Applied Soft Computing 33 (2015) 348–359

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Evolving statistical rulesets for network intrusion detection

Samaneh Rastegari*, Philip Hingston, Chiou-Peng Lam

School of Computer and Security Science, Edith Cowan University, WA 6050, Australia

ARTICLE INFO

Article history:
Received 14 August 2014
Received in revised form 22 March 2015
Accepted 18 April 2015
Available online 28 April 2015

Keywords:
Intrusion detection
DARPA
NSL-KDD
Genetic algorithm
Interval rule-based

ABSTRACT

Security threats against computer networks and the Internet have emerged as a major and increasing area of concern for end-users trying to protect their valuable information and resources from intrusive attacks. Due to the amount of data to be analysed and the similarities between attack and normal traffic patterns, intrusion detection is considered a complex real world problem. In this paper, we propose a solution that uses a genetic algorithm to evolve a set of simple, interval-based rules based on statistical, continuous-valued input data. Several innovations in the genetic algorithm work to keep the ruleset small. We first tune the proposed system using a synthetic data. We then evaluate our system against more complex synthetic data with characteristics associated with network intrusions, the NSL-KDD benchmark dataset, and another dataset constructed based on MIT Lincoln Laboratory normal traffic and the low-rate DDoS attack scenario from CAIDA. This new approach provides a very compact set of simple, human-readable rules with strongly competitive detection performance in comparison to other machine learning techniques.

1. Introduction

Network intrusion is defined as any unauthorised attempt to access, falsify, change or destroy information in order to make a system unreliable [1]. Nowadays, in almost all large-scale IT infrastructure, we need an efficient intrusion detection system to secure our networks against existing and future attacks. Despite improvements in protection and detection mechanisms, it is still impossible to secure computer networks completely. To compete in the arms race against highly sophisticated and varying types of network intrusions, traditional intrusion prevention techniques such as firewalls, access control or encryption are not enough to fully protect networks. Therefore, as argued in [2], we need to consider state-of-the-art computational intelligence technologies, which are able to identify existing intrusions and adapt themselves to new unseen attack strategies to update and enhance our security systems. By defining behavioural characteristics of network traffic, we are able to train a detection system to trigger alarms in the case of anomalous behaviours in the system. We believe that an appropriate statistical-based technique, which has the ability to learn the expected behaviour of the system from observations, combined with a rule-based machine learning method, providing robustness, flexibility, adaptability and scalability [3], can address many outstanding issues in network intrusion detection systems (NIDSs), such as dependency on the signature of network packets, which makes them unable to detect novel attacks, and generating protocol-based rules, which only tackle one specific problem at a time.

In general, a NIDS operates by monitoring network traffic and providing a view of malicious activities, and finally giving out alerts, which inform network administrators and facilitate the process of reaction. There are two broad groups of intrusion detection systems (IDSs): signature-based and anomaly-based. Mechanisms that combine these two systems are known as hybrid detection systems. Signature-based systems use a database of known patterns to identify intrusions, whereas anomaly-based systems detect intrusions based on deviations from normal network behaviours and can generate alarms for new or modified attacks. Signature-based mechanisms usually work faster and produce less false positives, but they need prior knowledge of intrusions and consequently, in the case of novel attacks, a network system is left vulnerable until the signature database is updated.

There are a number of different approaches used in each category, including threshold detection, statistical measurement, rule-based methods and other artificial intelligence mechanisms. Among these, artificial intelligence methods have attracted considerable interest from the research community. Artificial neural networks (ANNs), fuzzy logic (FL), genetic algorithms (GAs), genetic programming (GP), swarm intelligence (SI) and artificial immune systems (AISs) are examples of computational intelligence approaches that have been used. These techniques with learning

* Corresponding author. Tel.: +61 402533600.

http://dx.doi.org/10.1016/j.asoc.2015.04.041
1568-4946/

and adaptive capabilities have often been used together with rule-based methods to acquire knowledge of normal and abnormal behaviour. Classical machine learning methods such as decision trees and rule induction have also been used [2,4,5]. The knowledge extracted from training data is often in the form of *if–then* rules. Among these techniques, genetic algorithms show promising results for learning rule-like patterns. The advantages of genetic algorithms as a rule discovery method include the ability to explore a large search space efficiently, and being able to use arbitrary fitness functions in the search. For example, some rule discovery methods generate large, unwieldy rulesets. Carefully designed fitness functions can be used to tackle this problem [6].

One of the major issues that researchers face in the process of developing an IDS is that the detection technique should be designed so that it is adaptable to a wide range of network environments without a lot of manual tuning [7]. If we can develop a generic IDS that is not specialised to a particular pattern of traffic, then we will have taken an important step toward developing an effective and robust detection method that can be used against different existing attack tools and future malicious attacks, and which can be deployed at any point of network infrastructure. Previous research has shown that, although new improved attack tools are powerful enough to generate network packets with different characteristics, they distort statistical features of network traffic in the process [7]. Therefore, we can use statistical measures to differentiate between normal and anomalous traffic.

In this paper, we present a GA-based approach for discovering rulesets to solve network intrusion detection problems. The approach provides good detection performance, while identifying very concise rulesets that are easily understood. To provide contrast with other machine learning techniques, we compare our results to those obtained using decision trees, rule induction, k-nearest neighbour (kNN) from the category of non-GA-based algorithms and Genetic Algorithm based Classifier System with Adaptive discretisation intervals (GASSIST-ADI) [8] and Memetic Pittsburgh Learning Classifier System (MPLCS) [9] from the category of GA-based rule learning techniques. We used three sources of data for performance evaluations: synthetic data, the NSL-KDD dataset and a combined DARPA/CAIDA dataset. The NSL-KDD dataset is a new improved version of the KDD99 dataset [10,11], a widely used dataset for NIDS studies. The NSL-KDD dataset has solved some of the issues in the original dataset, such as redundant records in the training dataset. Since the DARPA/Lincoln Labs packet traces and the KDD99 dataset derived from them are now more than a decade old, they are not considered by some as not reflecting contemporary attacks [12]. Considering this issue, we used the MIT Lincoln Laboratory Scenario (attack-free) inside tcpdump dataset [13] as the normal network traffic and the low-rate DDoS attack scenario from DAIDA DDoS 2007 [14] as the attack traffic to evaluate the proposed algorithm on a dataset with newer attack traces, an approach also used by [15–17].

The rest of the paper is organised as follows. Section 2 reviews related work on the use of machine learning techniques for IDSs by focusing on rule-based algorithms. Section 3 describes our approach in detail. In Section 4, preliminary results from tuning of our algorithm are followed by results obtained from evaluation of the proposed system on complex synthetic data. Next, in Section 5, we apply our proposed approach to network intrusion detection. Finally, Section 6 provides conclusions and discusses future work.

2. Related work

In the literature, there are many machine learning approaches for intrusion detection systems, including computational intelligence methods such as fuzzy systems, neural networks, support vector machines, evolutionary algorithms and swarm intelligence, as well as classical machine learning methods such as decision trees, k-nearest neighbour, naïve Bayes classifiers and various hybrids. These techniques in particular are developed for classification of incoming network traffic and detecting malicious attacks. Recent reviews of existing studies of intrusion detection by machine learning techniques can be found in [2]. One rule induction algorithm that has been recommended for use in intrusion detection is RIPPER [18], due to its ability to generate concise, understandable classification rules from large, noisy datasets [5]. Lee and Stolfo described a framework for building intrusion detection models based on RIPPER ([19,20]). It is often used in comparative studies such as [21].

In recent times, classification rules have been combined with evolutionary algorithms such as GAs [22] to search for optimised sets of rules, especially when the size of the search space is very large [23]. GAs are a search method "based on the mechanics of natural selection and natural genetics" [24]. In contrast to traditional methods, GAs have the advantage of performing a global search using a population of individuals. In the evolutionary process, the goodness of an individual is evaluated by a function, which is called the *fitness function* [25]. In a classification problem, where an individual is represented by a rule in the form of an *if–then* statement, the fitness function encourages rules which accurately classify the training instances.

