A Software Implementation of a Genetic Algorithm Based Approach to Network Intrusion Detection

Ren Hui Gong, Mohammad Zulkernine, Purang Abolmaesumi
School of Computing
Queen's University
Kingston, Ontario, Canada K7L 3N6
{rhgong, mzulker, purang}@cs.queensu.ca

Abstract

With the rapid expansion of Internet in recent years, computer systems are facing increased number of security threats. Despite numerous technological innovations for information assurance, it is still very difficult to protect computer systems. Therefore, unwanted intrusions take place when the actual software systems are running. Different soft computing based approaches have been proposed to detect computer network attacks. This paper presents a genetic algorithm (GA) based approach to network intrusion detection, and the software implementation of the approach. The genetic algorithm is employed to derive a set of classification rules from network audit data, and the support-confidence framework is utilized as fitness function to judge the quality of each rule. The generated rules are then used to detect or classify network intrusions in a real-time environment. Unlike most existing GA-based approaches, because of the simple representation of rules and the effective fitness function, the proposed method is easier to implement while providing the flexibility to either generally detect network intrusions or precisely classify the types of attacks. Experimental results show the achievement of acceptable detection rates based on benchmark DARPA data sets on intrusions, while no other complementary techniques or relevant heuristics are applied.

Keywords: *Information assurance, misuse intrusion detection, genetic algorithms, support-confidence framework, software development.*

1. Introduction

The Internet and local area networks are expanding at an amazing rate in recent years. While we are benefiting from the convenience that the new technology has brought us, computer systems are exposed to increasing security threats that originate externally or internally. Different but complementary technologies have been developed and deployed to protect organizations' computer systems against network attacks, for example, anti-virus software, firewall, message encryption, secured network protocols, password protection, and so on. Despite different protection mechanisms, it is nearly impossible to have a completely secured system. Therefore, intrusion detection is becoming an increasingly important technology that monitors network traffic and identifies network intrusions such as anomalous network behaviors, unauthorized network access, and malicious attacks to computer systems [15].

There are two general categories of intrusion detection systems (IDSs): misuse detection and anomaly detection [16]. Misuse detection systems detect intruders with known patterns, and anomaly detection systems identify deviations from normal network behaviors and alert for potential unknown attacks. Some IDSs integrate both misuse and anomaly detection and form hybrid detection systems. The IDSs can also be classified into two categories depending on where they look for intrusions. A host-based IDS monitors activities associated with a particular host, and a network-based IDS listens to network traffic.

A number of soft computing based approaches have been proposed for detecting network intrusions [1, 2, 3, 4, 6, 10]. Soft computing refers to a group of techniques that exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve robustness and low solution cost. The principle constituents of soft computing are Fuzzy Logic (FL), Artificial Neural Networks (ANNs), Probabilistic Reasoning (PR), and Genetic Algorithms (GAs) [10]. When used for intrusion detection, soft computing techniques are often used in conjunction with rule-

0-7695-2294-7/05 $20.00 © 2005 **IEEE**

based expert systems acquiring expert knowledge [1, 4, 5, 6], where the knowledge is represented as a set of *if-then* rules. Despite different soft computing based approaches having been proposed, the possibilities of using the techniques for intrusion detection are still under-utilized.

In this paper, we present a GA-based approach to network misuse detection. GA is chosen because of some of its nice properties, *e.g.*, robust to noise, no gradient information is required to find a global optimal or sub-optimal solution, self-learning capabilities, etc. Using GAs for network intrusion detection has proven to be a cost-effective approach [1, 2, 3, 7, 8, 9, 11]. In this work, we implement a software based on the presented approach. The software is experimented using DARPA data sets on intrusions, which has become the de facto standard for testing intrusion detection systems. The experimental results show that our approach is effective, and it has the flexibility to either generally detect network intrusions or precisely classify the types of misuses. This is due to the use of both categorical and quantitative features of network audit data for deriving the classification rules, and the use of the support-confidence framework as the GA fitness function.

Paper Organization. Thus far, we have discussed the motivation and a brief overview of the presented work. The rest of the paper is organized as follows. Section 2 gives an overview of the genetic algorithm employed in this work. Section 3 reviews the work relevant to this research, while some of the more closely related work are discussed in the relevant parts of this paper. Sections 4 and 5 describe in detail the proposed method and its software implementation. Section 6 presents the experimental results, and Section 7 concludes the paper with some future recommendations.

