Fast Intrusion Detection System based on Flexible Neural Tree

Tomáš Novosád, Jan Platoš, Václav Snášel
Department of Computer Science
VŠB-Technical University of Ostrava
Ostrava, Czech Republic
{*tomas.novosad,jan.platos,vaclav.snasel*}*@vsb.cz*

Ajith Abraham
Machine Intelligence Research Labs (MIR Labs), USA
ajith.abraham@ieee.org

***Abstract*—Computer security is very important in these days. Computers are used probably in any industry and their protection against attacks is very important task. The protection usually consist in several levels. The first level is preventions. Intrusion detection system (IDS) may be used as next level. IDS is useful in detection of intrusions, but also in monitoring of security issues and the traffic. This paper present IDS based on Flexible Neural Trees. Flexible neural tree is hierarchical neural network, which is automatically created using evolutionary algorithms to solving of defined problem. This is very important, because it is not necessary to set the structure and the weights of neural networks prior the problem is solved. The accuracy of proposed technique is always above 98% and the speed of decision making process enable its using in real-time applications.**

***Keywords*-flexible neural tree, genetic algorithm, intrusion detection, real-time classification**

I. Introduction

An intrusion is defined to be a violation of the security policy of the system; intrusion detection thus refers to the mechanisms that are developed to detect violations of system security policy. Intrusion detection is based on the assumption that intrusive activities are noticeably different from normal system activities and thus detectable. Intrusion detection is not introduced to replace prevention-based techniques such as authentication and access control; instead, it is intended to complement existing security measures and detect actions that bypass the security monitoring and control component of the system. Intrusion detection is classified into two types: misuse intrusion detection and anomaly intrusion detection. Misuse intrusion detection uses well-defined patterns of the attack that exploit weaknesses in the system and application software to identify the intrusions. Anomaly intrusion detection identifies deviations from the normal usage behavior patterns to identify the intrusion.

Various intelligent paradigms namely Neural Networks [10], Support Vector Machine [14], Neuro-Fuzzy systems [17], Linear genetic programming [1], Decision Trees [5] and Adaptive Regression Splilnes [15] have been used for intrusion detection. Various data mining techniques have been applied to intrusion detection because it has the advantage of discovering useful knowledge that describes a user's or program's behavior from large audit data sets [18], [4].

This paper proposes a Flexible Neural Tree (FNT) [7] for selecting the input variables and detection of network intrusions. Based on the predefined instruction/operator sets, a Flexible Neural Tree Model can be created and evolved. FNT allows input variables selection, over-layer connections and different activation functions for different nodes. In our previous work, the probabilistic incremental program evolution (PIPE) and ant programming (AP) algorithm have been employed to find a near-optimal neural tree [8], [9].

In this work, the hierarchical structure is evolved using tree-structure based genetic algorithm. The fine tuning of the parameters and weights encoded in the structure is accomplished using evolutionary algorithms as well. The proposed method interleaves both optimizations. The procedure starts with randomly generated structures and corresponding parameters and weights. It first tries to improve the structure and then as soon as an improved structure is found, it fine tunes its parameters and weights. It then goes back to improving the structure again and, fine tunes the structure and rules parameters. This process continues until a satisfactory solution is found or a time limit is reached. The goal of this work is the usage of flexible neural tree model for selecting the important features of IDS and for constructing classifier for intrusion detection using flexible neural tree which accomplishes real-time classification.

Rest of the paper is organized as follows. In Section II, we present some theoretical background of flexible neural tree model. In Section V experiment details and results are provided, followed by conclusions in the last section.

II. Flexible Neural Tree

A general and enhanced flexible neural tree (FNT) model is proposed for problem solving. Based on the predefined instruction/operator sets, a flexible neural tree model can be created and evolved. In this approach, over-layer connections, different activation functions for different nodes and input variables selection are allowed. The hierarchical structure could be evolved by using tree-structure based evolutionary algorithms with specific instructions. The fine tuning of the parameters encoded in the structure could be accomplished by using parameter optimization algorithms. The proposed method interleaves

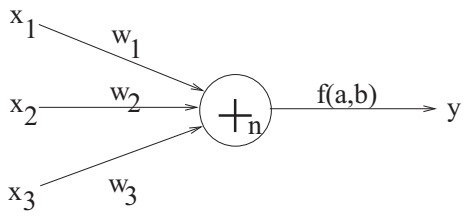

Figure 1. A flexible neuron operator (instructor)

both optimizations. Starting with random structures and corresponding parameters, it first tries to improve the structure and then as soon as an improved structure is found, it fine tunes its parameters. It then goes back to improving the structure again and, provided it finds a better structure, it again fine tunes the rules' parameters. This loop continues until a satisfactory solution is found or a time limit is reached.