In [26], a GA was utilised as a search method to discover a comprehensive set of rules from network audit data for intrusion detection. Each rule in their system includes seven features extracted from the 1998 DARPA dataset [27]. The first six features (duration, protocol, source_port, destination_port, source_IP and destination_IP) are checked with their corresponding values and connected using the AND operator to form the condition part and the last feature (Attack_name) is the outcome of a rule. Here, each rule is encoded using a fixed length vector and a support-confidence framework [28] is used as the fitness function to evaluate each rule in the population. To avoid over-fitting, the number of generations and the number of most-fit rules to use were found by trial and error. In [22], a GA was used to evolve simple rules in the form: *if* {*condition*} *then* {*act*} for network traffic. The condition part contains nine attributes of network connections. Each chromosome has 57 genes and the fitness of each chromosome is computed separately. This means a rule will receive a bonus if it finds an anomalous behaviour or a penalty if it matches a normal connection. Thus, there is a chance of redundant rules in the final optimised ruleset, which cover overlapping regions of search space. The evolution process starts with a population of randomly generated rules and continues with selection, crossover and mutation operators to find an optimised rule set. Finally, these rules will be added into the rule database used by the IDS. The DARPA dataset in tcpdump binary format was used for testing the system. In this paper, the authors used two statistical features of network traffic to overcome the problem associated with approaches using only categorical features of raw data. In [29], a genetic algorithm is used to generate optimised fuzzy rules for classification of training instances. A population of encoded fuzzy rules in the form of *if–then* statements is initially generated, and then evaluated based on a number of optimisation criteria which are combined into a single scalar fitness value. One of the contributions in their work is the use of a boosting mechanism, which helps the whole system to consider cooperation of fuzzy rules during the rule generation process. This mechanism weights the training records based on their difficulty. By assigning larger weights to those records which are not classified by the current rules, the system improves toward generation of new rules that are able to complete the existing set cooperatively. This will help to overcome the problem of having too many rules, including redundant ones, in the final optimised

Table 1
A summary of the reviewed machine learning (ML) techniques.

Method	ML category	Evaluation Metric	Evaluation Style	Individual Representation	Type of GA
[19,20]	ML non-GA rule-based	ROC curve	–	–	–
[26]	GA-based rule learning	Fitness value: support-confidence framework	–	Fixed length vector of different data types	Generational
[22]	GA-based rule learning	Fitness value	–	Fixed length string	Generational
[29]	Genetic Fuzzy Systems	Fitness value combined with a boosting algorithm	–	Continuous valued vector	Generational
[31]	LCS (UCS-supervised learning classifier system)	Fitness value: a direct function of accuracy	Michigan	Continuous valued interval-based	Steady State
[8]	LCS	Fitness value: MDL	Pittsburgh	adaptive discretisation intervals	Generational
Proposed approach	genetic interval rule-based	Two stage evaluation using fitness and performance functions	Michigan	Continuous valued interval based with regulatory genes which only holds relevant attributes	Generational

ruleset. The algorithm was tested on different datasets, including a synthetic dataset, and showed similar classification accuracies to other pattern recognition methods.

Another similar rule-based method used for IDSs is a Learning Classifier System (LCS). LCS was first introduced by John H. Holland [30] as a genetic-based machine learning technique. The GA module searches a large space by first randomly generating a population of individuals and then selecting the best and producing new individuals by mutation and crossover operators, iteratively. To evaluate the quality of individuals, LCSs utilise reinforcement learning or supervised learning techniques. In [31], a supervised signature learning classifier was used to evolve a set of classifiers where each classifier is in the form of a simple rule. Two main metrics were used to evaluate the performance of rules: Accuracy and Fitness. Accuracy is the ratio between the number of times a classifier was successful in detecting an instance and the number of times it has been fired. Fitness is then calculated as a function of accuracy. During the training phase, a set of matching classifiers for an input instance is built from the population of individuals. Then based on their updated performance parameters, this system finds the classifiers which predict the instance class (label) correctly. Finally, a GA is applied to the selected classifiers to form the next generation population. One problem of LCSs is the number of rules generated by the system, which makes the resulting ruleset less transparent, and is an issue for time critical applications. To solve this problem, the authors proposed two generalisation operators to modify rule boundaries, which reduces the overlapping regions covered by the individuals.

GASSIST-ADI [8] is another example of LCSs belonging to the pittsburgh style category, which uses an adaptive discretisation intervals (ADI) rule representation. In GASSIST-ADI, the continuous space is discretised into fixed intervals for developing rules. GASSIST-ADI uses a fitness function based on the Minimum Description Length (MDL) principle to make optimal trade-off between complexity and accuracy of the rulesets [32]. An example of application of this algorithm in Detection of SIP based Flooding Attacks can be found in [33].

From the analysis of the reviewed approaches, GAs have been used as a tool for generating knowledge for a rule-based IDS in all of them except for [19,20]. Each rule in these systems is considered a classifier. The goal is to find and add a set of the most fit classifiers to the rule database in an IDS to provide an acceptable detection rate. We believe that simply finding and adding the best individual rules to the ruleset is not enough, since those rules may not cooperatively work well together. As a result, the learning system may end up generating a set of rules, where each of them is a good classifier individually but one specific rule might cover a region of search space that has been covered by another rule. So, we will have numerous rules for each class. Thus, in this paper, we focus on the degree of cooperation of the rules when they are evaluated by the GA, which results in a more concise set of rules.

Table 1 provides a summary of the reviewed techniques for IDSs. We included our proposed technique in this table to highlight its characteristics in comparison to some existing methods.

Another advantage of our proposed IDS is that it does not depend on the categorical features of packet headers (e.g. source IP address). That differentiates our system from signature based IDS, and makes it more generic for various environments. We are using the statistical features of network traffic to detect any suspicious behaviour in the pattern of traffic. This helps us in detection of unknown future attacks as well. For the representation of the chromosomes, we borrow the idea of the interval predicate representation introduced by Wilson [34] and similar constraint-based detectors proposed in [35]. In the former, a classifier has six interval predicates, $int_i = (c_i, s_i)$, where c_i and s_i are reals. A classifier matches an input x if and only if $c_i - s_i <= x_i < c_i + s_i$, for all x_i. Consequently, the number of numerical values in the condition part of a rule is twice the number of components in x. In the latter, a classifier is in the form of ($lb_0..ub_0$, $lb_1..ub_1$, $lb_2..ub_2$, $lb_3..ub_3$, $lb_{port}..ub_{port}$, src), where the first 4 intervals (between 0 and 255) represent a set of IP addresses, the fifth one represents the range of port numbers (between 0 and 69 categories), and finally when src is 0, it indicates that the detector should be used on incoming traffic and when it is 1, the detector applies on outgoing traffic. In this work, an *any-3-intervals* matching rule is used to determine a match between a packet and a detector.

In our proposed framework, a new representation of individuals is combined with a well-formulated fitness function for the evaluation of each rule in the system. A generic performance function is also introduced to generalise the proposed model to work on different problems. This performance function operates on a higher level to evolve the rules, cooperatively. In the next section, we explain the proposed approach and the implementation of our experiments in detail.

3. Proposed GA-based approach

In this section, we will describe our proposed GA-based approach, which generates optimised rulesets through a learning stage and then uses the generated rules for classifying the data