2. Genetic Algorithms

Genetic algorithms [3, 12] employ metaphor from biology and genetics to iteratively evolve a population of initial individuals to a population of high quality individuals, where each individual represents a solution of the problem to be solved and is composed of a fixed number of genes. The number of possible values of each gene is called the cardinality of the gene. Figure 1 illustrates the operation of a general genetic algorithm. The operation starts from an initial population of randomly generated individuals. Then the population is evolved for a number of generations and the qualities of the individuals are gradually improved. During each generation, three basic genetic operators are sequentially applied to each individual with certain probabilities, *i.e.*, selection, crossover, and mutation. First, a number of best-fit individuals are selected based on a user-defined fitness function. The remaining individuals are discarded. Next, a number of individuals are selected and paired with each other. Each individual pair produces one offspring by partially exchanging their genes around one or more randomly selected crossing points. At the end, a certain number of individuals are selected and the mutation operations are applied, *i.e.*, a randomly selected gene of an individual abruptly changes its value.

One extension of genetic algorithms, namely Genetic Programming (GP) [3, 8], is also commonly used. It differs from GAs in the way of encoding individuals. GAs use fixed length vectors to represent individuals. In contrast, GP encodes each individual with a parse tree, where leaf nodes are genes and non-leaf nodes are primitive functions (*e.g.*, AND, OR, etc.). GP has the flexibility to represent very complex individuals. In the context of rule based expert systems, GAs are often used to efficiently derive simple rules, and GP is used when more complex or accurate rules are required.

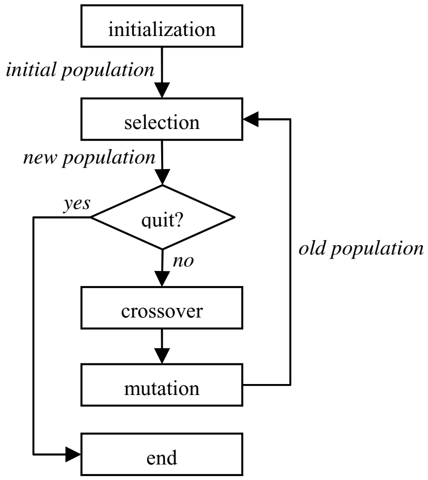

Figure 1. The operation of a generic GA.

When a GA is used for problem-solving, three factors will have impact on the effectiveness of the algorithm, they are: 1) the selection of fitness function; 2) the representation of individuals; and 3) the values of the GA parameters. The determination of these factors often depends on applications. In our implementation for network intrusion detection, the

0-7695-2294-7/05 $20.00 © 2005 IEEE

support-confidence framework was used as fitness function, a simple GA (rather than GP) was employed to represent and derive rules, and appropriate GA parameters, including selection rate, crossing over style, mutation rate, etc, were chosen based on a large number of experiments.

3. Related Work

This section briefly summarizes some of the applications of soft computing techniques for intrusion detection. However, a number of GA based IDSs are discussed in the later part of the paper in order to compare and contrast those work with our work.

GAs and GP have been used for network intrusion detection in different ways. Some approaches directly use GAs to derive the classification rules [2, 7, 8, 11], while some others use different AI methods for acquisition of rules, where GAs are used to select appropriate features or to determine the optimal parameters of some functions [1, 5, 9].

The early effort of using GAs for intrusion detection can be dated back to 1995, when Crosbie *et al.* [3] applied the multiple agent technology and GP to detect network anomalies. Each agent monitors one parameter of the network audit data and GP is used to find the set of agents that collectively determine anomalous network behaviors. This method has the advantage of using many small autonomous agents, but the communication among them is still a problem. Also the training process can be time-consuming if the agents are not appropriately initialized.

