An Ensemble-based Evolutionary Framework for coping with Distributed Intrusion Detection

Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano
Institute for High Performance Computing and Networking (ICAR)
Via P. Bucci 41c, I-87036 - Rende (CS), Italy
E-mail:(folino, pizzuti, spezzano)@icar.cnr.it

Abstract

A distributed data mining algorithm to improve the detection accuracy when classifying malicious or unauthorized network activity is presented. The algorithm is based on genetic programming (GP) extended with the ensemble paradigm. GP ensemble is particularly suitable for distributed intrusion detection because it allows to build a *network profile* by combining different classifiers that together provide complementary information. The main novelty of the algorithm is that data is distributed across multiple autonomous sites and the learner component acquires useful knowledge from this data in a cooperative way. The network profile is then used to predict abnormal behavior. Experiments on the KDD Cup 1999 Data show the capability of genetic programming in successfully dealing with the problem of intrusion detection on distributed data.

1 Introduction

The extensive use of Internet and computer networks, besides the known advantages of information exchange, has provided an augmented risk of disruption of data contained in information systems. In the recent past, several cyber attacks have corrupted data of many organizations creating them serious problems. The availability of *Intrusion Detection Systems(IDS)*, able to automatically scan network activity and to recognize intrusion attacks, is very important to protect computers against such unauthorized uses and make them secure and resistant to intruders. The task of an IDS is to identify computing or network activity that is malicious or unauthorized. An IDS is essentially an alarm system for the network. It enables the monitoring of the network for discovering intrusive activity. When an intrusive activity occurs, the IDS generates an alarm to notify that the network is

possibly under attack. An IDS, however, can generate ”false positives” or ”false alarms”. So, the main focus of an intrusion detection system is to design accurate algorithms that characterize the behaviors of individual nodes of the system by recognizing abnormal behaviors, yet minimizing false alarm rate.

Most current Intrusion Detection Systems collect data from distributed nodes in the network and then analyze them centrally to build a profile defining the normal user activity. The main drawback of this approach is the rise of security and privacy problems due to the necessity of transferring data. Moreover, if the central server becomes the objective of an attack, the whole network security is quickly compromised.

An alternative solution is to build a network profile by applying distributed data analysis methods. Distributed *profile-based* intrusion detection systems (dIDS) offer an alternative to centralized analysis, may afford greater coverage and provide an increase in security.

In this paper we propose ($GEdIDS$, *Genetic programming Ensemble for Distributed Intrusion Detection Systems*) a distributed data mining algorithm based on the ensemble paradigm that employs a Genetic Programming-based classifier as component learner in order to improve the detection capability of the system [12]. Ensemble paradigm is particularly suitable for distributed intrusion detection because it allows to build a network profile by combining different classifiers that together provide complementary information. The classifiers perform the necessary analysis of data at the locations where the data and computational resources are available and transmit the result of the analysis to the locations where the ensemble is needed. The idea is that combining together different classifiers, they yield a better performance than the individual classifiers. The network profile is then used to predict abnormal user activity. GP ensembles are built using a distributed cooperative approach based on a hybrid model [3] that combines the island model with the cellular model.

Each node of the network is considered as an island that contains a learning component, based on cellular genetic programming, whose aim is to generate a decision-tree predictor trained on the local data stored in the node. Every genetic program, however, though isolated, cooperates with the neighboring nodes by collaborating with the other learning components located on the network. Knowledge about the solutions found so far is diffused by taking advantage of the cellular model that exchanges the outermost individuals of the population.

A learning component employs the ensemble method AdaBoost.M2 [14], thus it evolves a population of individuals for a number of rounds, where a round is a fixed number of generations. Every round the islands import the remote classifiers from the other islands and combine them with their own local classifier. Finally, once the classifiers are computed, they are collected to form the GP ensemble.

In the distributed architecture proposed, each island thus operates cooperatively yet independently from the others, providing for efficiency and distribution of resources.

To evaluate the system proposed, we performed experiments using the network records of the KDD Cup 1999 Data [1]. Experiments on this data set point out the capability of genetic programming in successfully dealing with the problem to improve the detection accuracy.

The main contributions of the paper can be summarized as follows.

A data mining algorithm based on a distributed Genetic Programming model and ensemble learning for improving detection accuracy is presented.

The algorithm runs on a distributed environment and does not require that audit data present on each node of the network be collected and sent to a central location to be analyzed.

The network nodes work with their own population, their own data set and train the classification algorithm on the local data there contained. However, they cooperate by communicating each other the computed models. This strategy allows for the realization of an efficient co-evolutionary and cooperative model of GP since the islands communicate models and not data.

A main advantage of co-evolution and cooperation is the generation of classification trees of smaller size and an augmented generalization capability of the ensemble in predicting the class on new data.

The distributed approach, on the other hand, gives significant advantages in flexibility, extensibility, and fault tolerance since, if a node is temporarily unavailable, the other nodes can continue to work by using the information coming from the currently available nodes.