A. Encoding and Evaluation

A tree-structural based encoding method with specific instruction set is selected for representing a FNT model in this research. The reason for choosing the representation is that the tree can be created and evolved using the existing or modified tree-structure-based approaches, i.e., Genetic Programming (GP), Probabilistic Incremental Program Evolution (PIPE), Ant Programming (AP).

B. Flexible Neuron Instructor

The used function set F and terminal instruction set T for generating a FNT model are described as follows:

$$S = F \cup T = \{+_2, +_3, \ldots, +_N\} \cup \{x_1, x_2, \ldots, x_n\} \quad (1)$$

where $+_i$ $(i = 2, 3, \ldots, N)$ denote non-leaf nodes' instructions and taking i arguments. Input variables $x_1, x_2, \ldots, x_n$ are leaf nodes' instructions and taking no argument each. The output of a non-leaf node is calculated as a flexible neuron model (see figure 2). From this point of view, the instruction $+_i$ is also called a flexible neuron operator (instructor) with i inputs. In the **creation process** of neural tree, if a nonterminal instruction, i.e., $+_i$ is selected, i real values are randomly generated and used for representing the connection strength between the node $+_i$ and its children. In addition, two adjustable parameters a_i and b_i are randomly created as flexible activation function parameters. Activation function can vary according to given task. In this work we use following classical Gaussian activation function:

$$f(a_i, b_i, x) = e^{-(\frac{x-a_i}{b_i})^2} \quad (2)$$

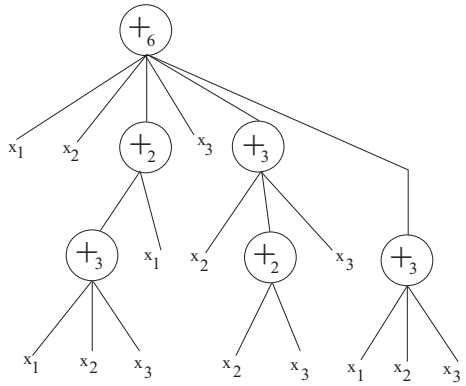

Figure 2. A typical representation of neural tree with function instruction set $F = \{+_2, +_3, +_4, +_5, +_6\}$, and terminal instruction set $T = \{x_1, x_2, x_3\}$

The output of a flexible neuron $+_n$ can be calculated as follows. The total excitation of the $+_n$ is

$$net_n = \sum_{j=1}^{n} w_j \times x_j \quad (3)$$

where x_j $(j = 1, 2, \ldots, n)$ are the inputs to node $+_n$. The output of the node $+_n$ is then calculated by

$$out_n = f(a_n, b_n, net_n) = e^{-(\frac{net_n - a_n}{b_n})^2} \quad (4)$$

A typical evolved flexible neural tree model is shown as Figure 2. The overall output of flexible neural tree can be computed from left to right by depth-first method, recursively.

C. Fitness function

A fitness function maps FNT to scalar, real-valued fitness values that reflect the FNT' performances on a given task. Firstly the fitness functions should be seen as error measures, i.e., MSE or $RMSE$. A secondary non-user-defined objective for which algorithm always optimizes FNTs is FNT size as measured by number of nodes. Among FNTs with equal fitness smaller ones are always preferred. Commonly used fitness function used for the PIPE and SA is given by mean square error (MSE):

$$Fit(i) = \frac{1}{P} \sum_{j=1}^{P} (y_1^j - y_2^j)^2 \quad (5)$$

or Root Mean Square Error ($RMSE$):

$$Fit(i) = \sqrt{\frac{1}{P} \sum_{j=1}^{P} (y_1^j - y_2^j)^2} \quad (6)$$

where P is the total number of samples, y_1^j and y_2^j are the actual time-series and the FNT model output of j-th sample. $Fit(i)$ denotes the fitness value of i-th individual.