Bridges *et al.* [1] develop a method that integrates fuzzy data mining techniques and genetic algorithms to detect both network misuses and anomalies. In most of the existing GA based IDSs, the quantitative features of network audit data are either ignored or simply treated, though such features are often involved in intrusion detection. This is because of the large cardinalities of quantitative features. The authors propose a way to include quantitative features by introducing fuzzy numerical functions. Their preliminary experiments show that the inclusion of quantitative features and the fuzzy functions significantly improved the accuracy of the generated rules. In this approach, a GA is used to find the optimal parameters of the fuzzy functions as well as to select the most relevant network features.

Lu *et al.* [8] present an approach that uses GP to directly derive a set of classification rules from historical network data. The approach employs the support-confidence framework as the fitness function and is able to generally detect or precisely classify network intrusions. However, the use of GP makes implementation more difficult and more data or time are required to train the system.

Li [7] propose a GA-based method to detect anomalous network behaviors. Both quantitative and categorical features of network data are included when deriving classification rules using GA. The inclusion of quantitative features may lead to increased detection rates. However, no experimental results are available yet.

Xiao *et al.* [17] present an approach that uses information theory and GA to detect abnormal network behaviors. Based on the mutual information between network features and the types of network intrusions, a small number of network features are closely identified with network attacks. Then a linear structure rule is derived using the selected features and a GA. The use of mutual information reduces the complexity of GA, and the single resulting linear rule makes intrusion detection efficient in real-time environment. However, the approach considers only discrete features.

4. A GA-Based IDS

The proposed GA-based intrusion detection approach contains two modules where each works in a different stage. In the training stage, a set of classification rules are generated from network audit data using the GA in an offline environment. In the intrusion detection stage, the generated rules are used to classify incoming network connections in the real-time environment. Once the rules are generated, the intrusion detection is simple and efficient. In the following sections, we focus our discussions on deriving the set of rules using GA.

4.1. Data Representation

Several network features have higher possibilities to be involved in network intrusions [7, 9]. In our approach, seven of those features are selected from the network audit data to compose a classification rule. Table 1 shows the features and their formats. The feature names are given in the first column, while the second and third columns indicate how each of the network features is encoded in a chromosome. The second column represents the feature format and the third column shows the number of genes used for the corresponding feature.

Note that different genes can be represented in different data types such as byte, integer, and float.

This is necessary because of different formats and data ranges for different features. For example, the feature "Duration" has three components (hours, minutes, and seconds), each of which is represented by one gene of type byte. Similarly, each of the features "Protocol", "Source port", "Destination port" and "Attack name" is encoded using one gene of type integer, and each of the features "Source IP" and "Destination IP" has four components (a, b, c, and d), each of which is represented by one gene of type byte.

Table 1. Selected network features.

Feature Name	Format	Number of Genes
Duration	h:m:s	3
Protocol	Int	1
Source_port	Int	1
Destination_port	Int	1
Source_IP	a.b.c.d	4
Destination_IP	a.b.c.d	4
Attack_name	Int	1

Each rule is an *if-then* clause, which contains a "condition" and an "outcome". The first six features in Table 1 are connected using the logical AND operations and compose the "condition" part of a rule. The feature "Attack name" is used in the "outcome" part, which indicates the classification of a network record (at training stage) or connection (at intrusion detection stage) when the "condition" part of a rule is matched. The following shows a rule example that classifies a network connection as the denial-of-service attack *neptune*.

if *(duration="0:0:1" and protocol="finger" and*
source_port=18982 and destination_port=79 and
source_ip="9.9.9.9" and
destination_ip="172.16.112.50")
then *(attack_name="neptune")*

The above rule expresses that if a network packet is originated from IP address 9.9.9.9 and port 18982, and sent to IP address 172.16.112.50 and port 79 using the protocol *finger*, and the connection duration is 1 second, then most likely it is a network attack of type *neptune* that may eventually cause the destination host out of service.