The paper is organized as follows. The next section gives a brief description of Intrusion Detection Systems. Section 3 outlines the genetic programming based ensemble paradigm for distributed IDS. Section 4 describes the algorithm and its implementation on a distributed environment. Finally, section 5 presents the results of our experiments.

2 Background

Intrusion detection Systems originally were developed to examine the activity of a single computer to process operating system audit records. This approach quickly expanded to systems looking at network traffic, gathering information produced in different hosts. Currently the emphasis is on developing Intrusion Detection Systems able to manage large volumes of data coming from multiple computers in a distributed system.

A Distributed Intrusion Detection System ($dIDS$) can be defined as a collection of multiple IDS spread over a large network, all of which communicate with each other to facilitate advanced network monitoring, incident analysis, and instant attack data [8]. One of the main advantages of dIDS is the ability to detect attack patterns across an entire network, with geographic locations dispersed over continents. The most important part of an IDS is its detection technique.

Traditionally, signature-based detection techniques have been used to individuate intrusions. Such methods gather features from the network data and detect intrusions by comparing the feature values to a set of attack signatures provided by a human expert. The main drawback of these approaches is that they are not able to discover new types of attacks, since their signature is not contained in the signature database. This weakness has stimulated research in trying data mining techniques to solve this task.

Data mining based intrusion detection approaches can be categorized in *misuse detection* [17, 4] and *anomaly detection* [16, 9]. In misuse detection, each instance in a set of data is labelled as normal or intrusion and a learning algorithm is trained over the labelled data. In anomaly detection a model is built over normal data and any deviation from the normal model in the new observed data is considered an intrusion. Misuse detection techniques reaches high degree of accuracy in discovering known intrusions and their variant, but they need to be retrained if new types of attacks are added to the input data set. On the other hand, anomaly detection techniques could generate high false alarm rate because normal unseen instances that differ from the normal behavior modeled are considered anomalies.

Genetic Programming for intrusion detection has received an increasing interest in the last few years. The first proposal is due to Crosbie and Spafford [7] which used agent technology to detect anomalous behavior in a system. Each autonomous agent was used to monitor a particular network parameter. However how to handle communication among the agents was not very clear. Orfila et al. [22] extended the previous approach to automatically create detection patterns/rules. The main contribution of this work is the adoption of domain knowledge in designing the function set and the usage of tree size limitations for obtaining simple and lightweight rules. Experiments have been conducted on publicly available raw tcp traffic from an enterprise network.

Another proposal for detecting novel attacks on network is presented by Lu and Traore [19]. The authors use Genetic programming to evolve rules whose aim is to discover attacks.

Linear Genetic Programming was proposed by Song et al. [28, 29] to address the intrusion detection classification problem over the KDD CUP 1999 data set. An individual is represented as a linear list of instructions and evaluation corresponds to the process of program execution associated with a simple register machine. In

section 5 a comparison of our results with those presented by Song et al. will be done. Linear GP is also adopted by Mukkamala et al. [21] to model intrusion detection systems and compare its performance with artificial neural networks and support vector machines. Abraham et al. [2] proposed Linear GP, Multi-Expression Programming, and Gene Expression Programming for detecting known types of attacks. The authors pointed out the suitability of Genetic Programming in developing accurate IDS.

An hybrid intrusion detection model that combines individual base classifiers generated by using decision trees and Support Vector Machines, is presented in [23]. The authors show that the performance of ensemble approach is better than those of the single classifiers.

Faraoun and Boukelif in [10] proposed an interesting method which consist of genetically coevolving a population of non-linear transformations on the input data to be classified, and map them to a new space with a reduced dimension, in order to get a maximum inter-classes discrimination. The application of the method to the KDD CUP dataset showed its capability of making accurate predictions on the unseen test data.

In [15], Hansen et al. presented a work oriented towards the problem of cyberterrorism. In this context a high-level of accuracy in identifying true positives must be reached and the technique must perform well on unseen instances. The authors experimented the accuracy reached using non homologous and homologous crossover. The latter obtained better accuracy in identifying positive and negative unseen instances in the test set of the KDD CUP 1999 dataset.

It worth to note that none of the above approaches cope with the problem of intrusion detection in distributed environments.

In the next section we propose the ensemble paradigm as a base framework to define a scalable, efficient and distributed algorithm for Intrusion Detection.

3 GP ensembles

Ensemble [14, 6] is a learning paradigm where multiple component learners are trained for the same task by a learning algorithm, and the predictions of the component learners are combined for dealing with new unseen instances. Let $S = \{(x_i, y_i)|i = 1, \ldots, N\}$ be a training set where x_i, called example or tuple or instance, is an attribute vector with m attributes and y_i is the class label associated with x_i. A predictor (classifier), given a new example, has the task to predict the class label for it.