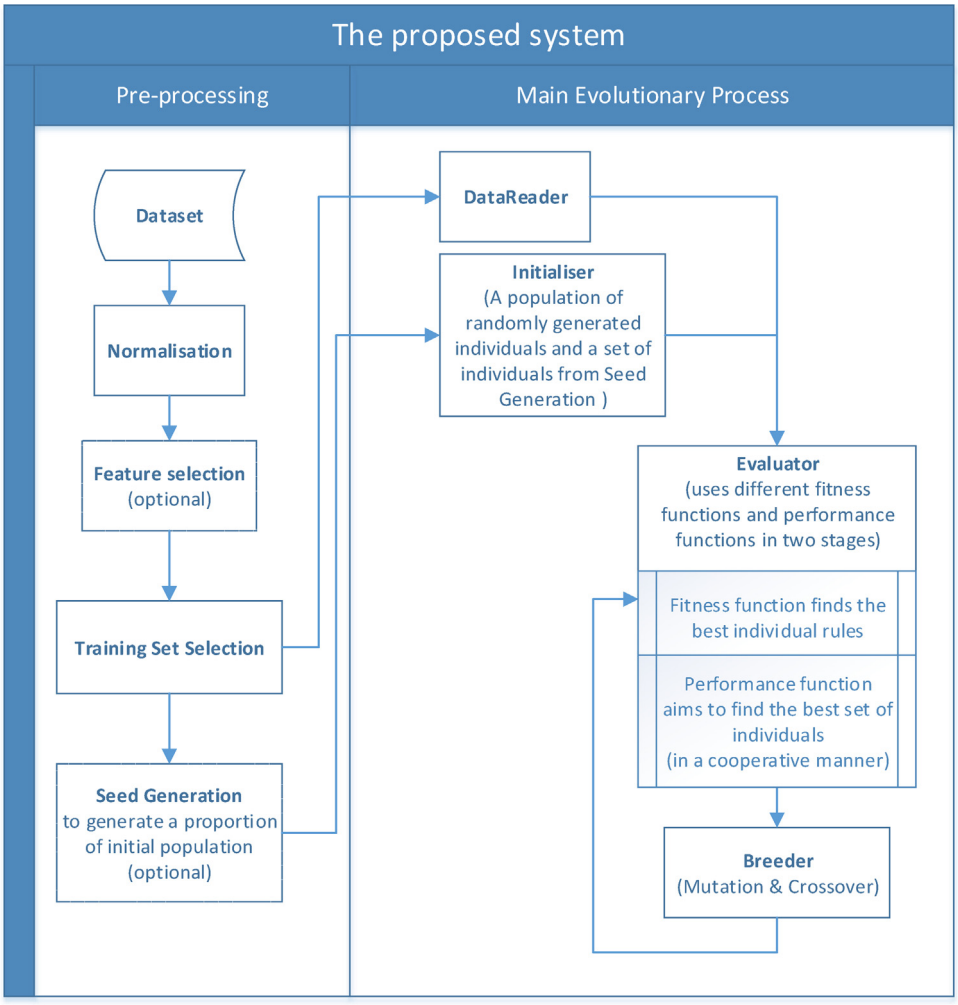

Fig. 1. The components of the proposed system. Preprocessing consists of the familiar normalisation, and optionally a feature selection step. The preprocessing stage provides normal and attack records to train and test the evolving rule-based classifier, and optionally can provide seed rules for the initial population of rules. The evaluation stage of the evolutionary algorithm includes a traditional fitness function to evaluate individual rules, as well as a higher level function that is used to select sets of rules that work well together.

points. The GA includes a number of innovations, which contribute to the performance of the system. These include a chromosome structure that allows for rules with variable numbers of features, a fitness function designed to reward cooperation between rules, and a mechanism that adaptively determines the amount of elitism in the selection process. All of these help to produce concise rulesets. The architecture of the proposed system is presented in Fig. 1.

There are two processing stages in our system: pre-processing and the main evolutionary process. In the former, we focus on the preparation of data for the system, which includes normalisation, feature selection (if needed), model validation, and seed selection (if needed). We used standard [0 1] normalisation in order to equalize the influence of features on the training results. For some of the experiments in this paper, we had two extra processes of feature selection and seed selection. The seed selection component generates some rules, which are derived from the training data, to be used as a proportion of the initial population in the next stage. A model validation method is the last component of the preprocessing stage, which is used to create training and test sets for performance evaluation (in our experiments with synthetic data, we used 10-fold cross-validation).

Next, the main evolutionary process of our proposed system starts, with two components to read the preprocessed data and to initialise the population of individuals (each individual represents a rule). The initialiser may use the output of the seed selection process to initialise the population. If the number of individuals provided by the seed selection process is less than the size of the population, the initialiser will randomly generate the rest of the individuals in the population. Then, for a number of generations, the evaluator and breeder will operate to select a set of fit individuals from the population and to apply genetic operators (mutation and crossover) to each individual. Selection is a combination of *elitism*, in which a number of the fittest individuals are copied directly to the next generation, followed by *stochastic universal sampling* with mutation and crossover for the rest of the population. In the following sections, more details of the evolutionary process will be explained.

3.1. Individual representation

The basic structure of a detector rule in our system is as follows:

Rule: if cond, then anomaly ,where $cond = x_1 \in [lowerbound_1, upperbound_1]$ and... and

$x_n \in [lowerbound_n, upperbound_n]$;

for an instance vector like: $(x_1,...,x_n)$, where n is the number of active features in this rule. Different rules can have different sets of active features.

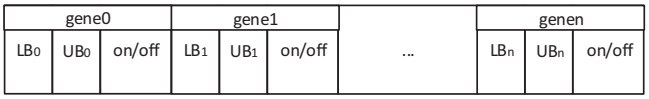

Fig. 2. The structure of a chromosome in our proposed system

The chromosome structure we use to represent a rule is as shown in Fig. 2. It contains one gene for each feature. Depending on the number of features in our dataset, the system can be customised by increasing or decreasing the length of the chromosome. Each gene has 3 components: LB_i, UB_i are real values between 0 and 1, and a binary on/off value signifies whether the feature is active in the rule. LB_i is mapped to a value in the range of the feature using the equation

$$\text{lowerbound}_i = \min + LB_i \times (\max - \min)$$

where *min* and *max* are the minimum and maximum values for that feature. The upper bound for this rule condition is calculated as

$$\text{upperbound}_i = \text{lowerbound}_i + UB_i \times (\max - \text{lowerbound}_i)$$

The binary *regulatory* value for each feature in the chromosome enhances the evolutionary algorithm by allowing it to find the required features to solve the problem. According to Oxford Dictionary of Biology [36], *regulatory genes* are defined as "genes that control development by regulating the expression of structural genes responsible for the formation of body components. They encode transcription factors, which interact with regulatory sites of other genes causing activation or repression of developmental pathways". In [37], the authors explored the impact of regulatory genes on the behaviour of an evolutionary algorithm. They added a binary regulator value for each position in the original linear representation, which enables deletion or insertion of genes when needed. As a result, during the evolution process, only those items whose matching regulatory value is "on" will be used. The results showed that the new gene regulatory representation enhances the evolutionary algorithm by finding new hills in the adaptive landscape.

If seed selection is used, a random subset of training examples is extracted from the training set, and for each, a rule is constructed using a small interval around each feature.

The mutation operator is different depending on which component of the gene is to be mutated. The upper and lower bound values are mutated by uniform mutation, while the regulatory value is mutated by bit-flipping. We used two-point crossover, with the crossover points restricted to the boundaries between genes.

3.1.1. Michigan versus Pittsburgh approach

In a GA, rules can be encoded in the population of individuals (chromosomes) using two approaches: Michigan and Pittsburgh. In the Michigan approach, an individual represents a single rule, while in the Pittsburgh approach, each individual represents a set of rules. In the former, only good cooperation of individuals can solve the problem whereas, in the latter, each individual is a good classifier by itself.

Our system is based on a Michigan style, where we evaluate each individual in cooperation with other individuals and find the best set of rules. Any input vector that is classified as anomalous by any rule in the set is considered as anomalous. That means each individual is a good classifier for a region of search space. Therefore, an optimal solution for the classification problem is represented by a set of rules with the best combined coverage. This is achieved by the design of the fitness function, which is described next.

3.2. Fitness function and performance function

The fitness of a rule in our system is defined by considering the following factors:

- The number of anomalies correctly detected.
- The number of errors (i.e. the number of normal instances incorrectly detected as anomalies)
- The overlapping with other rules; we give a higher score to the rules that detect anomalies which were not identified by other rules.
- A weighting on the scores and errors to control the balance between them and overcome the influence of imbalanced datasets on the results.

Based on each factor, we investigated the following three fitness functions and compared the results.

$$fitnessfunction(1) = no.ofinstancescorrectlydetected - no.oferrors \tag{1}$$

$$fitnessfunction(2) = (2p-1) * no.ofinstancescorrectlydetectedby1rule + (2p-2) * no.ofinstancescorrectlydetectedby2rules + \cdots + (2p-p) * no.ofinstancescorrectlydetectedbyallrules - (p) * \left(\frac{no.ofabnormalinstances}{no.ofnormalinstances}\right) * no.oferrors \tag{2}$$

$$fitnessfunction(3) = (p) * no.ofinstancescorrectlydetectedby1rule + (p-1) * no.ofinstancescorrectlydetectedby2rules + \cdots + (1) * no.ofinstancescorrectlydetectedbyallrules - (p) * \left(\frac{no.ofabnormalinstances}{no.ofnormalinstances}\right) * no.oferrors \tag{3}$$

where p is the number of individuals (rules) in the population.

The first fitness function is a naïve one, included as a baseline. It rewards correct classifications made by an individual rule and penalises incorrect ones, but does not take account of class imbalance, or overall coverage in combination with other rules. The second and third fitness functions both include an adjustment to account for class imbalance (the term that is used to weight errors). This is important for intrusion detection because training data typically has many more normal instances than anomalous ones. Both functions reward rules more for correctly identifying anomalies that fewer other rules have identified – moreso with fitness function 2 than with fitness function 3.