Each rule is encoded as a chromosome using a fixed length vector, where each network feature is encoded using one or more genes of different types (see the second and third column of Table 1). In the above example, the encoded form of the rule would look like as follows:

{0, 0, 1, 2, 18982, 79, 9, 9, 9, 9, 172, 16, 112, 50, 1}

To make the rules more general, wildcards are allowed in several network features. In case of a wildcard, the corresponding gene is encoded as -1. For example, if the above rule was generalized to be applicable to all packets originated from network 9.9.*.*, then the rule would be encoded as:

{0, 0, 1, 2, 18982, 79, 9, 9, -1, -1, 172, 16, 112, 50, 1}

4.2. Fitness Function

To determine the fitness of a rule, the support-confidence framework [8] is used. If a rule is represented as *if A then B*, then the fitness of the rule is determined using following equations:

$$support = |A \text{ and } B| \,/\, N$$
$$confidence = |A \text{ and } B| \,/\, |A|$$
$$fitness = w1 * support + w2 * confidence$$

Here, N is the total number of network connections in the audit data, $|A|$ stands for the number of network connections matching the condition A, and $|A \text{ and } B|$ is the number of network connections that matches the rule *if A then B*. The weights *w1* and *w2* are used to control the balance between the two terms and have the default values of *w1*=0.2 and *w2*=0.8.

One of the nice properties of using this fitness function is that, by changing the weights *w1* and *w2*, the approach can be used for either simply identifying network intrusions or precisely classifying the types of intrusions. In the former case, *w1* is set to 1 and *w2* is set to 0. On the other hand, *w1* = 0 and *w2* = 1 for the latter case. Unlike other fitness functions used in the GA, the selection of *w1* and *w2* in this framework is not crucial to the performance of the approach.

4.3. Detection Algorithm Overview

Listing 1 shows the major steps of the employed detection algorithm as well as the training process. It first generates the initial population, sets the defaults parameters, and loads the network audit data. Then the initial population is evolved for a number of generations. In each generation, the qualities of the rules are firstly calculated, then a number of best-fit rules are selected, and finally the GA operators are applied to the selected rules.

The training process starts by randomly generating an initial population of rules (line 1). The weights and fitness threshold values are initialized in line 2. Line 3 calculates the total number of records in the audit data. Lines 4-18 calculate the fitness of each rule and select the best-fit rules into new population. Lines 19-22 apply the crossover and mutation operators to each rule in the new population. Finally, line 23 checks and decides whether to terminate the training process or to enter the next generation to continue the evolution process.

Algorithm : Rule set generation using genetic algorithm.
Input : Network audit data, number of generations, and population size.
Output : A set of classification rules.

```
1. Initialize the population
2. $W1$ = 0.2, $W2$ = 0.8, $T$ = 0.5
3. $N$ = total number of records in the training set
4. For each chromosome in the population
5.    $A$ = 0, $AB$ = 0
6.    For each record in the training set
7.       If the record matches the chromosome
8.          $AB = AB + 1$
9.       End if
10.      If the record matches only the “condition” part
11.         $A = A + 1$
12.      End if
13.   End for
14.   Fitness = $W1 * AB / N + W2 * AB / A$
15.   If Fitness > $T$
16.      Select the chromosome into new population
17.   End if
18. End for
19. For each chromosome in the new population
20.      Apply crossover operator to the chromosome
21.      Apply mutation operator to the chromosome
22. End for
23. If number of generations is not reached, goto line 4
```

Listing 1. Major steps of the detection algorithm.

A similar technique of generating rules using a GA is used in Li’s approach [7]. However, the approach proposed in this study differs with Li’s method in two aspects: 1) the definition of fitness function; and 2) the representation of rules. First, Li’s method uses a simple form of weighted sum as fitness function, in which weights are used to indicate the significance of each network feature. The weight values are crucial to the final detection performance. For practical usage, additional techniques, for example artificial neural networks, are required to accurately determine those values. Second, Li’s approach encodes only the “condition” parts of the rules so the method is only suitable for detecting network anomalies. In contrast, our approach uses the support-confidence framework as fitness function and their values can be directly computed from historical data. Moreover, both “condition” and “outcome” of a rule are included in encoding in our approach. This has led the benefit of precisely detecting the types of network intrusions.

When comparing to Lu’s method [8], the same fitness function is used in both methods, however, our approach uses GA rather than GP to derive the classification rules. GP has the advantage of representing complex rules, but often more training data and longer training period are required.

5. IDS Implementation

The proposed method is implemented using the Java language, and it is built on top of a third party software package ECJ [14]. ECJ is a comprehensive GA/GP Java toolkit developed and maintained by the ECLab of George Manson University. The package provides a rich set of GA foundation classes. When using ECJ to solve user problems, one of two ways can be used: using the full power of ECJ or only using the ECJ foundation classes. In the former case, minimal programming is required and configuration files are heavily used. In the latter case, more programming effort is involved but it allows more flexibilities and better user customizations. We adopted the second approach, mainly because the genes in our approach have different data types.