Ensemble techniques build T predictors, each on a different training set, then combine them together to classify the test set. Boosting was introduced by Schapire

[26] and Freund [27] for boosting the performance of any "weak" learning algorithm, i.e. an algorithm that "generates classifiers which need only be a little bit better than random guessing" [27].

The boosting algorithm, called $AdaBoost$, adaptively changes the distribution of the training set depending on how difficult each example is to classify. Given the number T of trials (rounds) to execute, T weighted training sets $S_1, S_2, \ldots, S_T$ are sequentially generated and T classifiers $C^1, \ldots, C^T$ are built to compute a weak hypothesis h_t. Let w_i^t denote the weight of the example x_i at trial t. At the beginning $w_i^1 = 1/n$ for each x_i. At each round $t = 1, \ldots, T$, a weak learner C^t, whose error ϵ^t is bounded to a value strictly less than 1/2, is built and the weights of the next trial are obtained by multiplying the weight of the correctly classified examples by $\beta^t = \epsilon^t/(1 - \epsilon^t)$ and renormalizing the weights so that $\Sigma_i w_i^{t+1} = 1$. Thus "easy" examples get a lower weight, while "hard" examples, that tend to be misclassified, get higher weights. This induces AdaBoost to focus on examples that are hardest to classify. The boosted classifier gives the class label y that maximizes the sum of the weights of the weak hypotheses predicting that label, where the weight is defined as $log(1/\beta^t)$. The final classifier h_f is defined as follows:

$$h_f = arg\ max\ (\sum_t^T log(\frac{1}{\beta^t}) h_t(x, y))$$

Ensemble techniques have been shown to be more accurate than component learners constituting the ensemble [6, 25], thus such a paradigm has become a hot topic in recent years and has already been successfully applied in many application fields.

A key feature of the ensemble paradigm, often not much highlighted, concerns its ability to solve problems in a distributed and decentralized way.

We adopt such a paradigm to derive a network profile for modeling distributed intrusion detection systems and investigate the suitability of GP as component learner.

A GP ensemble offers several advantages over a monolithic GP that uses a single GP program to solve the intrusion detection task. First, it can deal with very large data sets. Second, it can make an overall system easier to understand, modify and implement in a distributed way. Finally, it is more robust than a monolithic GP, and can show graceful performance degradation in situations where only a subset of GPs in the ensemble are performing correctly.

One of the disadvantages of such an approach is the loss of interaction among the individual GPs during the learning phase. In fact, the individual GPs are often trained independently or sequentially. This does not take into account the interdependence that exists among the data and could bring to an ensemble overfitting

them and having weak characteristics of generalization. To this aim, we propose a method that emphasizes the cooperation among the individual GPs in the ensemble during the building of the solution.

Our approach is based on the use of cooperative GP-based learning programs that compute profile-based intrusion detection models over data stored locally at a site, and then integrate them by applying a majority voting algorithm.

The models are built using the local audit data generated on each node by, for example, operating systems, applications, or network devices so that each ensemble member is trained on a different training set.

The GP classifiers cooperate using the island model [3] to produce the ensemble members. Each node is an island and contains a GP-based learning component whose task is to build a decision tree classifier by collaborating with the other learning components located on the network. Each learning component evolves its population for a fixed number of iterations and computes its classifier operating on the local data. Each island may then import (remote) classifiers from the other islands and combine them with its own local classifier. Finally, once the classifiers are computed, they are collected to form the GP ensemble.

Diversity is an important problem that must be considered for forming successful ensembles. Genetic programming does not require any change in a training set to generate individuals of different behaviors. In [13] it is shown that GP enhanced with a boosting technique improves both the prediction accuracy and the running time with respect to the standard GP. We adopt the extension of GP with the AdaBoost.M2 algorithm as component learner to construct classification models for intrusion detection.

The next section describes the GEdIDS algorithm and its implementation using the dCAGE environment for the distributed execution of genetic programs by a hybrid island model.

4 The GEdIDS algorithm for dIDS

GEdIDS builds GP ensembles using a hybrid variation of the classic island model that leads not only to a faster algorithm, but also to superior numerical performance. The hybrid model combines the island model with the cellular model [3].

Figure 1 displays the software architecture of $GEdIDS$. The figure shows a network constituted by P nodes, each having its own data set S_j, a copy of the ensemble generated, and a $GEdIDS$ algorithm running on it. Each $GEdIDS$ is based on the cGP classification algorithm [13], better explained later. cGP runs for T rounds; for every round it generates a classifier, exchanges it with the other islands, and updates the weights of the tuples for the next round, according to the

boosting algorithm.