Using the equations given above, we measure the fitness of an individual rule among others in each generation. Although the fitness value indicates the quality of a rule, it is not enough for making a decision on which rules are able to cooperatively cover the search space. Therefore, we use a performance function to evaluate the performance of a group of rules for detection. This two-stage evaluation process derives a set of classification rules from the provided dataset.

Given a population of rules sorted by fitness, we evaluate, for each n, the combined performance of the first n rules. We then choose the value of n that gives the best performance, and the top n rules in the final generation is designated as the solution ruleset evolved by that run of the GA. As n is increased, the true positive

rate can only improve, but at some point, rules with too many false positives will be included, and combined performance will drop. It should be noted that this procedure may not choose an optimal subset of rules, but it is fast (with complexity linear in the population size), and in practice works well. For example, it might sometimes be possible to omit some of the top n rules and obtain better performance, but this should be a rare occurrence.

To evaluate a classifier, a confusion matrix, which includes true positive, true negative, false positive, and false negative rates is calculated and usually a classifier aims for high true positive and true negative rates and low false positive and false negative rates. Considering the evaluation methods of classifiers in both balanced and imbalanced domains, we used a metric called *g-performance* for the performance function (Eq. (4)). In [38], the *g-performance* is calculated by the geometric mean of the accuracies on positive and negative examples ($g = \sqrt{a^+ * a^-}$) to avoid the poor behaviour of some learners under the circumstances of having imbalanced datasets. Similarly this metric was used in [39,40] to measure the performance of classification over imbalanced datasets. Evolving the system using this measure allows the classifier to maximise the true positive and true negative rates at the same time.

$$\text{g} - \text{performance} = \sqrt{TP_{rate} * TN_{rate}} \tag{4}$$

, where TP_{rate} is the percentage of true positive cases correctly classified as positive, and TN_{rate} is the percentage of true negative cases correctly classified as negative.

This describes the fitness and performance functions used in the present study, but depending on the problem domain and the characteristics of the dataset, the fitness and performance functions can be customised in the proposed system in order to achieve the best results.

3.3. Adaptive elitist selection

Since it is important that our evolving sets of rules work together, we introduce an adaptive elitism mechanism, in which we adaptively adjust the number of elites copied into each new generation. This ensures that cooperating rules are kept together and not lost from one generation to the next. This also means that we do not have to select some arbitrary number of elites a priori. Thus, in each generation, the top n rules are designated elite, where n is chosen to maximise the performance measure, as described above.

4. Algorithm performance and tuning

In this section, we report the results of experiments in which we compare the performance of our algorithm with that of a representative set of existing classification algorithms that have often been used for intrusion detection. We will see that the accuracy of our system is comparable with these existing algorithms, is robust and reliable, and generates small, easily understood rulesets. For initial tuning, we use "synthetic" dataset that can easily be constructed and manipulated to create classification problems with desired features. We first defined a medium-size classification problem as presented in Fig. 3. Using three features, the generated data can be easily illustrated for the preliminary experiments. This is not possible for higher dimension data used in subsequent experiments.

In this problem, 500 data points for "normal" records (green area) and two clusters of 250 points for "anomalous" records (red area) with some overlap with the green region, are generated to test the accuracy of the system in detecting the anomalous records. The overlap between the normal and anomalous clusters is provided to simulate the similarities between normal and abnormal behaviours in real-world problems. The rules used to generate the data for this problem are as follows:

Green: *if* F1 ∈ [0, 0.3] *and* F2 ∈ [0.3, 0.6] *and* F3 ∈ [0, 0.2] *then normal*

Red: *if* F1 ∈ [0, 0.3] *and* F2 ∈ [0, 0.2] *and* F3 ∈ [0.4, 0.8] *then anomaly*

if F1 ∈ [0, 0.1] *and* F2 ∈ [0.5, 0.9] *and* F3 ∈ [0, 0.3] *then anomaly*

To provide a basis for comparison, we selected an instance-based algorithm (k-nearest neighbours), a decision tree method (a version of C4.5), a non-GA-based rule induction method (RIPPER) and two GA-based learning classifiers (GASSIST-ADI and MPLCS). For the first three classifiers, we used the open source Java-based package Weka, developed by researchers at the University of Waikato in New Zealand [41]. The Weka version of the C4.5 algorithm, known as J48, an implementation of RIPPER called JRip, and kNN, a linear nearest neighbours search algorithm, were used. The other two classifiers, GASSIST-ADI and MPLCS, are provided by the KEEL software [42].

4.1. Tuning using a medium-sized synthetic problem

To tune the proposed algorithm using the medium-sized problem, for each fitness function, we investigated different combinations of mutation probabilities (0.1, 0.3 and 0.5) and population sizes (50, 100 and 200). To keep the total number of evaluations the same (15000 evaluations) in all the experiments, when we increase the population size, the number of generations will be decreased. For synthetic problems (this problem and three more complex problems in Section 4.2), the seed selection module was not needed and the initial population of individuals was randomly generated.

For each of the algorithms used for comparison, we also carried out parameter tuning. For J48, confidence threshold and minimum number of instances per leaf were tuned; for kNN it was the number of neighbours; for JRip it was the number of runs (Weka's inbuilt cross-validated parameter selection facility was used for these). Additionally, for GASSIST-ADI and MPLCS, the crossover probability was tuned.

We used the results from this test problem to decide on the best fitness function and parameter choices for our evolutionary algorithm, which we then used in all later experiments.

The performance of our evolutionary algorithm using three different fitness functions is presented in Figs. 4–6 along with the five comparison algorithms. In each figure, we include box-and-whiskers plots for J48, JRip, kNN, GASSIST-ADI and MPLCS as well as for our proposed system, with each different combination of mutation rate and population size. The performance figure used is the g-performance metric (Eq. (4)). For J48, JRip and kNN, we performed standard 10-fold cross-validation testing, and the plot shows minimum, lower quartile, median, upper quartile, and maximum performance values over the 10 folds. For GASSIST-ADI, MPLCS and our proposed system, we ran the GA 3 times for each of the 10 folds, so the results summarise 30 runs in all (and thus the maximum and minimum values can be expected to show a slightly wider spread than for the other classifiers).

Based on the results, we concluded that the performance of our system using fitness function (3) is the most reliable among the three fitness functions. The variation of performance values in Fig. 6 is less compared to Figs. 4 and 5. As can be seen in Fig. 6, the performance range for our algorithm is similar to that of J48, JRip, kNN, GASSIST-ADI and MPLCS.

Since the variation of mutation probability, population size and number of generations is not greatly affecting the performance of our system using fitness function (3), we fix the system with the following parameters for the remaining experiments:

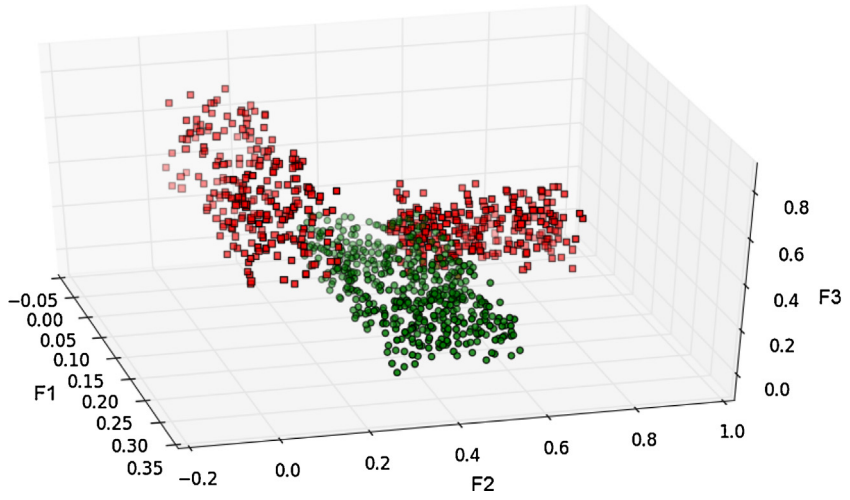

Fig. 3. A medium-sized problem. Red (darker) data points represent "anomolous" records, and green (lighter) points are "normal".