The implementation contains two systems: an offline training system for deriving rules from historical data, and an online detection system that uses the generated rules to classify incoming network connections in real-time environment. Figure 2 shows the high-level class diagram of the training system. Four types of classes are present in the diagram:

- A main class (`Main`) that integrates all components and records the generated rules.
- A class (`DarpaReader`) that interfaces with DARPA data. This class reads and preprocesses the DARPA data.
- Classes that interface with ECJ include `Initializer`, `Individual`, `Evaluator`, `Fitness`, and `Breeder`. These classes are extended from the abstract counterparts of ECJ.
- The classes below the thick horizontal line are base classes from the ECJ package [14]. An "ec" prefix is used in each of the class names to indicate that.

When the training process starts, the `Initializer` first initializes the population with a number of randomly generated individuals. Then the GA is executed for a number of generations, each generation contains following two steps:

1) The classes `Evaluator` and `Fitness` are used to select a set of best-fit individuals from the population; and
2) The class `Breeder` applies other genetic operators, *i.e. crossover* and *mutation*, to each individual with a certain probability.

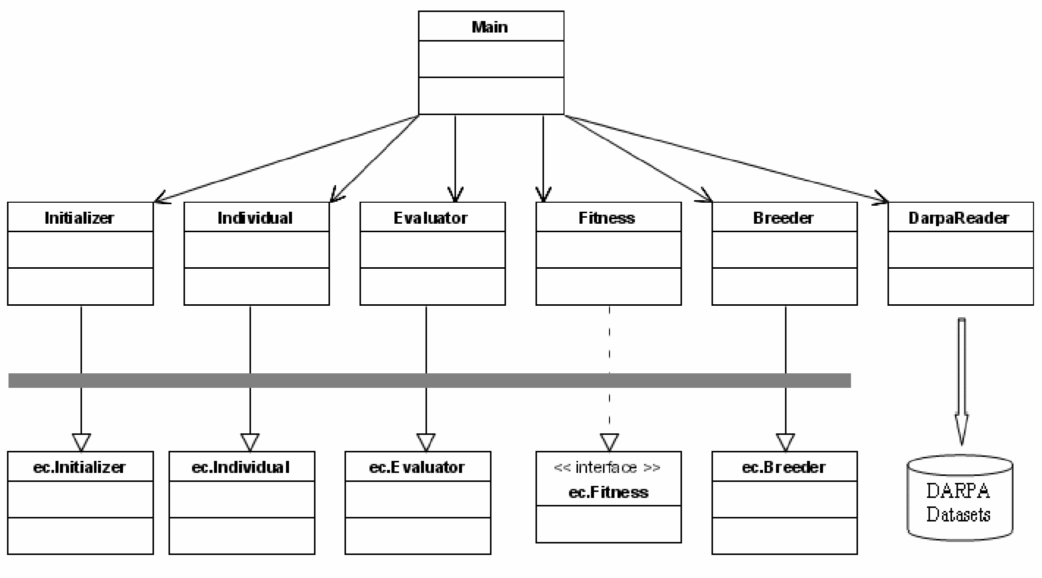

Figure 2. High-level class diagram of the training system.

6. Experimental Results

6.1. Training and Testing Data

The DARPA data from MIT Lincoln Laboratory [13] is broadly used to evaluate IDSs. In this study, two subsets were extracted from the 1998 DARPA data and used as the training and testing datasets. Each record of the datasets consists of 9 network features and 1 manually assigned record type. Six network features were used in the GA [7, 9], which are *connection duration*, *protocol*, *source port*, *destination port*, *source IP address*, and *destination IP address*. The record type indicates whether a record is a normal network connection or a particular network intrusion. Table 2 shows the distributions of record types in training and testing datasets. The first row shows the numbers of normal network packets, while the second and third rows give the distributions of two network attacks.