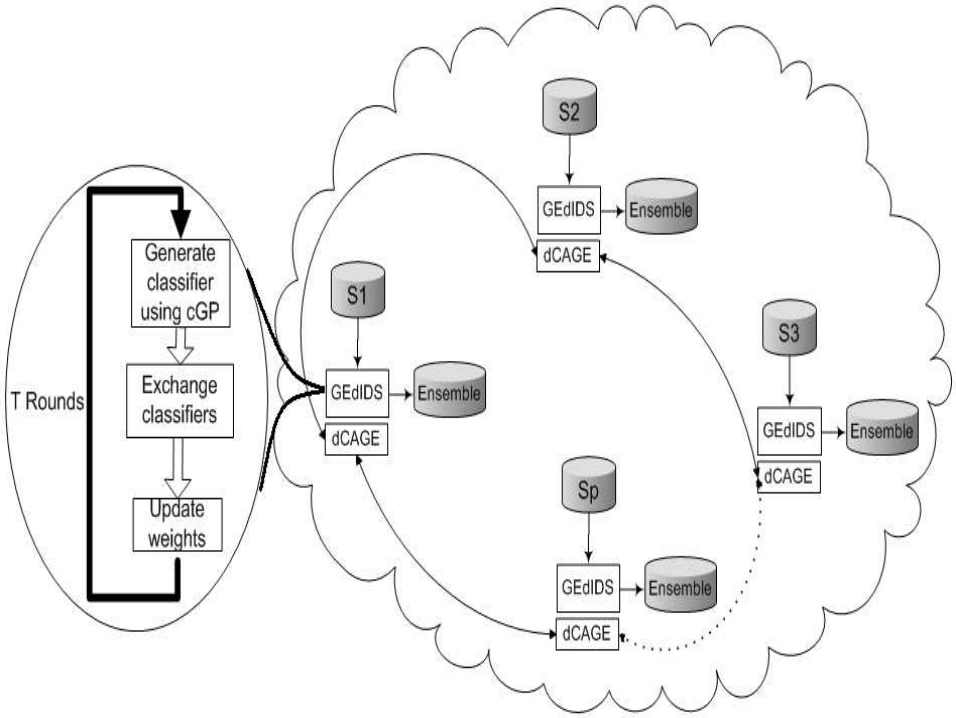

Figure 1: Software architecture of GEdIDS.

In order to create GP ensembles a distributed computing environment is required. We use dCAGE (distributed Cellular Genetic Programming System) a distributed environment to run genetic programs by an island model, which is an extension of [11].

dCAGE distributes the evolutionary processes (islands) that implement the detection models over the network nodes using a configuration file that contains the configuration of the distributed system. dCAGE implements the hybrid model as a collection of cooperative autonomous islands running on the various hosts within an heterogeneous network that works as a peer-to-peer system. The configuration of the processors is based on a ring topology.

The island model is based on subpopulations that are created by dividing the original population into disjunctive subsets of individuals, usually of the same size. Each subpopulation can be assigned to one processor and a standard (panmictic) GP algorithm is executed on it. Occasionally, migration process between subpopulations is carried out after a fixed number of generations. For example, the n best

individuals from one subpopulation are copied into the other subpopulations, thus allowing the exchange of genetic information between populations.

Our hybrid model modifies the island model by substituting the standard GP algorithm with a cellular GP (cGP) algorithm. In the cellular model each individual has a spatial location, a small neighborhood and interacts only within its neighborhood. The main difference in a cellular GP, with respect to a panmictic algorithm, is its decentralized selection mechanism and the genetic operators (crossover, mutation) adopted. Our model, as illustrated in figure 2, adopts a toroidal 2-D grid and the selection and mating operations are performed, cell by cell, only among the individual assigned to a cell and its neighbors. Selection, reproduction, and mating take place locally within the neighborhood. The individual to mate with the central individual k is chosen among the individuals in the Moore neighborhood and the best of the two offsprings replaces the current individual k. At each generation, the borders of the populations (the individuals on the left and on the right borders of the grid) are exchanged among neighboring peers, in an asynchronous fashion (i.e. if the borders do not arrive to a peer before starting the computation, the peer goes on with the old trees), so that all the islands can be thought as parts of a single population.

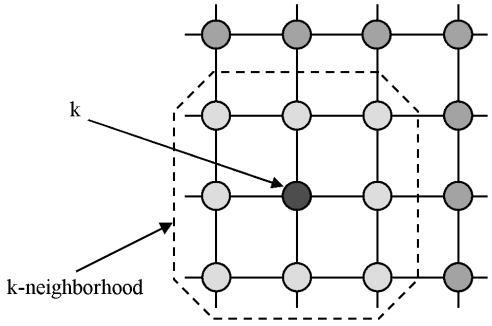

Figure 2: the cGP grid and the Moore neighborhood

GEdIDS uses the cGP algorithm to inductively generate a GP classifier as a decision tree for the task of data classification. Decision trees, in fact, can be interpreted as composition of functions where the function set is the set of attribute tests and the terminal set are the classes. The function set can be obtained by converting each attribute into an attribute-test function. For each attribute A, if $A_1, \ldots A_n$ are the possible values A can assume, the corresponding attribute-test function f_A has arity n and if the value of A is A_i then $f_A(A_1, \ldots A_n) = A_i$. When a tuple has to be evaluated, the function at the root of the tree tests the corresponding attribute

and then executes the argument that outcomes from the test. If the argument is a terminal, then the class name for that tuple is returned, otherwise the new function is executed. The fitness is the number of training examples classified in the correct class.

