An Intelligent Intrusion Detection System Using Genetic Algorithms and Features Selection

H.M. Shirazi[1], Y. Kalaji[2]
1- Faculty of ICT, Malek-Ashtar University of Tchnology, Tehran, Iran,
Email: shirazi@mut.ac.ir
2- Faculty of ICT, Malek-Ashtar University of Tchnology, Tehran, Iran,
Email: ykalaji@gmail.com

Received: September 2009 Revised: November 2009 Accepted: January 2010

ABSTRACT:
The reports show a rapid growth in the numbers of attacks to the information and communication systems. Also, we witness smarter behaviors from the attackers. Thus, to prevent our systems from these attackers, we need to create smarter intrusion detection systems. In this paper, a new intelligent intrusion detection system has been proposed using genetic algorithms. In this system, at first, the network connection features were ranked according to their importance in detecting attack using information theory measures. Then, the network traffic linear classifiers based on genetic algorithms have been designed. These classifiers were trained and tested using KDD99 data sets. A detection engine based on these classifiers was build and experimented. The experimental results showed a detection rate up to 92.94%. This engine can be used in real-time mode.

KEYWORDS: Intrusion Detection Systems, Anomaly Detection, Genetic Algorithms.

1. INTRODUCTION

The need to accurately detect malicious intrusions has arisen. Intrusion detection is an increasingly important technology that monitors network traffic and recognizes illegal use, misuse of computer systems, and malicious attacks to computer systems. The difference between security products and intrusion detection systems is that the latter needs more intelligence. They must analyze all gathered information and deduce useful results.

Intrusion Detection Systems **(**IDS) are either signature based or anomaly based. Most of current IDSs are signature based and which they search a signature in the captured traffic or log files and send an alarm. This is similar to antivirus work methodology. Anomaly based IDSs measure the deviation in behavior of normal system state, and then sends an alarm if the deviation exceeds a certain threshold [1]. Besides, IDS can be classified as Network IDS (NIDS) or host based IDS (HIDS). NIDS deals with network traffic while HIDS deals with computer system logs and calls.

A. Genetic Algorithms

Genetic Algorithms (GA) are stochastic search techniques based on the mechanism of natural selection and natural genetics. The main components of a GA are genes, Chromosome and the Fitness function. If we want to search a solution for a problem within the solutions space, we first determine the problem features or attributes, these are called *genes*. Then we gather all genes in one string called *chromosome*. The *chromosome* represents the problem to be solved. Next the *chromosome* must be coded into binary, numerical, or nominal values, but the most famous is the binary coding. The *Fitness function* is very important measure to calculate the "goodness" of a *chromosome*; this goodness value is called the fitness value.

The process of a GA [2] usually starts with randomly selected chromosomes called the *population*. The aim of any GA based problem is to search for the best chromosome (solution). This is done by the evolution cycle as shown in Fig. 1.

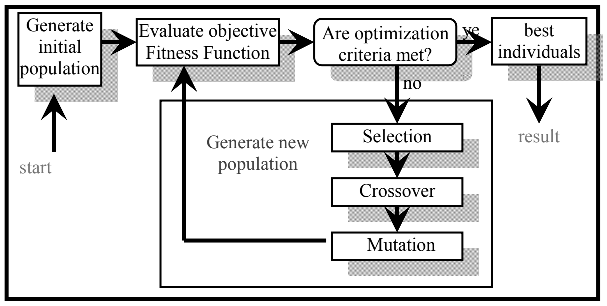
Generate initial population
Evaluate objective Fitness Function
Are optimization criteria met?
ye
best individuals
no
start
Generate new population
Selection
Crossover
Mutation
result

Fig. 1. Simple Genetic Algorithm structure [4]

2. BACKGROUND OF SIMILAR WORK

In [5] the authors have implemented a simple genetic algorithm which evolves weights for the features of the data set. Then k-nearest neighbor classifier was used for the fitness function of the GA as well as to evaluate the performance of the new weighted feature set. The main aim of work was to rank features according to their importance. The results provided an increase in intrusion detection accuracy.

Xiao et al. [6] present an approach that uses information theory and genetic algorithms to detect abnormal network behaviors. The information theory was used to determine the most relevant features to the detection operation. A small number of network features are closely identified with network attacks. However, this approach considers only discrete features and ignores other important features such as duration, source and destination bytes.

Chittur [7] designed a genetic algorithm that promoted a high detection rate of malicious behavior and a low false positive rate of normal behavior classified as malicious. The genetic algorithm was given “training data” from which an empirical model of malicious computer behavior was generated. This model was then tested over previously unseen data to gauge its real-world performance. The results were so good and the classification accuracy was 97.8%. Also these results are good but are biased towards the training dataset (KDD 10% training set), because the last rules extracted from the training phase have been tested on the rest (90%) of the KDD dataset, i.e. testing and training the dataset are from the same distribution. The author should test his results on the KDD unlabeled test dataset that contains 311029 records and is taken from different distribution

Wei Lu and Issa Traore [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

Wei Li, [2] proposed 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.

G. Jim, L.D and Cui [9] have developed a rule based classifier to filter the abnormal traffic, the classifier resultant shows a high detection rate if applied on the training 10%KDD99 dataset, but there is no indication about testing this classier on another dataset from another distribution like KDD99 test dataset, besides the resultant classier depends only on three features "service, host-count, host_srv_count" which is not sufficient for detecting diverse attacks.

3. KDD99 DATA SETS DESCRIPTION

The *KDD99* dataset [10] is now the benchmark for training, testing and evaluating learning IDSs, so it is the basis for IDS developers.