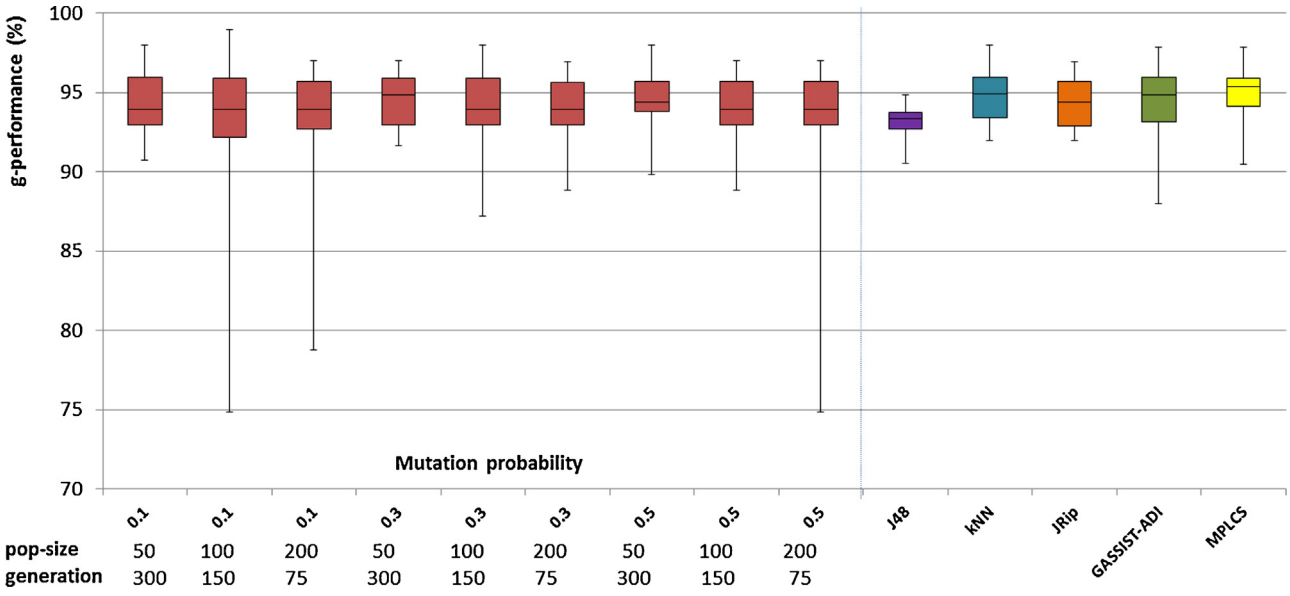

Fig. 4. Performance of the algorithm using fitness function (1). The performance value is the g-performance metric. Median performance for our algorithm is in the same range (94–95%) as for the comparison algorithms, but with this fitness function, the range is quite wide, and dependent on the choice of mutation rate and population size.

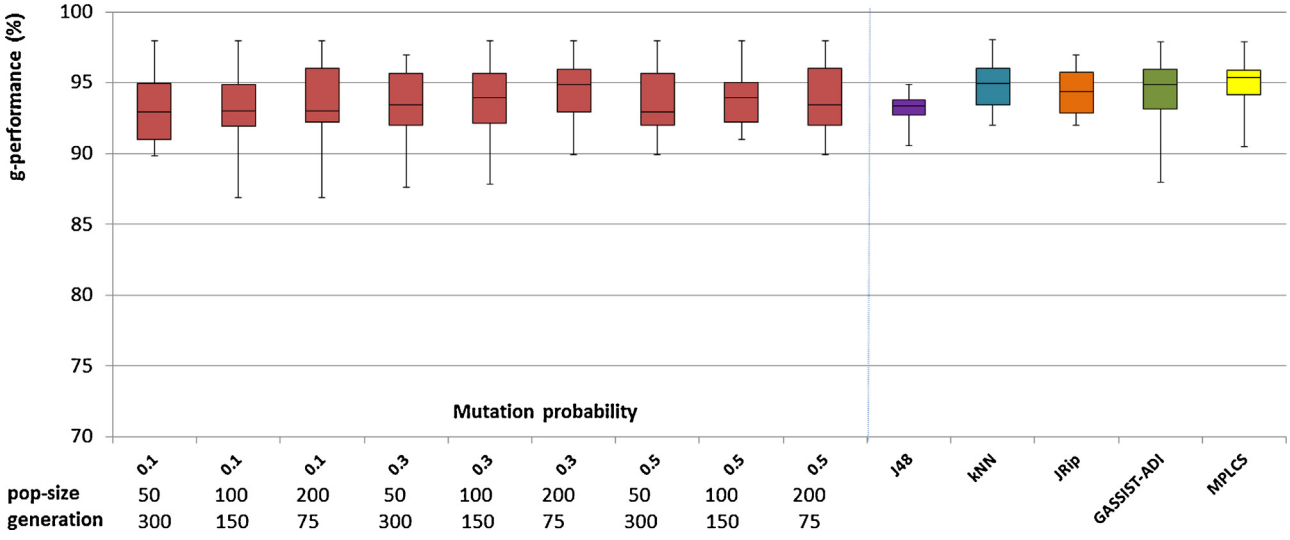

Fig. 5. Performance of the algorithm using fitness function (2). Median performance is similar to that for fitness function (1) but the variability is less, with minima between about 87% and 91%, depending on mutation rate and population size.

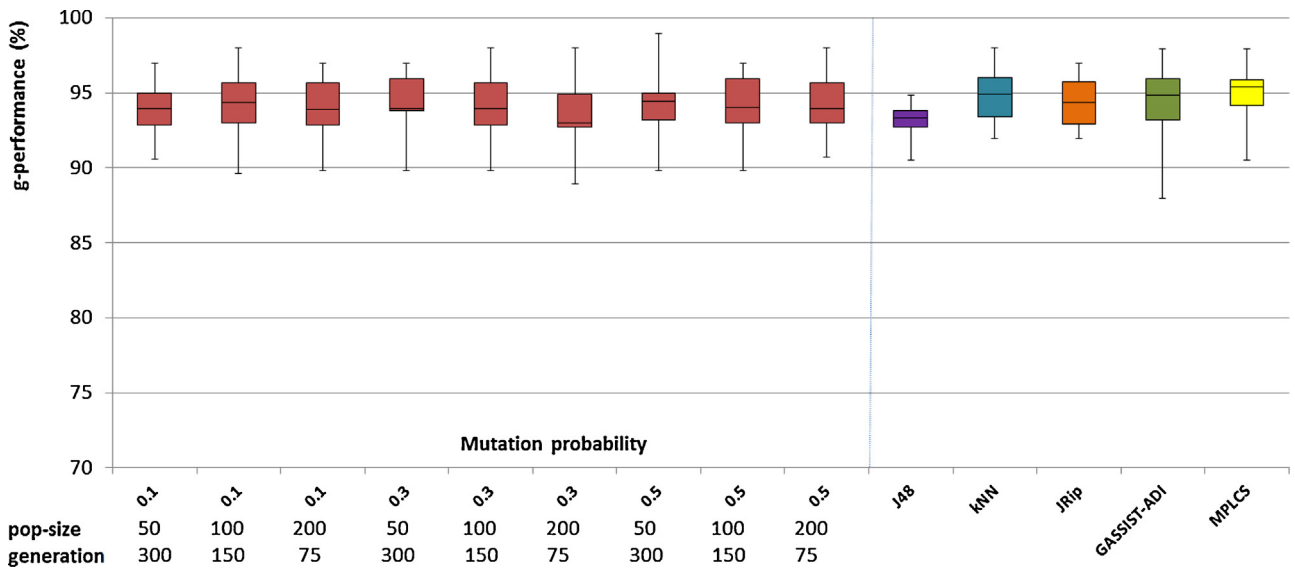

Fig. 6. Performance of the algorithm using fitness function (3). Once again, the median performance is similar among all the algorithms. The variability is less than that for the other two fitness functions, with minima consistently around 90%, regardless of mutation rate and population size.

Mutation probability = 0.1, Crossover probability = 0.5, Generations = 300, Population size = 50.

4.2. Evaluation of the proposed algorithm on complex synthetic data

To test the system further, we evaluate our proposed algorithm with more complex synthetic datasets with NID problem characteristics:

- A 3-dimensional problem with two normal clusters and five anomalous clusters.
- A problem with 6 input features (6-dimension).
- A problem with 12 input features (12-dimension).