Let p_c, p_m be crossover and mutation probability
for each cell i in the population **do in parallel**
  evaluate the fitness of t_i
end parallel for
while not MaxNumberOfGeneration **do**
  for each cell i in the population **do in parallel**
    generate a random probability p
    if ($p < p_c$)
      select the cell j, in the neighborhood of i,
      such that t_j has the best fitness
      produce the offspring by crossing t_i and t_j
      evaluate the fitness of the offspring
      replace t_i with the best of the two offspring
      if its fitness is better than that of t_i
    else
    if ($p < p_m + p_c$) **then**
      mutate the individual
      evaluate the fitness of the new t_i
    else
      copy the current individual in the population
    end if
    end if
end parallel for
end while

Figure 3: The algorithm CGP

cGP is described in figure 3. At the beginning, for each cell, the fitness of each individual is evaluated. Then, at each generation, every tree undergoes one of the genetic operators (reproduction, crossover, mutation) depending on the probability test. If crossover is applied, the mate of the current individual is selected as the neighbor having the best fitness, and the offspring is generated. The current tree is then replaced by the best of the two offsprings if the fitness of the latter is better

than that of the former. The evaluation of the fitness of each classifier is calculated on the entire training data. After the execution of the number of generations defined by the user, the individual with the best fitness represents the classifier.

Given a network constituted by P nodes,
each having a data set S_j
For j = 1, 2, . . ., P (for each island in parallel)
  Initialize the weights associated with each tuple
  Initialize the population Q_j with random individuals
end parallel for
For t = 1,2,3, . . ., T (boosting rounds)
  For j = 1, 2, . . ., P (for each island in parallel)
    Train cGP on S_j using a weighted fitness
    according to the weight distribution
    Compute a weak hypothesis
    Exchange the hypotheses among the P islands
    Update the weights
  end parallel for
end for t

Output the hypothesis

Figure 4: The GEdIDS algorithm using AdaBoost.M2

The pseudo-code of the GEdIDS algorithm is shown in figure 4. Each island is furnished with a cGP algorithm enhanced with the boosting technique AdaBoost.M2, a population initialized with random individuals, and operates on the local audit data weighted according to a uniform distribution. The selection rule, the replacement rule and the asynchronous migration strategy are specified in the cGP algorithm. Each island generates the GP classifier by running for a certain number of iterations, necessary to compute the number of boosting rounds. During the boosting rounds, each classifier maintains the local vector of the weights that directly reflect the prediction accuracy on that site. At each boosting round the hypotheses generated by each classifier are exchanged among all the processors in order to produce the ensemble of predictors. In this way each island maintains the entire ensemble and it can use it to recalculate the new vector of weights. After the execution of the fixed number of boosting rounds, the classifiers are used to evaluate the accuracy of the classification algorithm for intrusion detection on the

Table 1: Class distribution for training and test data for KDDCUP 99 dataset

	Normal	**Probe**	**DoS**	**U2R**	**R2L**	**Total**
Train	97277	4107	391458	52	1126	494020
Test	60593	4166	229853	228	16189	311029

entire test set.

In the next section, we experimentally prove the capability of our approach on the KDD CUP data set.

5 System evaluation and results

5.1 Data sets description

We performed experiments over the KDD Cup 1999 Data set [1]. Though this data set has been judged not representative of a realistic IDS scenario [20], it is a reference data set, extensively used to compare results of different intrusion detection techniques. The data set comes from the 1998 DARPA Intrusion Detection Evaluation Data [18] and contains a training data consisting of 7 weeks of network-based attacks inserted in the normal data, and 2 weeks of network-based attacks and normal data for a total of 4,999,000 of connection records described by 41 characteristics. The main categories of attacks are four: DoS (Denial of Service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to a local superuser privileges by a local unprivileged user), PROBING (surveillance and probing). However a smaller data set consisting of the 10% the overall data set is generally used to evaluate algorithm performance. In this case the training set consists of 494,020 records among which 97,277 are normal connection records, while the test set contains 311,029 records among which 60,593 are normal connection records. Table 1 shows the distribution of each attack type in the training and the test set. Note that the test set is not from the same probability distribution as the training data, in fact it includes specific attack types not in the training data. This makes the task more realistic.