D. Tree Structure and Parameter Learning

Finding an optimal or near-optimal neural tree could be accomplished by using tree-structure based evolutionary algorithms, i.e., genetic programming (GP), probabilistic incremental program evolution (PIPE), gene expression programming (GEP), multi expression programming (MEP), estimation of distribution programming (EDP) and the parameters optimization algorithms, i.e., genetic algorithms (GA), evolution strategy (ES), evolutionary programming (EP), particle swarm optimization (PSO), estimation of distribution algorithm (EDA), and so on. The general learning procedure for constructing the FNT model can be described in high level as follows [7]:

1) Set the initial values of parameters used in the GA algorithms. Set the elitist program as NULL and its fitness value as a biggest positive real number of the computer at hand. Create a random initial population (flexible neural trees and their corresponding parameters)
2) Structure optimization by genetic algorithm, in which the fitness function is calculated by mean square error (MSE) or root mean square error(RMSE)
3) If a better structure is found and no better structure is found for certain number of generations (10 in this study), then go to step (4), otherwise go to step (2)
4) Parameter optimization by genetic algorithms. In this stage, the tree structure or architecture of flexible neural tree model is fixed, and it is the best tree taken from the sorted population of trees. All of the parameters used in the best tree formulated a parameter vector to be optimized by local search
5) If the maximum number of local search is reached, or no better parameter vector is found for a significantly long time then go to step (6); otherwise go to step (4);
6) If satisfactory solution is found, then the algorithm is stopped; otherwise go to step (2).

We have modified the third step of the algorithm to achieve time savings in evolution of best flexible neural tree. Experiments have shown there is no need to optimize the connection weights and parameters immediately after the better structure is found, because very often better structure is discovered after few generations. The weights and parameters tunning is also very time consuming task, so we rather wait certain number of generations before we start weights and parameters optimization. By this enhancement we achieved very good time savings toward the original algorithm.

III. Genetic Algorithms and Flexible Neural Trees

In this work, we use genetic algorithms [2], [3] for structure optimization as well as for activation function parameters and tree nodes weights optimization. The selection, crossover and mutation operators used are same as those of standard genetic programming (GP). A genetic algorithm starts with selection of two parents from current population. The product of crossover operator can be one or more offspring - two in this study. The mutation of offspring is performed at the last step of genetic algorithm. After these three steps we have new offspring which is placed into a newly arise population. The process is repeated until desired new population is built. As soon as the new population is built, the new population is evaluated and sorted according to the fitness function.

We have also modified the population sort procedure to achieve smaller trees - feature extraction. If the fitness of two trees is equal or almost the same (differs about some user defined threshold) we prefer the smaller tree, thus the smaller trees with such a feature are placed better, what means it have better chance to be selected for crossover and mutation.

In this study we also use another genetic operator called Elitism. The elitism means that certain number of best individuals from the current generation are copied into a new generation, without performing crossover and mutation operations. This method can very rapidly increase performance of genetic algorithms, because it prevents losing the best found solution.

A. Selection

Suppose we have population P of n individuals sorted according to individuals fitnesses. We need to select two parents for crossover operator. The selection in our work is done by weighted roulette algorithm.

B. Crossover

In GP the tree structure crossover operation is implemented by taking randomly selected subtrees in the individuals and exchanging them. Crossover of node weights and activation function parameters are done same as for artificial neural networks [13], [12], [21], [20].

C. Mutation

A number of neural tree mutation operators are developed as follows:

1) Changing one terminal node: randomly select one terminal node in the neural tree and replace it with another terminal node.
2) Changing one function node: randomly select one function node and replace it with a newly generated subtree.
3) Growing: select a random function node in hidden layer of the neural tree and add newly generated subtree as a new child.
4) Pruning: randomly select a node in the neural tree and delete it in the case the parent node has more than two child nodes.

A mutation of tree weights and activation function parameters is the same as for artificial neural networks [13], [12], [21], [20].

IV. Experimental data

The data for the experiments was prepared by the 1998 DARPA intrusion detection evaluation program by MIT Lincoln Labs [11]. The original data contains 744 MB data with 4,940,000 records. The data set has 41 attributes for each connection record plus one class label. Some features are derived features, which are useful in distinguishing normal connection from attacks. These features are either nominal or numeric. Description of these data may be found in [18], [16], [11], [19]. For our experiments, the 10% sample of these data is used [19].

