56] J. Clegg, J. A. Walker, and J. F. Miller, "A new crossover technique for cartesian genetic programming," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, 2007, pp. 1580–1587.

[57] A. Osyczka, "Multicriteria optimization for engineering design," in *Design optimization*, Elsevier, 1985, pp. 193–227.

[58] K. Deb, "Multi-objective evolutionary algorithms," in *Springer Handbook of Computational Intelligence*, Springer, 2015, pp. 995–1015.

[59] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[60] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[61] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, Springer, 2000, pp. 1–15.

[62] G. Folino and F. S. Pisani, "Combining ensemble of classifiers by using genetic programming for cyber security applications," in *European Conference on the Applications of Evolutionary Computation*, Springer, 2015, pp. 54–66.

[63] M. P. Sesmero, A. I. Ledezma, and A. Sanchis, "Generating ensembles of heterogeneous classifiers using stacked generalization," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 1, pp. 21–34, 2015.

[64] M. Crosbie and G. Spafford, "Applying genetic programming to intrusion detection," in *Working Notes for the AAAI Symposium on Genetic Programming*, Cambridge, MA: MIT Press, 1995, pp. 1–8.

[65] W. Lu and I. Traore, "Detecting new forms of network intrusion using genetic programming," *Computational intelligence*, vol. 20, no. 3, pp. 475–494, 2004.

[66] C. Yin, S. Tian, H. Huang, and J. He, "Applying genetic programming to evolve learned rules for network anomaly detection," in *International Conference on Natural Computation*, Springer, 2005, pp. 323–331.

[67] K. Faraoun and A. Boukelif, "Genetic programming approach for multi-category pattern classification applied to network intrusions detection," *International Journal of Computational Intelligence and Applications*, vol. 6, no. 01, pp. 77–99, 2006.

[68] A. Orfila, J. M. Estevez-Tapiador, and A. Ribagorda, "Evolving high-speed, easy-to-understand network intrusion detection rules with genetic programming," in *Workshops on Applications of Evolutionary Computation*, Springer, 2009, pp. 93–98.

[69] *Lbnl/icsi enterprise tracing project - project overview.* [Online]. Available: `http://www.icir.org/enterprise-tracing/` (visited on 11/08/2018).

[70] J. Blasco, A. Orfila, and A. Ribagorda, "Improving network intrusion detection by means of domain-aware genetic programming," in *International Conference on Availability, Reliability, and Security (ARES'10)*, IEEE, 2010, pp. 327–332.

[71] D. Wilson and D. Kaur, "Using grammatical evolution for evolving intrusion detection rules," *WSEAS Transactions on Systems*, vol. 6, no. 2, p. 346, 2007.

[72] G. Folino, C. Pizzuti, and G. Spezzano, "GP ensemble for distributed intrusion detection systems," *Pattern Recognition and Data Mining*, pp. 54–62, 2005.

[73] S. Picek, E. Hemberg, D. Jakobovic, and U.-M. O'Reilly, "One-class classification of low volume dos attacks with genetic programming," in *Genetic Programming Theory and Practice XV*, Springer, 2018, pp. 149–168.

[74] K. Badran and P. Rockett, "Multi-class pattern classification using single, multi-dimensional feature-space feature extraction evolved by multi-objective genetic programming and its application to network intrusion detection," *Genetic Programming and Evolvable Machines*, vol. 13, no. 1, pp. 33–63, 2012.

[75] J. Gómez, C. Gil, R. Baños, A. L. Márquez, F. G. Montoya, and M. Montoya, "A pareto-based multi-objective evolutionary algorithm for automatic rule generation in network intrusion detection systems," *Soft Computing*, vol. 17, no. 2, pp. 255–263, 2013.

[76] E. De la Hoz, E. de la Hoz, A. Ortiz, J. Ortega, and A. Martıénez-Álvarez, "Feature selection by multi-objective optimisation: Application to network anomaly detection by hierarchical self-organising maps," *Knowledge-Based Systems*, vol. 71, pp. 322–338, 2014.

[77] G. Kumar and K. Kumar, "A multi-objective genetic algorithm based approach for effective intrusion detection using neural networks," in *Intelligent Methods for Cyber Warfare*, Springer, 2015, pp. 173–200.

[78] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[79] S. Sen, J. A. Clark, and J. E. Tapiador, "Power-aware intrusion detection in mobile ad hoc networks," in *International Conference on Ad Hoc Networks*, Springer, 2009, pp. 224–239.