A. KDD99 Dataset Description

The dataset was used for the Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99, the Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model or a classifier that can tell what are "bad" connections, called intrusions or attacks, and what are "good", called normal connections.

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, and inserted multiple attacks. These attacks fall into four main categories:

- **DOS:** denial-of-service, e.g. syn flood.
- **R2L:** unauthorized access from a remote machine, e.g. guessing password;
- **U2R:** unauthorized access to local super user (root) privileges, e.g., various "buffer overflow" attacks;
- **Probing**: surveillance and other probing, e.g., port scanning.

The KDD99 datasets is composed of records that are called connections. Each connection has 41 features and a label which indicates the attack name. The label is supported only for the training dataset. Fig. 2. shows these features.

<u>**Basic Features**</u>
1 duration
2 protocol_type
3 service
4 flag
5 src_bytes
6 dst_bytes
7 land
8wrong_fragment
9 urgent

<u>**Content features**</u>
10 hot
11num_failed_logins
12 logged_in
13 num_compromised
14 root_shell
15 su_attempted
16 num_root
17num_file_creations
18 num_shells

<u>**Traffic features**</u>
19 num_access_files
20num_outbound_cmds
21 is_host_login
22 is_guest_login
23 Count
24 srv_count
25 serror_rate
26 srv_serror_rate
27 rerror_rate
28 srv_rerror_rate
29 same_srv_rate
30 diff_srv_rate
31 srv_diff_host_rate
32 dst_host_count
33 dst_host_srv_count
34 dst_host_same_srv_rate
35 dst_host_diff_srv_rate
36dst_host_same_src_port_rate
37dst_host_srv_diff_host_rate
38 dst_host_serror_rate
39 dst_host_srv_serror_rate
40 dst_host_rerror_rate
41 dst_host_srv_rerror_rate

Fig. 2. The 41 connection features of KDD99 dataset

B. Distribution of KDD99 Attacks

1.1.1.The KDD99 datasets are divided into three parts:

- Full training dataset which has 4898431 labeled records and is used for training purposes.
- 10% training dataset which has 494021 labeled records. Since the full dataset is huge an IDS developer can train his IDS on just 10% of the full dataset.
- Testing dataset. It has 311029 unlabeled connections.

1.2.1.Table 1 shows the distribution of these connections.

Table 1. Distribution Of Attack Classes

Class	10% KDD Training Dataset		Full Training Dataset		Testing Dataset	
normal	97278	19.69%	972781	19.86%	60593	19.48%
DoS	391458	79.24%	3883370	79.28%	229853	73.90%
Probe	4107	0.83%	41102	0.84%	4166	1.34%
R2L	1126	0.23%	1126	0.02%	16347	5.26%
U2R	52	0.01%	52	0.001%	70	0.02%

In the following, Table 2 shows the known and unknown attacks in both training and testing KDD99 datasets.

Table 2. Known and Unknown Attacks in Training and Testing datasets

Class	Known Attacks (in training + testing datasets)	unknown Attacks (in testing data set)
DoS	back, land, neptune (syn flood), pod (ping of death), smurf, teardrop	apache2, mailbomb, processtable, udpstorm
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster	httptunnel, name, sendmail, snmpguess, worm,xlock, xsnoop
U2R	buffer overflow, loadmodule, perl, rootkit	ps, sqlattack, xterm
PROBE	ipsweep, nmap, portsweep, satan	mscan, saint.

4. FEATURES SELECTION

Indeed it is very important to reduce the number of features by selecting the most important ones. For example in the problem of designing an intelligent intrusion detection system, developers are exposed to deal with the KDD99 data sets, the training dataset contains about 5 million connection records, each record contains 41 features. So it is time consuming to train the system considering all features. But if we could reduce the number of features for example to 8 features we will decrease very much of the amount of huge computations.

Our approach will employ the information theory to sort the most important features.

Indeed it is thought that the dataset space can be divided into:

- Broadly 2 main classes: Normal & Attack.
- Or more precisely 5 classes: Normal, DoS, R2L, U2R, Probe

So the dataset space can be described by the following random variables:

X: the decision random variable. Its state space is {Normal, DOS, U2R, R2L, and Probe}

Y: The connection feature random variable. Indeed there are 41 independent random variables like

- $Y = Y_{\text{Protocol\_type}} = \{\text{ICMP, TCP, UDP}\}$
- $Y = Y_{\text{land}} = \{0, 1\}$

Now to extract the importance of a connection feature (ex. ptotocol_type), one can calculate **"the amount of information about X (normal connection or DOS or R2L or U2R or Probe) contained in Y (connection feature ex. protocol_type)"**

Indeed the previous statement is exactly the mutual Information [1] of X and Y that are explained in I-A.

A. Case Study: Information about Attacks Contained In Protocol_Type Feature.

In the following comes a case study to give an explanation about mutual information is calculated for X {normal, Attack} and Y {UDP, TCP, ICMP}.

<u>*Step1:*</u> we calculate the distribution of connections of X and Y using the KDD99 training Full dataset as shown in Table 3.

Table 3. Conn# according to protocol_type and traffic Class

		$X_{\text{random variable}}$					
		Normal	**DoS**	**R2L**	**U2R**	**PROBE**	**Total**
$Y_{\text{protocol\_type}}$	**ICMP**	12763	2808150	0	0	12632	2833545
	TCP	768670	1074241	1126	49	26512	1870598
	UDP	191348	979	0	3	1958	194288
	TOTAL	972781	3883370	1126	52	41102	*4898431*

Then we calculate the probability distribution of these connections as shown i Table 4.

In Table 4, the upper probability value in italic is the intersection probability P(X, Y). The lower probability value is the conditional probability P (X|Y) = P(X,Y) / P(Y).