Fig. 7 presents a problem with 2 green and 5 red regions to test the ability of the proposed system to generate an adequate number of rules when there are more clusters of attacks. Table 2 compares the performance of our proposed system to the comparison methods when applied to this problem. The result shows that our proposed system is comparable in performance and reliability to the other methods.

Similarly, we compared the results from these classification techniques on problems with more features (6 and 12) in Tables 3 and 4. In both problems, our proposed system provides results comparable to other techniques, with J48, JRip, GASSIST-ADI and MPLCS slightly better with 12 features and kNN slightly worse.

Next, we will test our proposed system within the context of a real-world problem. This will show the flexibility of our system to be applied on different problems with continuous-valued features.

5. Use of the proposed approach in network intrusion detection

In this section, we report on experiments using two sources of network data: the well-known NSL-KDD dataset and a combined DARPA/CAIDA dataset. As in the experiments described above, we examine and compare the performance of our system, alongside k-nearest neighbours, decision trees, RIPPER, GASSIST-ADI and MPLCS. As before, we are interested in accuracy, reliability, and the

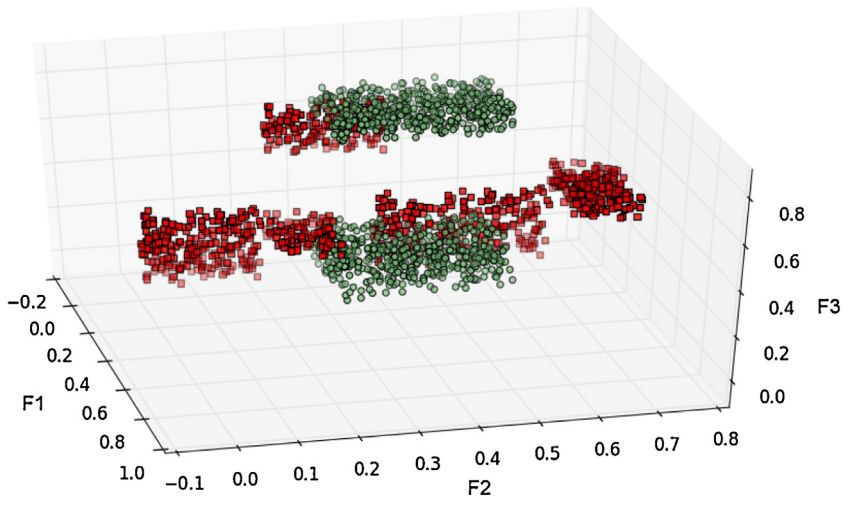

Fig. 7. A problem to test the proposed system in the detection of more clusters of hits. This problem has two “normal” clusters and five “attack” clusters.

Table 2
The performance of different techniques on the problem shown in Fig. 7.

Classifier	Specifications	g-performance				
		Min	$q1$	Median	$q3$	Max
J48	Size of tree: 11 number of leaves: 6	90.5	91.9	92.9	93.8	95
kNN	k = 31	89.4	91.6	93.3	93.8	96.3
JRip	number of rules: 6	91	91.4	93.8	94.2	95.8
GASSIST-ADI	number of rules: 4	89.9	91.3	92.3	93.7	96.9
MPLCS	number of rules: 5	87.3	90.4	93.1	93.7	95.9
Our algorithm	number of rules: 6	90	91.5	92.9	94.3	95.7

Table 3
The performance of different techniques on a problem with 6 features.

Classifier	Specifications	g-performance				
		Min	$q1$	Median	$q3$	Max
J48	Size of tree: 13 number of leaves: 7	95.7	97.9	99.1	99.5	99.8
kNN	k = 22	96	98	99	99.2	100
JRip	number of rules: 5	92	97.2	97.9	99.7	99.9
GASSIST-ADI	number of rules: 5	96.9	97.9	98.9	100	100
MPLCS	number of rules: 5	94.8	96.9	97.9	99.7	100
Our algorithm	number of rules: 5	94.7	97.2	98.9	98.9	99.9

Table 4
The performance of different techniques on a problem with 12 features.

Classifier	Specifications	g-performance				
		Min	$q1$	Median	$q3$	Max
J48	Size of tree: 9 number of leaves: 5	87.7	89.5	90.5	91.4	92.3
kNN	k = 12	83.1	86	86.7	88	89.3
JRip	number of rules: 5	87.3	89	89.5	90.6	92.3
GASSIST-ADI	number of rules: 5	84.4	89.3	89.5	91.2	92.3
MPLCS	number of rules: 5	88.3	89.1	90.1	91.3	92.8
Our algorithm	number of rules: 5	84	86	88	89	92

understandability of the resulting classifier (using the number of rules as a proxy).

To run the experiments with these two sets of data, we used the settings that were found to be most reliable in the experiments with synthetic dataset for all the algorithms (Section 4.1). We also used the seed selection module to insert a set of rules as a proportion (20%) of the initial population. This enables the algorithm to evolve faster than with random initialisation.

5.1. Choice of dataset

Based on the reviewed research work, there are two publicly available datasets that have been widely used in either signature-based or anomaly-based detection system evaluations: the DARPA-Lincoln datasets and the KDD99 dataset. Both of these datasets are recognised to have some weaknesses in terms of their use in machine learning studies. Therefore, some researchers have used various methods to generate their own datasets to better suit their particular needs. In this paper, we first use a well known and well understood dataset (NSL-KDD) to allow comparison with existing studies. The NSL-KDD proposed by Tavallaee et al. [10] is a new version of KDD99, which addresses two problems in the KDD99 dataset. First, it does not include redundant (repeated) records. Second, records from KDD are resampled based on an estimate of how difficult each record is to classify. The number of selected records from each difficulty group is inversely proportional to the percentage of records in the original dataset (so records that are difficult to classify are selected more often). These changes allow for greater discrimination between performance levels of different algorithms. Additionally, the number of records in the training and test datasets is not too large, which enables researchers to run experiments on the complete set and avoids the need to randomly select a small portion of the much larger KDD99 for training and testing, as has been done in the past. However, to address the criticisms associated with KDD99 and NSL-KDD datasets that have been discussed by many researchers in the community [12], the MIT Lincoln Laboratory tcpdump data (i.e. real-time pure normal data) is combined with the CAIDA DDoS dataset to test the proposed algorithm further. The normal traffic scenario is the data from Thursday in the third training week, which does not contain any attacks. Additionally, the CAIDA dataset contains 5 min of anonymised traffic of a DDoS attack on August 4, 2007. Non-attack traffic has been removed as much as possible from this data. This approach of obtaining the combined dataset has been used by other researchers in this domain to evaluate techniques for IDSs [15–17].

5.2. Features extracted from the NSL-KDD and the combined DARPA/CAIDA dataset

Our proposed algorithm is designed to classify records based on statistical (and therefore continuous) features. Therefore, for the NSL-KDD, we remove the categorical features, such as *protocol_type*, from the existing 41 features in the dataset. Our reduced dataset only contains 32 continuous features. As part of the proposed framework, we can optionally apply a feature selection module to find the most relevant continuous features. To identify the important input features in the NSL-KDD dataset for building an efficient and effective IDS, S. Mukherjee and N. Sharma [43] investigated three feature selection methods implemented in WEKA 3.6: Correlation-based Feature Selection (CFS), Information Gain (IG) and Gain Ratio (GR). The result showed that the feature subset identified by CFS gave better classification accuracy than IG and GR. While a CFS method measures the individual predictive ability of each attribute along with the degree

Table 5
The performance of our proposed algorithm on the NSL-KDD dataset with 8 features.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 245 number of leaves: 123	76.3	96.6	60.3	76
kNN	1 nearest neighbour	77.6	97	62.2	77
JRip	number of rules: 39	76.4	96.5	60	76.1
GASSIST-ADI	number of rules: 5	76.1	97	59.8	76
MPLCS	number of rules: 5	76	97	59.6	76
Our algorithm	number of rules: 9	76.5	96.5	60.8	76.2

of redundancy between them, the Consistency Subset Evaluator (CSE) measures the inconsistency of a feature set given different class labels. The CSE approach was compared to the CFS method in [44] and gave better classification accuracy. The 8 attributes selected by CFS are: src_bytes, dst_bytes, num_root, same_srv_rate, diff_srv_rate, srv_serror_rate, dst_host_diff_host_rate and dst_host_srv_serror_rate. The 15 features selected by CSE are: duration, src_bytes, dst_bytes, count, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dist_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_diff_host_rate, dst_host_serror_rate, dst_host_srv, serror_rate, dst_host_rerror_rate and dst_host_srv_rerror_rate.