[80] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "Training genetic programming on half a million patterns: An example from anomaly detection," *IEEE transactions on evolutionary computation*, vol. 9, no. 3, pp. 225–239, 2005.

[81] A. Abraham and C. Grosan, "Evolving intrusion detection systems," in *Genetic systems programming*, Springer, 2006, pp. 57–79.

[82] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An evaluation framework for intrusion detection dataset," in *2016 International Conference on Information Science and Security (ICISS)*, IEEE, 2016, pp. 1–6.

[83] A. Ghourabi, T. Abbes, and A. Bouhoula, "Characterization of attacks collected from the deployment of web service honeypot," *Security and Communication Networks*, vol. 7, no. 2, pp. 338–351, 2014.

[84] F. Pouget and M. Dacier, "Honeypot-based forensics," in *AusCERT Asia Pacific Information Technology Security Conference*, 2004.

[85] U. Thakar, N. Dagdee, and S. Varma, "Pattern analysis and signature extraction for intrusion attacks on web services," *International Journal of Network Security & its Applications (IJNSA)*, vol. 2, no. 3, 2010.

[86] A. Ghourabi, T. Abbes, and A. Bouhoula, "Data analyzer based on data mining for honeypot router," in *International Conference on Computer Systems and Applications (AICCSA)*, IEEE, 2010, pp. 1–6.

[87] S. I. S. Center, *Sans isc: Demonstrating the value of your intrusion detection program and analysts*. [Online]. Available: `https://isc.sans.edu/diary.html?date=2012-09-02` (visited on 11/08/2018).

[88] Y. Park and J. Park, "Web application intrusion detection system for input validation attack," in *Third International Conference on Convergence and Hybrid Information Technology*, IEEE, 2008, pp. 498–504.

[89] D. Stevanovic, A. An, and N. Vlajic, "Feature evaluation for web crawler detection with data mining techniques," *Expert Systems with Applications*, vol. 39, no. 10, pp. 8707–8717, 2012.

[90] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for internet traffic classification," *IEEE Transactions on neural networks*, vol. 18, no. 1, pp. 223–239, 2007.

[91] F. Pouget, M. Dacier, and V. Pham, "On the advantages of deploying a large scale distributed honeypot platform," in *Proceedings of the E-Crime and Computer Evidence Conference*, 2005.

[92] E. Alata, M. Dacier, Y. Deswarte, M. Kaaâniche, K. Kortchinsky, V. Nicomette, V.-H. Pham, and F. Pouget, "Collection and analysis of attack data based on honeypots deployed on the internet," in *Quality of Protection*, Springer, 2006, pp. 79–91.

[93] A. Ghourabi, T. Abbes, and A. Bouhoula, "Behavior analysis of web service attacks," in *IFIP International Information Security Conference*, Springer, 2014, pp. 366–379.

[94] *Mit lincoln laboratory, darpa intrusion detection evaluation data sets*. [Online]. Available: `https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset` (visited on 09/03/2018).

[95] *Kdd cup 99 dataset*. [Online]. Available: `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html` (visited on 09/03/2018).

[96] *Argus*. [Online]. Available: https://qosient.com/argus/ (visited on 11/08/2018).

[97] *Flowcalc*. [Online]. Available: http://mutrics.iitis.pl/flowcalc (visited on 11/08/2018).

[98] G. Folino, F. S. Pisani, and P. Sabatino, "A distributed intrusion detection framework based on evolved specialized ensembles of classifiers," in *European Conference on the Applications of Evolutionary Computation*, Springer, 2016, pp. 315–331.

[99] M. B. Kursa and W. R. Rudnicki, "Feature selection with the boruta package," *J Stat Softw*, vol. 36, no. 11, pp. 1–13, 2010.

[100] S. Khanchi, A. Vahdat, M. I. Heywood, and A. N. Zincir-Heywood, "On botnet detection with genetic programming under streaming data label budgets and class imbalance," *Swarm and evolutionary computation*, vol. 39, pp. 123–140, 2018.

[101] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, ACM, 2011, pp. 29–36.

[102] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Intelligent rule-based phishing websites classification," *IET Information Security*, vol. 8, no. 3, pp. 153–160, 2014.

[103] R. M. Mohammad, L. McCluskey, and F. Thabtah, *UCI machine learning repository: Phishing websites data set*, 2015. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/phishing+websites.

[104] *Phishtank | join the fight against phishing*. [Online]. Available: https://www.phishtank.com/ (visited on 11/08/2018).