Table 4. Probability and conditional probability of X and Y=Yprotocol_type

		X random variable					
		P_{normal}	P_{DoS}	P_{R2L}	P_{U2R}	P_{PROBE}	P(Y)
$Y_{protocol\_type}$ r.v	P_{ICMP}	*0.0026*	*0.5733*	*0*	*0*	*0.0026*	**0.5785**
		0.0045	0.9910	0	0	0.0045	
	P_{TCP}	*0.1569*	*0.2193*	*0.0002*	*1.0003E-05*	*0.0054*	**0.3819**
		0.4109	0.5743	0.0006	2.6195E-05	0.0142	
	P_{UDP}	*0.0390*	*0.0002*	*0*	*6.1244E-07*	*0.0004*	**0.0396**
		0.984	0.0050	0	1.5441E-05	0.0101	
	P(X)	**0.6140**	**0.3581**	**0.0008**	**3.6811E-05**	**0.0270**	

Step2: Let's calculate the entropy H(X) of X which means the uncertainty of X (normal or attack)

$$H(X) = -P_{normal}\log_2 p_{normal} - p_{attack}\log_2 p_{attack}$$

$$where\ p_{attack} = P_{DoS} + P_{R2L} + P_{U2R} + P_{PROBE} = 1 - P_{normal}$$

H(X) = -0.614 log (0.614) - 0.3851 log (0.3851) = 0.9622

Step3: Let's calculate the conditional entropy H(X, connection feature) of X which means the uncertainty of X (normal or attack) given that the connection feature is protocol_type.

H(X|$Y_{protocol\_type}$)=P(icmp){H(normal|icmp)+H(attack |icmp)}+p(tcp){H(normal|tcp)+H(attack|tcp)}+P(udp){ H(normal |udp)+H(attack |udp)}

H (normal |icmp) =-p (normal |icmp) log P (normal |icmp) H(X |protocol_type) = 0.5785 {-0.0045 log (0.0045) -0.9955 log (0.9955)} +0.3819 {-0.4109 log (0.41092)-0.5891 log (0.5891)} +0.0397{-0.9849 log (0.9849) - 0.0151 log (0.0151)} = 0.4016

So when protocol_type is introduced, the Uncertainty decreased from 0.9622 to 0.4016.

Step4: calculating the mutual information of X and Y

I(X, Y) =H(X)– H(X|protocol_type) =0.9622-0.4016 = 0.5606

So 0.5606 is the amount of information about
X {normal or attack} contained in protocol_type

In the same way one can get the I(X,Y=Yprotocol_type) for X={DoS, Others}, {R2L, Others}, {U2R, Others}, {PROBE, Others}.

B. All Features Ranking

In this case study we have studied the role of protocol_type feature in classifying a connection into one of the following classes {Normal, DoS, U2R, R2L, and PROBE}. Now we have to generalize previous calculation for the 41 features. So Table III, Table 4 and mutual measures I(X,Y_{fi}) will be recalculated 41 times. The resulting calculations will be organized in a table like Table 5.

Table 5. MUTUAL information TABLE

Feature	Normal /attack	DoS/ Others	R2L/ Others	U2R/ others	probe/ others
Duration	0.052897	.0577	0.004084	0.0008	0.0014
Protocol_type	0.304063	.3046	0.003147	0.00013	0.00180
Service	0.570986	.5999	0.014867	0.00079	0.0341

Now to rank features according to their importance we sort each column in descending order. Table VI shows the ranking results. The feature name was substituted with its number according to Fig. 2.

C. Data Preprocessing And Normalization

Each record in the KDD99 data sets contains 41 features that belong to different range values as shown in the example in Table 6.

Table 6. Features values differences

Feature name (f_i)	Type	Values example	Max
Duration	Continous	565,255	58329
Dst_Bytes	Continous	125454, 45454	1309937401
Same_Srv_Rate	Continous	0.23, 0.65	1
Protocol_type	Discrete, Text	Tcp, Icmp, Udp	×
Flag	Discrete, Text	SF, REJ, S1	×

So normalization is needed to put all features in a homogenous value space. The following normalization measure is suggested:

$$f_i^{'} = normalized_{f_i} = I(X, Y_{f_i})\ \frac{f_i}{\max(F_i)}$$

- Where *I(X,Y_{fi})* is the mutual information contained feature f_i (or amount of information about attack contained in f_i)

- Max(F_i) is the maximum value of the feature f_i in the complete KDD99 training dataset (4898431 connections) and is defined as the following:
  - If $f_i \neq 0$ then we take the maximum over the complete data set.
  - If $f_i = 0$ then $Max(F_i)=1$
  - If f_i is text like protocol_type, flag, service then we sort feature values in ascending order according to their probability distributions, then we assign serial numbers to them from 1 to last value. After then we normalize according to relation before. The next example shows how to normalize text features.

Ex. normalizing protocol_type feature: Table 7 shows how to normalize text values by sorting in descending order then numbering and then dividing by max value.

Table 7. Protocol_type feature normalization

Protocol_type	records		sorting	Numbering	Dividing by max	f'_i
ICMP	2833545	➔	UDP	1	1/3	0.1013
TCP	1870598		TCP	2	2/3	0.2027
UDP	194288		ICMP	3	3/3	0.3041

In the previous example textual features are not only normalized between [0,1] but are given an importance weight according to their distribution in the KDD connections so:

$$f_i \text{ is more important than } f_j \Leftrightarrow I(X,Y_{f_i}) > I(X,Y_{f_j})$$

All KDD99 or other traffic will be now passed through the "Normalization & Preprocessing Unit". The output will be considered the new KDD99 dataset.