5.2 Performance measures

To evaluate our system, besides the classical accuracy measure, the two standard metrics of *detection rate* and *false positive rate* developed for network intrusions, have been used. Table 2 shows these standard metrics. Detection rate is computed

as the ratio between the number of correctly detected attacks and the total number of attacks, that is

$$DR = \frac{\#TruePositive}{\#FalseNegative + \#TruePositive}$$

False positive (also said false alarm) rate is computed as the ratio between the number of normal connections that are incorrectly classifies as attacks and the total number of normal connections, that is

$$FP = \frac{\#FalseAlarm}{\#TrueNegative + \#FalseAlarm}$$

These metrics are important because they measure the percentage of intrusions the system is able to detect and how many misclassifications it makes. To visualize the trade-off between the false positive and the detection rates, the ROC (Receiving Operating Characteristic) curves [24] are also depicted. Furthermore, to compare classifiers it is common to compute the area under the ROC curve, denoted as AUC [5]. The higher is the area, better is the average performance of the classifier.

Table 2: Standard metrics to evaluate intrusions.

		Predicted label	
		Normal	Intrusions
Actual Class label	Normal	True Negative	False Alarm
	Intrusions	False Negative	True Positive

5.3 Experimental setup

The experiments were performed by assuming a network composed by 10 dual-processor 1,133 Ghz Pentium III nodes having 2 Gbytes of memory. The training set of 499,467 tuples was equally partitioned among the 10 nodes using a random sampling, thus containing 1/10 of instances for each class. On each node we run $AdaBoost.M2$ as base GP classifier with a population of 100 elements for 10 rounds, each round consisting of 100 generations. The GP parameters used are the same for each node and they are shown in table 3. All the experiments have been obtained by running the algorithm 10 times and averaging the results. Each ensemble has been trained on the train set and then evaluated on the test set.

Table 3: Main parameters used in the experiments

Name	Value
max_depth_for_new_trees	6
max_depth_after_crossover	17
max_mutant_depth	2
grow_method	RAMPED
selection_method	GROW
crossover_func_pt_fraction	0.7
crossover_any_pt_fraction	0.1
fitness_prop_repro_fraction	0.1
parsimony_factor	0

Table 4: Detection Rate and False Positive Rate for GEdIDS (avg, best Det. Rate and Best False Pos. Rate) after 2, 5 and 10 rounds.

		Detection Rate	FP Rate	Accuracy
2 rounds	Avg GEdIDS	0.907483	0.023154	0.901569
	Best Det. Rate	0.910267	0.045658	0.909548
	Best FP Rate	0.903428	0.012931	0.911478
5 rounds	Avg GEdIDS	0.907038	0.008324	0.917586
	Best Det. Rate	0.911434	0.013632	0.919644
	Best FP Rate	0.908815	0.004621	0.920468
10 rounds	Avg GEdIDS	0.905812	0.005648	0.918624
	Best Det. Rate	0.911522	0.005941	0.923592
	Best FP Rate	0.910165	0.004340	0.923782

5.4 Results and comparison with other approaches

The results of our experiments are summarized in table 4 where the detection rate, false positive rate, and accuracy are reported after 2, 5, and 10 rounds of the boosting algorithm. For each of them the table shows the average values obtained by running the algorithm 10 times, and the best values obtained with respect to the false positive and detection rates. The table shows that increasing the number of boosting rounds has a positive effect on both the false positive rate and accuracy, while the average detection rate slightly diminishes, though the best value obtained among the 10 executions improves. Table 5 reports the classification accuracy detailed for each class, after 2, 5, and 10 rounds, respectively, attained on the test set

Table 5: Classification accuracy of GEdIDS for each class, after 2, 5 and 10 rounds.

	2 rounds	**5 rounds**	**10 rounds**
normal	97.6846%	99.1676%	99.44%
probe	15.6505%	68.7326%	71.97%
DoS	96.1086%	96.5456%	96.53%
u2r	2.0614%	2.9825%	5.18%
r2l	2.5264%	3.2380%	3.60%

Table 6: Comparison with kdd-99 cup winners and other approaches

Algorithm	**Detection Rate**	**FP Rate**	**ROC Area**
Winning Entry	0.919445	0.005462	0,956991
Second Place	0.915252	0.005760	0,954746
Best Linear GP - FP Rate	0.894096	0.006818	0,943639
Avg GEdIDS	0.905812	0.005648	0.950082
Best GEdIDS - FP Rate	0.910165	0.004340	0.952912

by averaging results coming from 10 different executions of $GEdIDS$. The table points out that the prediction is worse on the two classes U2R and R2L. For this two classes, however, there is a discrepancy between the number of instances used to train each classifiers on every node and the number instances to classify in the test set (only 52 and 228 tuples respectively for training, while 1126 and 16189 for testing).