As shown in Table 2, most network packets in the selected datasets are normal, and two kinds of network attacks are present: *Portsweep and Pod*. *Portsweep* is a kind of attack that sweeps through many ports to determine which services are supported on a single host. *Pod* is a denial of service attack that keeps pinging a host until the service is not available.

Table 2. The distributions of record types.

Record Type	Training Set	Testing Set
Normal	48886	27322
Portsweep	1804	1009
Pod	450	241
Total	51144	28574

6.2. Experiments

Two experiments have been conducted. In the first experiment, the system was trained with the training dataset, and the default fitness function and the GA parameters were used, *i.e.*, *w1*=0.2, *w2*=0.8, 5000 generations, 500 initial rules in the population, crossover rate of 0.5, two-point crossover, and mutation rate of 0.02. When the training process was finished, the top 20 best quality rules were taken as the final classification rules. The rules were then used to classify the training data and the testing data respectively. The results are presented in Table 3.

Table 3. Results (detection rates) of Experiment 1.

Record Type	Training Set	Testing Set
Normal	97.7%	94.2%
Portsweep	91.8%	67.4%
Pod	94.3%	78%

The experimental results show that the proposed method yielded good detection rates when using the generated rules to classify the training data itself (the second column in the table). That is what we expected. The detection rates could be higher if the fitness function and the GA parameters were chosen more appropriately. When the resulting rules were used to classify the testing dataset, the detection rates of network attacks were decreased by around 20% (the third column in the table). The results have indicated that the generated rules were biased to the training data. This is a typical over-fitting problem inherent in major learning techniques, and this problem may be solved in the following two ways: 1) have less number of generations when training the system; or 2) use less number of best-fit rules when classifying new network data. In either approach, an appropriate number (*i.e.*, number of generations or number of best-fit rules) has to be found. This is often done by trial and error.

In the second experiment, the training data was employed to train the system, and the default GA parameters were used. The top 20 best-fit final rules were used to classify the training data. The experiment investigated how the two weights of the fitness function can be used to steer the system to simply detect general network intrusions or to precisely classify the types of intrusions. The experiment was repeated for two kinds of weight settings: 1) *w1*=0, *w2*=1; and 2) *w1*=1, *w2*=0. Table 4 shows the experimental results.

Table 4. Results (detection rates) of Experiment 2.

Record Type	*w1*=0, *w2*=1	*w1*=1, *w2*=0
Normal	97.9%	98.0%
Portsweep	94.9%	35.4%
Pod	96.1%	25.6%
Anomaly	95.7%	96.3%

An additional row (the last row) has been added to Table 4, which shows the percentages of network intrusions being correctly detected, without considering their attack types. When an intrusion of a specific type (*Portsweep* or *Pod*) is detected, it is also classified as an anomaly. As indicated in the table, when *w2* was set to 1, the detection rates of network attacks *Portsweep* and *Pod* were fairly high (second column). On the other hand, when *w1* was set to 1 and *w2* was set to 0, the detection rates for particular attacks were poor, but the total detection rate of network intrusions remained high (third column).

7. Conclusions and Recommendations

In this paper, a method of applying genetic algorithms for network intrusion detection is presented. A software is implemented for the presented method, and its architecture and operations are described in detail using high level class diagram and pseudo-code. A number of experiments have been carried out using a benchmark data set in order to show the efficacy of the developed software. One of the major advantages of this technique is due to the fact that in the real world, the types of intrusions change and become complicated very rapidly. The proposed detection system can upload and update new rules to the systems as the new intrusions become known. Therefore, it is cost effective and adaptive.

A GA is used to derive a set of classification rules from network audit data. Seven network features including both categorical and quantitative data fields were used when encoding and deriving the rules. A simple but efficient and flexible fitness function, *i.e.* the support-confidence framework, is used to select the appropriate rules. Depending on the selection of fitness function weight values, the generated rules can be used to either generally detect network intrusions or precisely classify the types of intrusions.

The method has been implemented using Java and the third party package ECJ. The implementation was tested using selected subsets of the 1998 DARPA data. Experimental results showed that the proposed method worked effectively for the selected datasets and has the flexibility to be used in different ways to meet users' special requirements.