5. GA BASED ANOMALY IDS

In this section some anomaly models will be introduced, all these and other models will be evaluated according to performance measures.

A. Algorithm Performance Measures

To evaluate how much the detection algorithm is good, the following measures must be calculated (indeed *FPR, DR, and NDR are the most important ones):*

- *TP (True Positive): Number of connections that were correctly classified as Attack.*
- *TN (True Negative): Number of connections that were correctly classified as Normal.*
- *FP (False Positive): Number of normal connections that were classified as Attack.*
- *FN (False Negative): Number of attack connections that were classified as Normal.*
- *TPR = DR:* Detection Rate= $\frac{TP}{TP+FN}$
- *NDR* : Detection Rate of new attacks

$$NDR = \frac{nb\ of\ New\ correctly\ \det ected\ attacks}{number\ of\ all\ new\ attack}$$

Since the KDD99 testing dataset contains new attacks that were not included in the training data set. NDR can be called the "algorithm cleverness".

- *TNR:* True Negative Rate $= \frac{TN}{TN+FP}$
- *CR* **:**Classification Rate $= \frac{TP+TN}{sizeof\ data\ set}$
- *FPR***:** False Positive Rate $= \frac{FP}{FP+TN}$
- *FNR***:** False Negative Rate $= \frac{FN}{FN+TP}$

B. GA Classifier

Our genetic classifier is named GACL and has the following form:

$$GACL(conn_j) = \begin{cases} if \ \sum_{i=1}^{n} W_i f_i(conn_j) < threshold \ \ then \ \ conn_j \ \ is \ \ ATTACK \\ \\ else \ \ conn_j \ \ is \ \ NORMAL \end{cases}$$

Where $\mathbf{conn_j}$ = $\mathbf{connection_{j=}}$ (duration, service, src_bytes, dst_bytes, count) $_\mathbf{j}$

The inequality < was chosen since for the previous features most of attack connections in the KDD training set have low values. For example, the KDD training dataset contains 396743 attack records, 396083 of them have a duration value equals to 0.

Parameters Coding:

The GACL parameters are encoded in the following way:

- The *GACL* has six parameters:
  - 5 parameters represent the weights of features *(W_1,W_2,W_3,W_4,W_5)*
  - 1 parameter represents the *threshold* value.
- Each *parameter* has 11 bits, except the Threshold which has 10 bits:
  - 10 bits for the parameter value
  - One bit for the sign.
- The 10 bits (parameter value) was chosen to have a precision of $2^{10} \approx 0.001$.
- The Threshold always holds a positive value
- As an example, one can gets the value of Gene1 as the following:
  - Gene1= 11000000010 = 1 & 1000000010
  - The 11th bit = 1 so the sign is negative (if the 11th bit = 0 then the sign is positive).
  - The decimal value of 1000000010 = $1\times2^9+0\times2^8+0\times2^7+0\times2^6+0\times2^5+0\times2^4+0\times2^3+0\times2^2+1\times2^1+0\times2^0=514$

- So Gene1= 11000000010 = -514

Table 8.a shows the chromosome genes.

Table 8.a. Parameters Coding

Genes	*gene*	*Gene borders*	example	
			Binary representation	**Gene decimal**
Gene1	$W_{duration}$	[-1023,1023]	11000000010	-514
Gene2	$W_{service}$	[-1023,1023]	01000000010	514
Gene3	W_{src_bytes}	[-1023,1023]	00000000010	2
Gene4	W_{dst_bytes}	[-1023,1023]	11111111111	-1023
Gene5	W_{count}	[-1023,1023]	01111111111	1023
Gene6	***Threshold***	**[0,1023]**	**0101011010**	**532**

The chromosome of the previous genes is

Gene6	***Gene5***	***Gene4***	***Gene3***	***Gene2***	***Gene1***
0101011010	01111111111	11111111111	00000000010	01000000010	11000000010
532	1023	-1023	2	514	-514

So the GACL of this chromosome is:

$$GACL(conn_j)=\begin{cases} if\ -514f_{duration}+514f_{service}+2f_{src\_byte}-1023f_{dst\_byte}+1023f_{count}<532 \\ then\ conn_j\ \ is\ ATTACK\ else\ conn_j\ \ is\ NORMAL \end{cases}$$

Data Structure:

The data structure of a classifier is shown in Fig.3.

```
Type INDIVIDUAL     // individual structure
   crom (1..gensNb*genLength) As Byte
        // classifier or GACL ex.  110101010000100110001100
   decCode (1. . gensNb) As Integer
        // decimal values of genes ex.  (512  340  211 -20 -10 20)
   fitVal As Double
        // the fitness function value  of the chromosome
   fitProb As Double  // the probability of fitness value
   reFit As Boolean  //indicates if it is necessary to recalculate Fitness
   TP As Long  // attack connections detected by this chromosome
   TN As Long // normal connections discovered by chromosome
   FP As Long  // False positive connections by chromosome (filter)
   FN As Long //False negative connections by chromosome (filter)
```

Fig. 3. Individual data structure

$$Where\ \ fitprob_i = \frac{fitVal_i}{\sum_1^{population\,size} fitVal} \qquad i = chromosom_i$$

Fig.4 shows the generation data structure

```
Type generation    // generatiopn structure
   pop (1..populationSize) As INDIVIDUAL
      // set of  classifiers or chromosomes in the generation
   sumFitness  As Double
      //generation strength=sum of fitness of  population chromosomes
   BestIdv As Integer        // best individual
   WorstIDV As Integer    // worst individual
   genrNb As integer        // generation number
```

Fig. 4. Individual data structure

When evolution operation starts the whole results will be stored in a database according to previous data structure, this enables to extract and build many useful statistics about the found solutions.