For the combined dataset (DARPA/CAIDA) used in our experiments, we collected entropy of selected packet attributes for a time-window of 10 s: entropy of source IP address and port number, entropy of destination port number, entropy of packet type and entropy of packet size. Entropy is a measure of the uncertainty associated with a random variable. Let an information source has n independent symbols each with probability of choice p_i. Then, the entropy H is defined as follows [45]:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \quad (5)$$

Prior research has shown that entropy-based detectors are simple statistical measures, which discriminate DDoS traffic from legitimate [7,16,46–48]. Simple calculation, high sensitivity, low false positive rate and no need of additional network traffic and device are the advantages of entropy-based approaches [46].

5.3. Discussion

Tables 5–7 compare the results obtained using our proposed algorithm with J48, RIPPER, kNN, GASSIST-ADI and MPLCS when applied to the NSL-KDD dataset with 8 (using CFS) and 15 (using CSE) features, respectively. Additionally, we tested the system without feature selection on all 32 continuous features, with results shown in Table 8.

Table 6
A ruleset generated by our system for the NSL-KDD dataset with 8 features.

Ruleset with 77% performance	
rule1:	*diff_srv_rate* ∈ [0.04 0.29]
rule2:	*dst_bytes* ∈ [0 1261360321] and *num_root* ∈ [0 149] and *srv_rerror_rate* ∈ [0 0.53] and *dst_host_srv_serror_rate* ∈ [0.34 1]
rule3:	*srv_rerror_rate* ∈ [0.98 1] and *dst_host_srv_diff_host_rate* ∈ [0 0.02] and *dst_host_srv_serror_rate* ∈ [0 0.10]
rule4:	*dst_bytes* ∈ [0 26198748] and *num_root* ∈ [0 149] *srv_rerror_rate* ∈ [0.47 1] and *same_srv_rate* ∈ [0 0.93] and *dst_host_srv_serror_rate* ∈ [0 0.75]
rule5:	*dst_host_srv_diff_host_rate* ∈ [0.23 0.93] and *dst_host_srv_serror_rate* ∈ [0 0.97]
rule6:	*dst_bytes* ∈ [0 26198748] and *srv_rerror_rate* ∈ [0.61 0.92] and *dst_host_srv_diff_host_rate* ∈ [0 0.02]

Note that NSL-KDD provides both a training set and a separate test set, so these results are not cross-validated, and should be considered less reliable than the results in earlier sections. We trained each classifier on the training set and tested on the test set, as this is the common practice in studies using NSL-KDD in the literature. As our algorithm is stochastic, we repeated each test 30 times, and reported average values. Similarly, for GASSIST-ADI and MPLCS, average values for 30 runs are presented. J48, kNN and JRip are deterministic, so only a single figure is reported for each test.

The results indicate that our approach has similar performance to the other algorithms in all cases, except that J48 does slightly better when there are many more features, and kNN, JRip, GASSIST-ADI and MPLCS do slightly worse. In all cases, our system discovers rulesets up to an order of magnitude smaller than those found by J48, and having only 23-50% as many rules as RIPPER. Generally in the case of NSL-KDD dataset, GASSIST-ADI and MPLCS generated

Table 7
The performance of our proposed algorithm on the NSL-KDD dataset with 15 features.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 371 number of leaves: 186	78.4	97.2	63.3	77.9
kNN	1 nearest neighbour	75	93.4	60.3	74.6
JRip	number of rules: 46	77.8	96.1	63.1	77
GASSIST-ADI	number of rules: 6	75.3	92.1	61.6	74.7
MPLCS	number of rules: 6	73.3	91.4	58.8	72.8
Our algorithm	number of rules: 18	76.4	92.7	63.3	75.9

Table 8
The performance of our proposed algorithm on the NSL-KDD dataset with all continuous features.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 323 number of leaves: 162	82.8	95	72.3	82
kNN	1 nearest neighbour	75.2	93	60.9	75.1
JRip	number of rules:38	76.3	95	61.3	76.8
GASSIST-ADI	number of rules: 6	74.9	93.4	60.2	74.5
MPLCS	number of rules: 5	75.5	90.8	62.9	74.9
Our algorithm	number of rules: 19	78.1	87	70	78

Table 9
The performance of our proposed algorithm on the combined DARPA/CAIDA dataset.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 3 number of leaves: 3	81.9	86.7	77.4	84
kNN	1 nearest neighbour	98.3	100	96.7	98
JRip	number of rules:3	96.7	100	93.5	97.9
GASSIST-ADI	number of rules: 4	99.2	100	98.5	98.9
MPLCS	number of rules: 4	98.3	100	96.7	98.9
Our algorithm	number of rules: 2	97.8	99.7	96	98.4

less number of rules with slightly less accuracy than the proposed approach. When the input data is more complex and includes all the attributes, although GASSIST-ADI and MPLCS generated less rules than our system, this decreased the effectiveness of the system in detection of anomalous records (and as a result the accuracy of the system) as can be seen from the true positive rates in Table 8. To provide an example of the output of our system, Table 6, illustrates the final 6-rule ruleset generated for classification of the NSL-KDD dataset with 8 features.

Finally, the results obtained from the experiments on the DARPA/CAIDA dataset is presented in Table 9. In this classification problem, our algorithm, kNN, GASSIST-ADI and MPLCS produced better results than J48, RIPPER classifiers.

6. Conclusions and future work

In this paper, we demonstrated a hybrid approach that combines statistical measurements of network traffic with an evolutionary algorithm for evolving rulesets for intrusion detection. Three different fitness functions were tested in order to evolve rules cooperatively to provide a final optimised ruleset, which covers the area of search precisely. This reduces the number of rules covering overlapping regions, which is an issue in other rule-based systems. The algorithms used in this paper are first tuned using a synthetic problem. Using the settings obtained from the tuning for all the respective approaches, the proposed algorithm is then evaluated against the other techniques using more complex synthetic datasets with NID problem characteristics and two network datasets (NSL-KDD and combined DARPA/CAIDA). The results showed that the proposed method worked effectively for the synthetic data compared to other machine learning techniques: decision trees, RIPPER, kNN, GASSIST-ADI and MPLCS.

For NSL-KDD data, continuous-valued network features were extracted to avoid the problem of being dependant on the signature of network packets in some IDSs. For the combined DARPA/CAIDA dataset, the input data features are entropy of 5 packet attributes. The results showed that the performance of our system is comparable to the other methods tested, and has produced very compact, easily understood rulesets. Another advantage of our system is that, depending on the problem domain and the characteristics of the dataset, the system is customisable by changing the fitness and performance function to provide customised optimal results. This option can not be found in other techniques and thus makes our system a more flexible model.

There are a number of directions to explore in order to improve the performance of our system as well as its range of application. With some evolved rulesets, the fitness and performance measurements that drive the algorithm are insensitive to small modifications to boundary values (because there are no training examples nearby). We intend to investigate efficient local generalisation and specialisation operators that could exploit this fact. For example, they could be used in a memetic version of the algorithm, combining evolution with local modification.

The system has been designed to detect attacks, but by searching instead for rules that "detect" normal traffic, it could be used to discover rulesets to be used in an anomaly-based NIDS. Also, the system should be flexible enough to be modified for multi-class classification, to discover rulesets that can identify which kind of attack is taking place.

Finally, in an operational NIDS, detection could be considered as a dynamic optimisation problem. As new processed data becomes available (either normal traffic or new attacks), the population of rules could undergo continuous evolution. As our experiments show, only a few rules from each population are actually used in the optimised ruleset, so the remaining rules may be useful "genetic material" as the system adapts to changing patterns of normal usage or new attack variations.

Acknowledgements

The authors would like to thank the associate editor and anonymous reviewers for their valuable comments and suggestions that have helped to improve the paper.