However, some limitations of the method are also observed. First, the generated rules were biased to the training dataset. This issue may be resolved by carefully selecting either the number of generations in the training phase or the number of top best-fit rules in the intrusion detection phase. The problem of low extrapolation power of the presented technique to a new dataset is primarily due to some inherent shortcomings associated with the most soft computing techniques, and this paper is mainly an initial attempt towards overcoming those various shortcomings in the context of network intrusion detection. Nevertheless, the deployment of a number of complimentary techniques and some potential heuristics are being investigated to resolve this problem. Second, while the support-confidence framework is simple to implement and provides improved accuracy to final rules, it requires the whole training data to be loaded into memory before any computation. For large training datasets, it is neither efficient nor feasible. The use of some sorts of cache technologies may solve the problem.

7. References

[1] S. M. Bridges and R. B. Vaughn, "Fuzzy Data Mining And Genetic Algorithms Applied To Intrusion Detection", *Proceedings of 12th Annual Canadian Information Technology Security Symposium,* pp. 109-122, 2000.

[2] A. Chittur, "Model Generation for an Intrusion Detection System Using Genetic Algorithms", http:// www1.cs.columbia.edu/ids/publications/gaids-thesis 01.pdf (accessed in January 2005).

[3] M. Crosbie and E. Spafford, "Applying Genetic Programming to Intrusion Detection", *Proceedings of the AAAI Fall Symposium,* 1995

[4] D. Dasgupta and F. A. Gonzalez, "An Intelligent Decision Support System for Intrusion Detection and Response", *MMM-ACNS, Lecture Notes in Computer Science*, vol. *2052,* pp. 1-14, 2001.

[5] J. Gomez and D. Dasgupta, "Evolving Fuzzy Classifiers for Intrusion Detection", *Proceedings of the IEEE,* 2002.

[6] G. Helmer, J. Wong, V. Honavar and L. Miller, "Automated discovery of concise predictive rules for intrusion detection", *The Journal of Systems and Software,* issue 60, pp. 165-175, 2002.

[7] W. Li, "A Genetic Algorithm Approach to Network Intrusion Detection", SANS Institute, USA, 2004.

[8] W. Lu and I. Traore, "Detecting New Forms of Network Intrusion Using Genetic Programming", *Computational Intelligence,* vol. 20, pp. 3, Blackwell Publishing, Malden, pp. 475-494, 2004.

[9] M. Middlemiss and G. Dick, "Feature selection of intrusion detection data using a hybrid genetic algorithm/KNN approach", *Design and application of hybrid intelligent systems,* IOS Press Amsterdam, pp. 519-527, 2003.

[10] M. Moradi and M. Zulkernine, "A Neural Network Based System for Intrusion Detection and Classification of Attacks", *Proceedings of the 2004 IEEE International Conference on Advances in Intelligent Systems - Theory and Applications,* Luxembourg, November 2004.

[11] M. M. Pillai, J. H. P. Eloff and H. S. Venter, "An Approach to Implement a Network Intrusion Detection System using Genetic Algorithms", *Proceedings of SAICSIT,* pp. 221-228, 2004.

[12] H. Pohlheim, "Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms", http://www.geatbx.com/docu/index.html (accessed in January 2005).

[13] MIT Lincoln Laboratory, DARPA datasets, MIT, USA, http://www.ll.mit.edu/IST/ideval/data/data_index.html (accessed in November 2004).

[14] ECLab: evolutionary Computation laboratory, A Java-based Evolutionary Computation (ECJ) and Genetic Programming Research System, George Mason University, Fairfax, VA, USA, http://cs.gmu.edu/~eclab/projects/ecj/ (accessed in November 2004).

[15] A. Adetoye, A. Choi, M. Md. Arshad, and O. Soretire, "Network Intrusion Detection & Response System", Group Report, September 2003, http://www.cs.ucl.ac.uk/teaching/dcnds/group-reports /2003/2003-hailes-b.pdf (accessed in January 2005).

[16] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection", *IEEE Network*, 8(3): 26-41, May/June 1994.

[17] T. Xiao, G. Qu, S. Hariri, and M. Yousif, "An Efficient Network Intrusion Detection Method Based on Information Theory and Genetic Algorithm", *Proceedings of the 24th IEEE International Performance Computing and Communications Conference (IPCCC '05),* Phoenix, AZ, USA. 2005.