Fitness Function:

The fitness function explained in Fig.5 is the same as the classification rate*:*

fitVal or fitnessValue = CR

Since every chromosome in the population is a solution (classifier), its CR value will be calculated according to the KDD10% training dataset which contains the connection features and labels (normal, attack).

```
Function GetFitnessValue
   Input : (individual , training set)
   // individual contains 6 genes = W1,W2,W3,W4, W5, Threshold
   // training set = normalized & preprocessed KDD10%
  Output:(fitVal= individual fitness value)
   for  i = 1 to sizeof ( training dataset)   // conn_i  is a record from training set
      if $\sum_{j=1}^{4} W_j f_j(conn_i) < threshold$   then  // conn_i classified as an attack
         if  conn_i.label = "attack" then     //conn is realy an attack
               individual.TP = individual.TP + 1
         else         //conn is normal in the training set
               individual.FP = individual.FP + 1  //conn is normal
         end if
      else  // conn is classified as normal
         if  conn_i.label = "normal" then    //conn is realy an attack
               individual.TN = individual.TN + 1
         else       //conn is normal in the training set
               individual.FN = individual.FN + 1    //conn is normal
         end if
      end if
   next

return  $fitVal = individual.fitVal = CR = \frac{individual.TP + individual.TN}{sizeof(training dataset)}$
End function
```

Fig. 5. Structure of the Fitness function

GA Operations:

Our approach uses "two-point crossover" technique and "Bitwise bit-flipping" as a mutation method.

The following is a brief presentation about the genetic operations used in proposed learning GA, these operations are: Selection, Crossover, and mutation.

The "Roulette Wheel" selection method was adopted because it is easy to implement and converge quickly but a drawback is that if for example an individual fitness is 90% there is a low chance for other individuals to be selected. The main idea of this selection method is that better individuals get higher chance and chances are proportional to their fitness value. To implement this, each individual was assigned to a part of the roulette wheel (slice) and then one can spin the wheel to select an individual. The following example explains this strategy.

Selection using "Roulette Wheel" example

Given that the population of the generation is as in Table 8.b.

Calculate the sum of all fitness values for all individuals in the population and then calculate the slice value (fitness probability) of each individual. Now the roulette wheel will be constructed as shown in Fig.6.

Table 8.b. Population example

Individual Nb	Chromosome	Fitness Value	Fitness Probability
Indiv 1	10101110100010	0.60	24.00%
Indiv 2	00101000100110	0.12	4.80%
Indiv 3	00001110111110	0.97	38.80%
Indiv 4	10000110111110	0.80	32.00%
Total		2.5	100 %

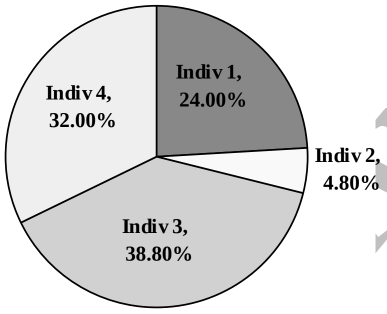

Fig. 6. Roulette wheel according to Fitness probability

To spin the wheel we select a random number between 0 and 1 for example 0.3451 = 34.51% then we compute the sum of individual's fitness value until reaching to 0.3451.

24.00 % + 4.8 % < 34.51% < 24.00 % + 4.8 % + 38.8 %

So the selected spice (parent) is the individual 3. This is predictable since individual 3 has the largest probability. In the same way you can select the second parent by reconstructing the wheel after eliminating individual 3.

This strategy is good because the strongest parents will generates more children.

Running a GA To Get GACL Parameters:

Figure 7 represents the learning genetic algorithm to determine the GACL parameters. The following is the detection algorithm which determines the 6 parameters to build the network traffic classifier.

Where HighCR is High Classification Rate and it represents the stop criteria, it may be for example in the range [0.975,1]. If this detection rate hasn't been reached then the GA will stop when the predefined generation's number is reached. At the end of the previous algorithm, the chromosome that has the largest HighCR will be selected and considered as the problem solution (classifier).

Experimental Results:

- *Best Individual and Classifier:* The proposed GA in Fig. 8 was executed using the following parameters:

Chromosome length = 66
Generations number=200
Training set=KDD10% training set
Genes Number = 6
mutation propability Pm=0.02

features selected =1,3,5,6,23
Population Size=500
Crossover Probability Pc=0.7
HighCR >=97.50 %

```
CreateInitialRandomPopulation ⇨ CURRENT_GENERATION
i=0
While   i <= generationsNumber   OR
        CURRENT_GENERATION.BestChrom.fitValue >= HighCR
  - For each individual in CURRENT_GENERATION calculate its fitness
    using the function GetFitnessValue
  - For each individual in CURRENT_GENERATION Calculate the
    fitness probability // this step is necessary for roulette wheel selection
  - Repeat the following steps until n offspring have been created:
                                        // n = population size
      o Parent1 <- rouletteWheel selection form
          URRENT_GENERATION
      o Parent2 <- rouletteWheel selection form
          {CURRENT_GENERATION - Parent1}
      o Cross over the two parents (2-point crossover) with probability
          Pc to form two offspring. (If no crossover occurs, the
          offspring are exact copies of the parents.) Select one of the
          offspring at random and discard the other.
      o Mutate each bit in the selected offspring with probability Pm, and
          place it in the NEW_GENERATION
  - Store NEW_GENERATION in a database // this step is optional but
    useful to restart the GA from this point
  - CURRENT_GENERATION = NEW_GENERATION
  - i=i+1
end while
```

Fig. 7. Genetic Algorithm to build Network traffic classifier

The best chromosome found was:

0011101000111110010101100011000000111011100101101001011 00001100111

Table 10 shows a description of the individual of this chromosome. Of course this chromosome is the classifier aim of the search.