References

[1] E.H. Spafford, D. Zamboni, Intrusion detection using autonomous agents, Comput. Netw. 34 (4) (2000) 547–570.
[2] S.X. Wu, W. Banzhaf, The use of computational intelligence in intrusion detection systems: a review, Appl. Soft Comput. 10 (1) (2010) 1–35.
[3] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based network intrusion detection: techniques systems and challenges, Comput. Secur. 28 (1) (2009) 18–28.
[4] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, W.-Y. Lin, Intrusion detection by machine learning: a review, Expert Syst. Appl. 36 (10) (2009) 11994–12000.
[5] G. Kumar, K. Kumar, M. Sachdeva, The use of artificial intelligence based techniques for intrusion detection: a review, Artif. Intell. Rev. 34 (4) (2010) 369–387.
[6] V. Dhar, D. Chou, F. Provost, Discovering interesting patterns for investment decision making with GLOWER: a genetic learner overlaid with entropy reduction, Data Min. Knowl. Discov. 4 (4) (2000) 251–280.
[7] L. Feinstein, D. Schnackenberg, R. Balupari, D. Kindred, Statistical approaches to DDoS attack detection and response, in: DARPA Information Survivability Conference and Exposition, 2003. IEEE Proceedings, vol. 1, 2003, pp. 303–314.
[8] J. Bacardit, J.M. Garrell, Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based learning classifier system, in: Genetic and Evolutionary Computation-GECCO 2003, Springer, 2003, pp. 1818–1831.
[9] J. Bacardit, N. Krasnogor, Performance and efficiency of memetic Pittsburgh learning classifier systems, Evol. comput. 17 (3) (2009) 307–342.
[10] M. Tavallaee, E. Bagheri, W. Lu, A.-A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in: Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications, 2009.
[11] KDD99, http://kdd.ccs.uci.edu/databases/kddcup99/task.html
[12] R. Sommer, V. Paxson, Outside the closed world: On using machine learning for network intrusion detection, in: 2010 IEEE Symposium on Security and Privacy (SP), IEEE, 2010, pp. 305–316.
[13] MIT Lincoln Laboratory Datasets, DARPA dataset, 2000 http://www.ll.mit.edu/ideval/data/2000data.html,.
[14] CAIDA, http://www.caida.org/data/passive/ddos-20070804_dataset.xml
[15] S. Bhatia, G. Mohay, A. Tickle, E. Ahmed, Parametric differences between a real-world distributed denial-of-service attack and a flash event, in: 2011 Sixth International Conference on Availability, Reliability and Security (ARES), IEEE, 2011, pp. 210–217.
[16] Y. Xiang, K. Li, W. Zhou, Low-rate DDoS attacks detection and traceback by using new information metrics, IEEE Trans. Inform. Forensics Secur. 6 (2) (2011) 426–437.
[17] M.H. Bhuyan, D. Bhattacharyya, J. Kalita, Information metrics for low-rate DDoS attack detection: A comparative evaluation, in: 2014 Seventh International Conference on Contemporary Computing (IC3), IEEE, 2014, pp. 80–84.
[18] W.W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 115–123.

[19] W. Lee, S.J. Stolfo, K.W. Mok, A data mining framework for building intrusion detection models, in: Proceedings of the 1999 IEEE Symposium on Security and Privacy, IEEE, 1999, pp. 120–132.
[20] W. Lee, S.J. Stolfo, A framework for constructing features and models for intrusion detection systems, ACM Trans. Inform. Syst. Secur. 3 (4) (2000) 227–261.
[21] K. Shafi, H.A. Abbass, Evaluation of an adaptive genetic-based signature extraction system for network intrusion detection, Pattern Anal. Appl. 16 (4) (2013) 549–566.
[22] W. Li, Using genetic algorithm for network intrusion detection, in: Proceedings of the United States Department of Energy Cyber Security Group, 2004, pp. 1–8.
[23] M.V. Fidelis, H. Lopes, A. Freitas, Discovering comprehensible classification rules with a genetic algorithm, in: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 1, IEEE, 2000, pp. 805–810.
[24] D.E. Goldberg, et al., Genetic Algorithms in Search, Optimization, and Machine Learning, vol. 412, Addison-Wesley Reading Menlo Park, 1989.
[25] C.R. Reeves, J.E. Rowe, Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory, vol. 20, Springer, 2003.
[26] R.H. Gong, M. Zulkernine, P. Abolmaesumi, A software implementation of a genetic algorithm based approach to network intrusion detection, in: 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks on Software Engineering, Sixth International Conference on Artificial Intelligence, Networking and Parallel/Distributed Computing, IEEE, SNPD/SAWN 2005, 2005, pp. 246–253.
[27] MIT Lincoln Laboratory, DARPA Dataset, 2015 http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/index.html
[28] W. Lu, I. Traore, Detecting new forms of network intrusion using genetic programming, Comput. Intell. 20 (3) (2004) 475–494.
[29] F. Hoffmann, Combining boosting and evolutionary algorithms for learning of fuzzy classification rules, Fuzzy Sets Syst. 141 (1) (2004) 47–58.
[30] J.H. Holland, L.B. Booker, M. Colombetti, M. Dorigo, D.E. Goldberg, S. Forrest, R.L. Riolo, R.E. Smith, P.L. Lanzi, W. Stolzmann, et al., What is a learning classifier system? in: Learning Classifier Systems, Springer, 2000, pp. 3–32.
[31] K. Shafi, H.A. Abbass, An adaptive genetic-based signature learning system for intrusion detection Expert Syst. Appl. 36 (10) (2009) 12036–12043.
[32] J. Bacardit, J.M. Garrell, Bloat control and generalization pressure using the minimum description length principle for a Pittsburgh approach learning classifier system, in: Learning Classifier Systems, Springer, 2007, pp. 59–79.
[33] M.A. Akbar, M. Farooq, Application of evolutionary algorithms in detection of sip based flooding attacks, in: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, 2009, pp. 1419–1426.
[34] S.W. Wilson, Get real! XCS with continuous-valued inputs, in: Learning Classifier Systems, Springer, 2000, pp. 209–219.
[35] H. Hou, G. Dozier, Immunity-based intrusion detection system design, vulnerability analysis, and GENERTIA's genetic arms race, in: Proceedings of the 2005 ACM Symposium on Applied Computing, ACM, 2005, pp. 952–956.
[36] R.S. Hine, E. Martin, Oxford Dictionary of Biology, 2004.
[37] D. Ashlock, W. Ashlock, Impact of regulatory genes on optimization behavior, in: 2012 IEEE Congress on Evolutionary Computation (CEC), 2012, pp. 1–8, http://dx.doi.org/10.1109/CEC.2012.6252981
[38] M. Kubat, S. Matwin, et al., Addressing the curse of imbalanced training sets: one-sided selection, vol. 97, in: ICML, Nashville, USA, 1997, pp. 179–186.
[39] S. Garcí a, F. Herrera, Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy, Evol. Comput. 17 (3) (2009) 275–306.
[40] R. Barandela, J.S. Sánchez, V. Garcıa, E. Rangel, Strategies for learning in class imbalance problems, Pattern Recogn. 36 (3) (2003) 849–851.
[41] I.H. Witten, E. Frank, L.E. Trigg, M.A. Hall, G. Holmes, S.J. Cunningham, Weka: Practical Machine Learning Tools and Techniques with Java Implementations, 1999.
[42] J. Alcala-Fdez, L. Sanchez, S. Garcia, M.J. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, et al., KEEL: a software tool to assess evolutionary algorithms for data mining problems, Soft Comput. 13 (3) (2009) 307–318.
[43] S. Mukherjee, N. Sharma, Intrusion detection using naive Bayes classifier with feature reduction, Proc. Technol. 4 (2012) 119–128.
[44] K.-C. Khor, C.-Y. Ting, S.-P. Amnuaisuk, Forming an optimal feature set for classifying network intrusions involving multiple feature selection methods, in: 2010 International Conference on Information Retrieval & Knowledge Management (CAMP), IEEE, 2010, pp. 179–183.
[45] C.E. Shannon, A mathematical theory of communication, ACM SIGMOBILE Mobile Comput. Commun. Rev. 5 (1) (2001) 3–55.
[46] J. Zhang, Z. Qin, L. Ou, P. Jiang, J. Liu, A. Liu, An advanced entropy-based DDOS detection scheme, in: 2010 International Conference on Information Networking and Automation (ICINA), Vol. 2, IEEE, 2010, pp. V2–67.
[47] L. Zi, J. Yearwood, X.-W. Wu, Adaptive clustering with feature ranking for DDoS attacks detection, in: 2010 4th International Conference on Network and System Security (NSS), IEEE, 2010, pp. 281–286.
[48] M.H. Sqalli, S.N. Firdous, Z. Baig, F. Azzedin, An entropy and volume-based approach for identifying malicious activities in honeynet traffic, in: 2011 International Conference on Cyberworlds (CW), IEEE, 2011, pp. 23–30.