Table 6 compares our approach with the first and second winner of the KDD-99 CUP competition and the linear genetic programming approach proposed by Song et al. [28]. The table shows the values of the standard metrics described above. In particular we show the detection rate, the false positive rate, and the ROC area of these three approaches and those obtained by $GEdIDS$. For the latter we show both the average values of the 10 executions and the best value with respect to the false positive rate (for the sake of comparison with a GP-based approach). It is worth to note that, as regards Linear GP, the authors in their paper [28] reported only the best values found. From the table we can observe that the average and best $GEdIDS$ detection rates are 0.905812 and 0.910165, respectively, while those of the first two winners are 0.919445 and 0.915252. As regard the false positive rate the average value of $GEdIDS$ 0.005648 is lower than the second entry, while the

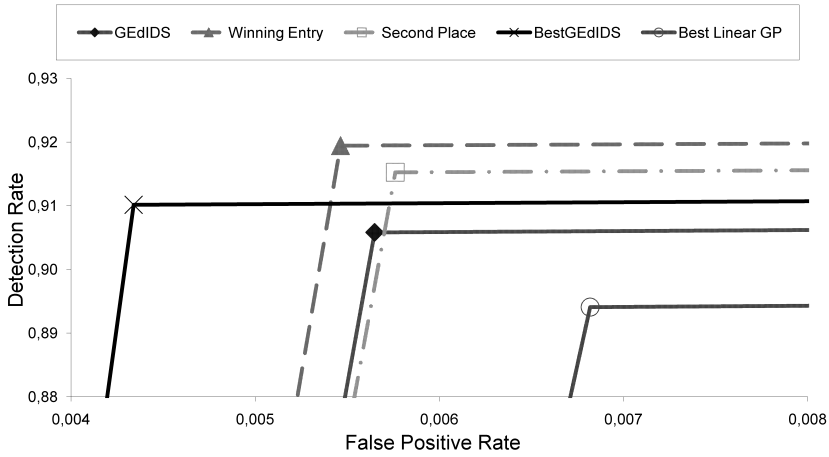

Figure 5: ROC curves.

best value obtained 0.004340 is lower than both the first and second entries. Thus the goodness of the solutions found by $GEdIDS$ is not very different from the winning entry and from the Linear GP approach. These experiments emphasizes the capability of genetic programming to deal with this kind of problem. Figure 5 shows an enlargement of the ROC curves of the methods listed in table 6 and better highlights the results of our approach.

Finally we compared $GEdIDS$ with the other well known classifications methods $C4.5$, and its boosting and bagging versions.

We used the implementations contained in the $WEKA$ [30] open source software available at http://www.cs.waikato.ac.nz/ml/weka/. For this experiment we report the classification accuracy of all the methods, detailed for each class, and the global detection rate and false positive rate. In particular, for $GEdIDS$, we presents the results obtained when each node works with 10%, 50%, and 100% of the data set. Table 7 points out that when the ensemble is built by using only the 10% of the data set on each node, $GEdIDS$ has a generalization capability reduced with respect to the other methods. This behavior is clearly a consequence of the fact that the size of the training set is too small for some kind of attacks. For example, the class U2R consists of 52 training tuples, thus each node receives only 5 tuples. This implies that the predictive accuracy of the ensemble can not be good on the 228 tuples of the test set. However, as soon as the size of the training set augments, we can note that $GEdIDS$ outperforms the other approaches. For example, the predictive accuracy of the two most difficult classes, U2R and R2L, increases from 5.18 to 9.47 and 10.04 as regards U2R, and from 3.60 to 7.36

Table 7: Comparison between $GEdIDS$ and C4.5, boosting and bagging C4.5.

	GEdIDS			**C4.5**	**BoostC4.5**	**BagC4.5**
	10%	**50%**	**100%**			
normal	99.44%	99.48%	99.51%	99.49%	99.50%	99.48%
probe	71.97%	**81.51%**	81.83%	74.70%	79.09%	79.67%
DoS	96.53%	97.03%	97.10%	97.27%	97.27%	97.27%
u2r	5.18%	**9.47%**	10.04%	2.63%	8.33%	5.32%
r2l	3.60%	**7.36%**	7.44%	5.84%	6.22%	4.64%
Det. Rate	0.90581	0.91295	0.91357	0.91096	0.91117	0.91112
FP rate	0.005648	0.005225	0.004915	0.005067	0.005017	0.005166

and 7.44 as regards R2L, when 10%, 50% and 100% of the training set is used. The table shows also the good outcomes of detection rate and false positive rate of $GEdIDS$.

To statistically validate the results, we performed a two-tailed paired t-test at 95% confidence interval. The values in bold of the column 50% of $GEdIDS$ highlight when our algorithm obtains an error lower than the other approaches, meaningful with respect to the statistical test.

6 Conclusions

A distributed intrusion detection approach based on Genetic Programming and extended with the ensemble paradigm, to classify malicious or unauthorized network activity has been presented. GP ensembles are built using a distributed cooperative approach based on a hybrid model that combines the cellular and the island models. The combination of these two models provides an effective implementation of distributed GP, and the generation of classifiers with good classification accuracy. A main advantage of the distributed architecture is that it enables for flexibility, extensibility, and efficiency since each node of the network works with its local data, and communicate with the other nodes, to obtain the results, only the local model computed, but not the data. This architecture is thus particularly apt to deal with enormous amount of data generated at different locations. Experimental results showed the suitability of GP as component learner of the ensemble for this kind of problems. An extension that deserves to be investigated regards the possibility of considering not batch data sets but data streams that change online on each node of the network.