Table 9. Best Individual for our GACL

GACL parameters (Fitness = 0.97971)	Threshold	Weights of Network Connection Features				
		W_{count}	W_{dst_bytes}	W_{src_bytes}	$W_{service}$	$W_{duration}$
Binary Chrom	00111010001	11110010101	10001100000	01110111001	01101001011	00001100111
Coded Chrom	465	-917	-96	953	843	103

Table 10. Linear Genetic Algorithm classifier performance measures

Class	Records	Detected	Rate	
			DR	NDR
NORMAL	60593	58330	96.27%	-
DOS	229853	222142	96.65%	16.34%
PROBE	4166	3227	77.46%	92.34%
R2L	16347	4953	30.30%	0.40%
U2R	70	53	75.71%	77.42%
TOTAL	**DR**=91.99% **NDR**=14.89% **CR**=92.82% **FP**=3.73% **FN**=8.01%			

So the Network Anomaly Intrusion Detection classifier is

$$GACL(conn_j) = \begin{cases} if\ \ 103 f_{duration} + 843 f_{service} + 953 f_{src\_byte} - 96 f_{dst\_byte} - 917 f_{count} < 465 \\ then\ \ conn_j\ \ is\ \ ATTACK\ \ else\ \ conn_j\ \ is\ \ NORMAL \end{cases}$$

Classifier 1

- *Discussion:* the previous classifier gives a classification rate equals to 97.971% if applied to the training data set. This classifier was applied to detect attacks in the KDD99 test dataset; and following results given in Table 11 were found.
- *GACL Detection Rates for all types of attacks:* Table 12 shows our system efficiency for all attack types and for both known types of attacks that were included in the KDD10% training dataset and for known and unknown attacks that are included in the KDD99 test data set.

GACL has detected all types of known and unknown attacks except *"phf"* and *"phf"* attack from the known group and *"mailbomb", "udpstorm" and "snmpgetattack"* from the *unknown attack* group.

C. Multi GACls Anomaly IDS (GACAL ANOMALY IDS)

The previous technique based on GACL can be improved by designing a detection engine with multi classifiers. Figure 8 shows the structure of this engine.

- ***Classifier1*** detects anomaly behavior assuming that attacks are in the lower area. Indeed it is assumed that if the classifier value is less than a threshold (< 465) then the connection is an attack, so there are some attacks that are in the upper area that were not detected.
- ***Classifier 2*** detects attacks in the upper area. The same learning genetic algorithm (in *VII.5)* was launched by changing the inequality condition.

$$GACL(conn_j) = \begin{cases} if\ \ 44 f_{duration} + 508 f_{service} + 710 f_{src\_byte} + 325 f_{dst\_byte} - 912 f_{count} >= 259 \\ then\ \ conn_j\ \ is\ \ ATTACK\ \ else\ \ conn_j\ \ is\ \ NORMAL \end{cases}$$

Classifier 2

Table 11. Attacks detection rates using GACL

	attackClass	attack_name	Total records	Detected	Detection Rate
KNOWN ATTACK	DOS	back	1098	0	0.00%
		Land	9	1	11.11%
		Neptune	58001	57075	98.40%
		Pod	87	6	6.90%
		Smurf	164091	163986	99.94%
		Teardrop	12	3	25.00%
	PROBE	Ipsweep	306	3	0.98%
		Nmap	84	4	4.76%
		Portsweep	354	23	6.50%
		Satan	1633	1545	94.61%
	R2L	ftp_write	3	2	66.67%
		guess_passwd	4367	4367	100.00%
		Imap	1	1	100.00%
		Multihop	18	7	38.89%
		Phf	2	0	0.00%
		Warezmaster	1602	535	33.40%
	U2R	buffer_overflow	22	18	81.82%
		Loadmodule	2	2	100.00%
		Perl	2	2	100.00%
		Rootkit	13	7	53.85%
UNKNOWN ATTACK	DOS	apache2	794	312	39.29%
		Mailbomb	5000	0	0.00%
		Processtable	759	759	100.00%
		Udpstorm	2	0	0.00%
	PROBE	Mscan	1053	1053	100.00%
		Saint	736	599	81.39%
	R2L	Httptunnel	158	2	1.27%
		Named	17	11	64.71%
		Sendmail	17	11	64.71%
		Snmpgetattack	7741	0	0.00%
		Snmpguess	2405	2	0.08%
		Worm	2	2	100.00%
		Xlock	9	9	100.00%
		Xsnoop	4	4	100.00%
	U2R	Ps	16	12	75.00%
		Sqlattack	2	2	100.00%
		Xterm	13	10	76.92%

Table 12. Comparison BETWEEN RANDOM classifier and GA based chaffier

Class	RANDOM CLASIFIER	OUR GACL
NORMAL	88.61%	96.27%
DOS	96.83%	96.65%
PROBE	62.48%	77.46%
R2L	0.1%	30.30%
U2R	0	75.71%
	TOTAL	
TP=DR	89.92%	91.99%
CR	89.67%	92.82%
FP	11.39%	3.73%

- ***Classifier 3***: detects more PROBE attacks in the upper area. The GA was trained to find the best weight for top 5 probe features found in Table VI. After running this GA detector, the following classifier was found:

$$if\ 1020f_{service} - 191f_{src\_bytes} + 989f_{rerror\_rate} - 832f_{dst\_host\_srv\_count} + 898f_{dst\_host\_diff\_srv\_rate} >= 782$$

$$then\ conn_j\ is\ ATTACK\ else\ conn_j\ is\ NORMAL$$

Classifier 3

The paper's GA was also launched using selected features for U2R and R2L features and it was found that the results were not encouraging for the training KDD dataset so no more classifiers were included in the anomaly detection engine. Table 13 shows the results of this engine with a comparison with previous approaches when applied to the KDD test dataset.