[105] *Phishing scams and spoof emails at millersmiles.co.uk*. [Online]. Available: http://www.millersmiles.co.uk/ (visited on 11/08/2018).

[106] *Yahoo search - web search*. [Online]. Available: https://uk.search.yahoo.com/?guccounter=1 (visited on 11/08/2018).

[107] *Starting point directory*. [Online]. Available: http://www.stpt.com/directory/ (visited on 11/08/2018).

[108] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems," in *Military Communications and Information Systems Conference (MilCIS)*, IEEE, 2015, pp. 1–6.

[109] *Cicflowmeter: A network traffic biflow generator and analyzer*. [Online]. Available: `http://netflowmeter.ca/` (visited on 11/02/2019).

[110] Y. Zhang and S. Bhattacharyya, "Genetic programming in classifying large-scale data: An ensemble method," *Information Sciences*, vol. 163, no. 1-3, pp. 85–101, 2004.

[111] T. A. Pham, Q. U. Nguyen, and X. H. Nguyen, "Phishing attacks detection using genetic programming," in *Knowledge and Systems Engineering*, Springer, 2014, pp. 185–195.

[112] T. A. Le, T. H. Chu, Q. U. Nguyen, and X. H. Nguyen, "Malware detection using genetic programming," in *Computational Intelligence for Security and Defense Applications (CISDA), 2014 Seventh IEEE Symposium on*, IEEE, 2014, pp. 1–6.

[113] S. Luke, *ECJ evolutionary computation library*, 1998. [Online]. Available: `https://cs.gmu.edu/~eclab/projects/ecj/`.

[114] W. R. Inc., *Mathematica, Version 11.3*, Champaign, IL, 2018.

[115] F. Noorian, A. M. de Silva, and P. H. Leong, "Gramevol: Grammatical evolution in r," *Journal of Statistical Software*, 2015.

[116] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[117] F. J. Ordóñez, A. Ledezma, and A. Sanchis, "Genetic approach for optimizing ensembles of classifiers," in *FLAIRS conference*, 2008, pp. 89–94.

[118] H2O.ai, *H2o: R interface for h2o*, Package version 3.18.0.2, Mar. 2018. [Online]. Available: `http://www.h2o.ai`.

[119] D. Cook, *Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI*. " O'Reilly Media, Inc.", 2016.

[120] M. Landry, S. Aiello, E. Eckstrand, A. Fu, and P. Aboyoun, *Machine learning with r and h2o*, Jun. 2018. [Online]. Available: `http://h2o.ai/resources/`.

[121] N. Acosta-Mendoza, A. Morales-Reyes, H. J. Escalante, and A. Gago-Alonso, "Learning to assemble classifiers via genetic programming," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 28, no. 07, p. 1 460 005, 2014.

[122] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, "Multiobjective genetic programming: Reducing bloat using spea2," in *Proceedings of the Congress on Evolutionary Computation*, 2001, pp. 536–543.

[123] L. Shao, L. Liu, and X. Li, "Feature learning for image classification via multiobjective genetic programming," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 7, pp. 1359–1371, 2014.

[124] P. Roocks, "Computing pareto frontiers and database preferences with the rpref package," *RJ*, vol. 8, no. 2, pp. 393–404, 2016.

[125] Y. Zhu, J. Liang, J. Chen, and Z. Ming, "An improved nsga-iii algorithm for feature selection used in intrusion detection," *Knowledge-Based Systems*, vol. 116, pp. 74–85, 2017.

[126] H. Lu, K. Zheng, B. Liu, X. Zhang, and Y. Liu, "A memory-efficient parallel string matching architecture for high-speed intrusion detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1793–1804, 2006.

[127] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," in *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE, vol. 4, 2004, pp. 2628–2639.

[128] M. Stehlik, A. Saleh, A. Stetsko, and V. Matyas, "Multi-objective optimization of intrusion detection systems for wireless sensor networks.," in *ECAL*, 2013, pp. 569–576.

[129] J. Ellis, *Java agent for memory measurements*. [Online]. Available: `https://github.com/jbellis/jamm` (visited on 10/02/2018).

[130] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors," in *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2008, pp. 116–134.

[131] J. B. Cabrera, J. Gosar, W. Lee, and R. K. Mehra, "On the statistical distribution of processing times in network intrusion detection," in *43rd IEEE Conference on Decision and Control*, IEEE, vol. 1, 2004, pp. 75–80.

[132] R. Kohavi and D. H. Wolpert, "Bias plus variance decomposition for zero-one loss functions," in *ICML*, vol. 96, 1996, pp. 275–83.