References

[1] The third international knowledge discovery and data mining tools competition dataset kdd99-cup. In *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*, 1999.

[2] Ajith Abraham, Crina Grosan, and Carlos Martin-Vide. Evolutionary design of intrusion detection programs. *International Journal of Network Security*, 4(3):328–339, March 2007.

[3] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transaction on Evolutionary Computation*, 6(5):443–462, October 2002.

[4] D. Barbara, N. Wu, and S. Jajodia. Detecting novel network intrusions using bayes estimator. In *First SIAM Conference on Data Mining*, 2001.

[5] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[6] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[7] M. Crosbie and G. Spafford. Applying genetic programming techniques to intrusion detection. In *Proceedings of the AAAI Fall Symposium Series*. AAAI Press, November 1995.

[8] N. Einwechter. An introduction to distributed intrusion detection systems. In *http://www.securityfocus.com/infocus/1532*, 2002.

[9] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection : Detecting intrusions in unlabeled data. In *Applications of Data Mining in Computer Security, Kluwer*, 2002.

[10] Kamel Faraoun and Aoued Boukelif. Genetic programming approach for multi-category pattern classification applied to network intrusions detection. *International Journal of Computational Intelligence and Applications*, 6(1):77–99, 2006.

[11] G. Folino, C. Pizzuti, and G. Spezzano. A scalable cellular implementation of parallel genetic programming. *IEEE Transaction on Evolutionary Computation*, 7(1):37–53, February 2003.

[12] G. Folino, C. Pizzuti, and G. Spezzano. GP ensemble for distributed intrusion detection systems. In *ICAPR 2005, Proc. of the 3rd International Conference on Advanced in Pattern Recognition*, pages 54–62, 2005.

[13] G. Folino, C. Pizzuti, and G. Spezzano. Ensembles for large scale data classification. *IEEE Transaction on Evolutionary Computation*, 10(5):604–616, October 2006.

[14] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th Int. Conference on Machine Learning*, pages 148–156, 1996.

[15] James V. Hansen, Paul Benjamin Lowry, Rayman D. Meservy, and Daniel M. McDonald. Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection. *Decis. Support Syst.*, 43(4):1362–1374, 2007.

[16] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proc. SIAM Int. Conf. on Data Mining (SIAM-03)*, 2003.

[17] W. Lee and S.J. Stolfo. Data mining approaches for intrusion detection. In *Proc. of the 1998 USENIX Security Symposium*, pages 66–72, 1998.

[18] Richard P. Lippmann, David J. Fried, David J. Fried, Isaac Graf, Isaac Graf, Joshua W. Haines, Joshua W. Haines, Kristopher R. Kendall, Kristopher R. Kendall, David Mcclung, David Mcclung, Dan Weber, Dan Weber, Seth E. Webster, Seth E. Webster, Dan Wyschogrod, Dan Wyschogrod, Robert K. Cunningham, Robert K. Cunningham, Marc A. Zissman, and Marc A. Zissman. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, pages 12–26, 2000.

[19] Wei Lu and Issa Traore. Detecting new forms of network intrusion using genetic programming. In *Proc. of the Congress on Evolutionary Computation CEC'2003*, pages 2165–2173. IEEE Press, 2003.

[20] J. . McHugh. Testing intrusion detection systems: a critique of the 1988 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[21] Srinivas Mukkamala, Andrew H. Sung, and Ajith Abraham. Modeling intrusion detection systems using linear genetic programming approach. In *17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004*, pages 633–642, Ottawa, Canada, 2004.

[22] Agustin Orfila, Juan M. Estevez-Tapiador, and Arturo Ribagorda. Evolving high-speed, easy-to-understand network intrusion detection rules with genetic programming. In *EvoWorkshops '09: Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing*, pages 93–98, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] Sandhya Peddabachigari, Ajith Abraham, Crina Grosan, and Jonson Thomas. Modeling intrusion detection system using hybrid intelligent systems. *International Journal of Network and Computer Applications*, (30):114–132, 2007.

[24] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proc. Int. Conf. on Machine Learning (ICML'98)*, 1998.

[25] J. Ross Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence AAAI96*, pages 725–730. Mit Press, 1996.

[26] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[27] R. E. Schapire. Boosting a weak learning by maiority. *Information and Computation*, 121(2):256–285, 1996.

[28] D. Song, M.I. Heywood, and A. Nur Zincir-Heywood. A linear genetic programming approach to intrusion detection. In *Proceedings Of the Genetic and Evolutionary Computation Conference GECCO 2003*, pages 2325–2336. LNCS 2724, Springer, 2003.

[29] D. Song, M.I. Heywood, and A. Nur Zincir-Heywood. Training genetic programming on half a millio patterns: an example from anomaly detection. *IEEE Transaction on Evolutionary Computation*, 9(3):225–239, June 2005.

[30] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd Edition, San Francisco, 2005.