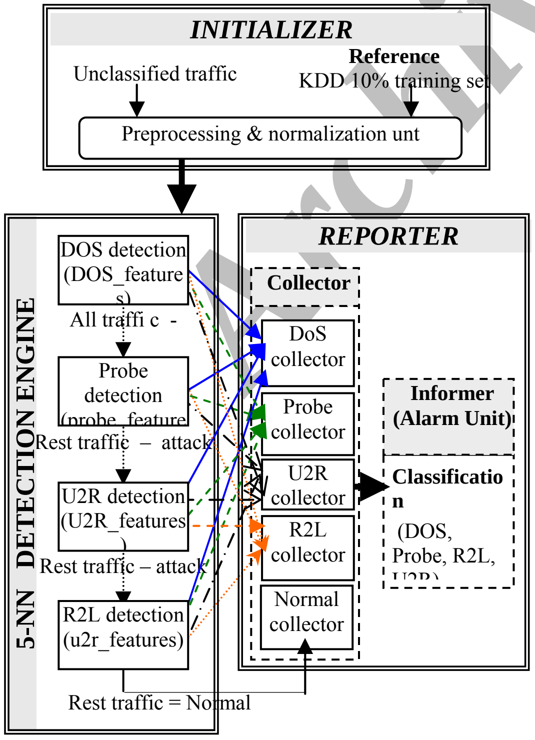

Fig. 8. "GACL ANOMALY IDS" based on 3 genetic algorithms classifiers

Table 13. Results of our GA based Detection engine

Class	Classifier1	GACL Anomaly IDS
NORMAL	96.27%	96.14%
DOS	96.65%	96.68%
PROBE	77.46%	85.77%
R2L	30.30%	30.30%
U2R	75.71%	75.71%
	TOTAL	
TP=DR	91.99%	92.16%
NDR	14.89%	14.95%
CR	92.82%	92.94%
FP	3.73%	3.86%

It is seen that there is a serious improvement in detecting probe attack and a little improvement in detecting DoS attacks but the FP was lightly increased.

D. *Criticizing Using GAs In The Detection:*

- *Does GA* decrease *the time to find a classifier?* Of course it does. In this problem the chromosome length is 6*11=66 so there is 2^{66} solutions (filters). While GA found a solution after 160 generations, so it has tested 160 × 500 = 80000 solutions instead of 2^{66} which is a huge save in time. This GACL may be not the best solution but the results were satisfying.

- *Does GA* differ *from random treatment?* In this problem one may wonder that GA technique doesn't differ from random solutions. To ensure or not this interrogation a random filters (chromosomes) has been generated. The number of chromosomes is equal to those generated by the GA when the best one has been found:

Number of generations × *population size= 160 × 500 = 80000 random filters*

These filters have been applied to the KDD99 10% training set and the filter with the best classification rate was selected. Then this best filter was applied to the KDD test dataset and the following results presented in Table 14 were found.

These results support this claim that GA is not just a fancy form of random solutions.

6. COMPARING APPROCHES AND GATHERING ALL ANOMALY MODELS

A. *Comparing Approaches*

Table 15 and Fig.9 show a comparison between this approach and other ones [11]:

It is observed that this approach performs better, especially for detecting R2L attacks.

B. Gathering All Anomaly Models

All previous approaches (this approach + others) can be programmed into one package which may be called "***Anomaly IDS package***" as shown in Fig.11a. Then it is up to the user to select one anomaly detection method

Then to position the "Anomaly IDS package", there are two possibilities:

1. Standalone: put the package immediately on the network. Of course firstly the following is needed:
   - o sniffer or *TCPDUMP* software
   - o tcpdump_to_kdd99 function to convert formats.

2. After a signature based IDS like SNORT. In this case SNORT or another will alarm for bad signatures and then the resident traffic which is classified as normal will be send to tcpdump_to_kdd99 transformer and then will be processed by "*Anomaly IDS package*".

Table 14. Comparing our approach to design anomaly IDS with other's

	Our IDS s		Other's IDSs										
CLASS	GACL1	GACL ANOMALY IDS	5NN [11]	Other's LGA classifier [6]	Runner up of KDD [12]	RSS-DSS [13]	MOGF-IDS [11]	PNrule [14]	CTree [6]	C5 [6]	C4.5 [11]	SVM [11]	Cup winner [15]
NORMAL	96.27	96.14	95.89	98.34	99.4	96.5	98.36	99.5	92.78	99.5	98.38	97.99	99.5
DOS	96.65	96.68	97.00	99.33	97.5	99.7	97.20	96.9	98.91	97.1	96.99	97.56	97.10
R2L	30.3	30.3	6.90	5.86	7.3	31.2	11.01	7.3	7.41	8.4	1.45	3.55	8.40
U2R	75.71	75.71	14.91	63.64	11.8	76.3	15.79	11.8	88.13	13.2	14.47	10.09	13.20
PROBE	77.46	85.77	81.61	93.95	84.5	86.8	88.6	84.5	50.35	83.3	81.88	81.61	83.30

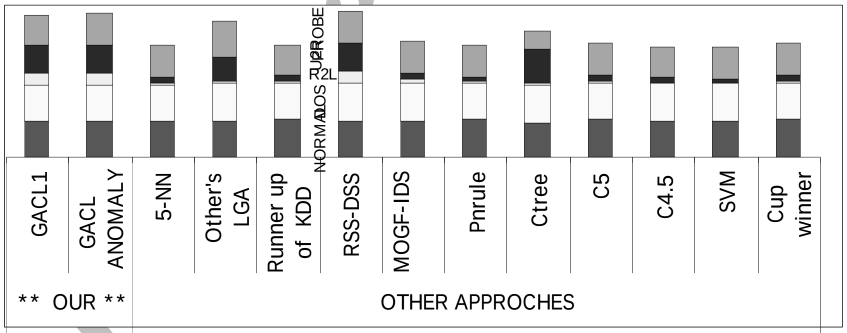

Fig. 9. Detection Rate Comparison

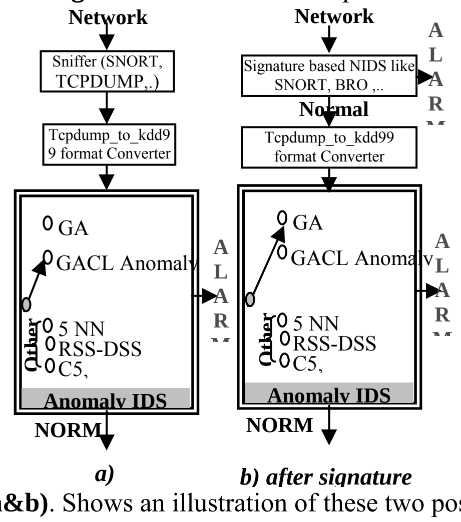

Fig. 20(a&b). Shows an illustration of these two possibilities.

7. CONCLUSION

By applying the information theory measures like entropy and mutual information, 41 connection features were ranked after the normalization process according to each attack class. This ranking allows decrement in the computing complexity by selecting the most important features for each attack class. Features selection proved that they dramatically decreased the detection speed without affecting the detection rates.

Using GA, a linear classifier was designed that uses the top five features according to their importance in detecting attack using information theory measures. The classification rate was 92.82% and, the new detection rate was 14.89% which is significantly better than any other approach presented before. But the best detection rate was for detecting R2L attacks (30.30%) knowing that R2L attacks are hard to detect.

Later, multi GACLs anomaly intrusion detection engine was proposed. This engine, based on the selected features for each attack classes, consists of three classifiers; two of them are employed to detect DoS attacks in the upper and lower area of the dataset space, and one to detect more PROBE attacks. This engine proved that it detects more DOS and PROBE attacks.

GA based detection models were better than C5, C4.5, cupWinn, K-NN based models and others, especially in R2L attacks. The great gain is that they are so fast since each connection will be classified in no more than three steps. So GA based classifiers can be used in real-time mode. But the long training time still is the drawback of this methodology.

A package called "Anomaly IDS package" was proposed to detect anomaly attack in network traffic. This package gathers all IDSs proposed in the paper and others. The package functions standalone or with a traditional signature based IDS like SNORT and BRO.

Finally, one can see that all previous proposed models in this paper are modularized so that new learning algorithms are easy to be added in and tested or even inserted as a new module. This well suits the dynamics in the research world.

REFERENCES

[1] Papoulis and Pillai S.U.; **Probability, Random Variables and Stochastic Processes**, book, (2002)

[2] Wei Li; **Using Genetic Algorithm for Network Intrusion Detection**, SANS Institute, USA, (2004)

[3] Melanie M.; **An Introduction to Genetic Algorithms,** Cambridge, Massachusetts London, England, Fifth printing, (1999)

[4] Hartmut P.; **Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms** Genetic and Evolutionary Algorithm Toolbox. Hartmut Pohlheim, (2003)

[5] Middlemiss M.J. and Dick G.; **"Weighted Feature Extraction Using a Genetic Algorithm for Intrusion Detection**", *Evolutionary Computation*,Vol. 3 pp. 1699 - 1675, (2003)

[6] Qu X., Hariri S. and Yousif M.; **"An Efficient Network Intrusion Detection Method Based on Information Theory and Genetic Algorithm"**, Proceedings of the 24th IEEE International Performance Computing and Communications, (2005)

[7] Chittur; **"Model Generation for an Intrusion Detection System Using Genetic Algorithms"**, http://www1.cs.columbia.edu/ids/publications/gaids-paper01.pdf, (2005)

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

[9] Jim G., Da-xin L. and in-ge C.; **"An Induction Learning Approach for Building Intrusion Detection Models Using Genetic Algorithms"** , Proceedings of the 5Ih World Congress on Intelligent Control and Automation, (June 15 - 19 2004)

[10] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[11] Tsang, S. Kwong and Wang H.; **Anomaly Intrusion Detection using Multi-Objective Genetic Fuzzy System and Agent-based Evolutionary Computation Framework,** Proceedings of the Fifth IEEE International Conference on Data Mining, (2005)

[12] Levin; "**KDD-99 Classifier Learning Contest LLSoft's Results"**, *Overview. SIGKDD Explorations. ACMSIGKDD*, Vol. 1, No. 2, pp. 67 – 75, (2000)

[13] Song, Heywood M.1. and Zincir-Heywood A.N.; "**Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection**", *IEEE Transactions on Evolutionary Computation*, Vol. 9, No. 3, (2005)

[14] Agarwal R.and Joshi. M.V.; **"Pnrule: A New Framework for Learning Classifier Models in Data Mining"**, Department of Computer Science, University of Minnesota, Report No. RC-21719, (2000)

[15] Elkan; **Results of the Kdd'99 Classifier Learning**, ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Boston, MA, Vol. 1, No. 2, (2000)