The data was preprocessed; all nominal values were converted to the numeric representation and all features were normalized. The data was divided into 4 classes, according the attack classes defined in [19]. Each class contains all records signed as normal activity and all records which belong to specified attack class. For each class, a division into two collections - training data and testing data was performed. The training data contains 40% of randomly selected data and the testing data contains the rest of them. Finally, another one class was defined - this class contains all records from all 4 attack classes and the normal activity records. This class represented the main functionality of the IDS - distinguish attacks from the normal activity.

The data collections are called according the name of the attack class, which it contains. The five used collection are called *DoS* (Denial of Services), *U2R* (User to Root), *R2L* (Remote to User), *Probe* (Probing) and the last one which contains all records is called *Attack*, because its purpose is to test algorithm to distinguish any attack.

The statistics of the data collection is shown in Table I. As may be seen, the number of records differs in each collection. The class *DoS* contains the largest amount of records and the class *U2R* contains the lowest number of records. The amount of records, which belongs to the normal traffic is the same in each collection - approximately 90 000. This will compare the efficiency of the proposed algorithm in detecting events, which are rare (*U2R*) and events which are very frequent in the data (*DoS*).

Table I
Statistics of the data collections

Collection	training records	Testing records	Total records
DoS	195 494	293 242	488 736
U2R	38 931	58 399	97 330
R2L	39 361	59 043	98 404
Probe	40 553	60 832	101 385
Attack	197 608	296 413	494 021

V. Experiments and Results

In this section we present the results of experiments with flexible neural tree based IDS for large data. One of key benefit of the flexible neural tree is dimension reduction feature. Secondly, it has shown the FNT is a very fast classifier which accomplishes real-time decision making. All of the experiments have more than 98% accuracy in all of the testing sets. Time consumption for classification of all records did not exceed 300 milliseconds for bigger collections (DoS and Attack) and 100 milliseconds for smaller collections (U2R, R2L and Probe).

The mechanisms of input selection in the FNT constructing procedure are as follows. (1) Initially all the input variables are selected to formulate the random FNT model (initial population of randomly generated trees) with same probabilities. (2) The variables that have more contribution to the objective function will be enhanced and have a high opportunity to survive at the next generations by a evolutionary procedure. (3) The evolutionary operators, that is crossover and mutation, provide an input selection method by which the FNT should select appropriate variables automatically. The size of the initial population of randomly generated trees were set to 50 individuals and size of the initial population of connection weights and activation function parameters was set to 100 individuals. At the beginning of the algorithm each of the randomly generated tree covered all of the 41 input variables. In this stage of the algorithm the training set is used.

In this work we use flexible neural tree as a classifier as well, i.e. the part of IDS responsible for decision making. Once the flexible neural tree structure, weights and activation function parameters are evolved, we use this evolved tree for classification of records appearing in our collection. The classification made by flexible neural tree is the overall output of the evolved tree. The input to the classifier (FNT) are variables selected during the training phase, i.e. evolved tree terminal nodes (leaf nodes).

Summary of the results is shown in table II and table III respectively. In table II are the results for teaching collections and table III denotes the results for testing collections. Description of the tables is follows: column *Precision* means percent of correctly classified records, column *FP* holds for False Positive Error, column *FN* represents False Negative Error, column *Time* is the total time in milliseconds of classification of all records and column *Records* is the number of records being classified.

As was mentioned above, one of the key feature of flexible neural trees is the dimension reduction. Our flexible neural tree algorithm based on genetic algorithms and with the feature of preferring smaller trees, reduced the number of important variables of class U2R and R2L to just 2 of original 41 input variables. This significantly improves the classification time. Table IV shows the sizes of best evolved

trees and extracted important variables as well. Description of variables can be found in [6].

In figures 3, 4, 5, 6 are presented best evolved trees by our FNT algorithm. These trees are used as a classifiers for samples of training and testing collections.

Table II
RESULTS FOR TRAINING COLLECTIONS

Collection	Precision[%]	FP[%]	FN[%]	Time[ms]	Records
DoS	98.88	0.42	0.68	140	195 494
U2R	99.95	0.0	0.05	31	38 931
R2L	98.86	0.0	1.14	31	39 361
Probe	99.25	0.03	0.71	47	40 553
Attack	98.54	0.25	1.21	188	197 608

Table III
RESULTS FOR TEST COLLECTIONS

Collection	Precision[%]	FP[%]	FN[%]	Time[ms]	Records
DoS	98.84	0.42	0.71	250	293 242
U2R	99.94	0.001	0.05	47	58 399
R2L	98.85	0.001	1.14	47	59 043
Probe	99.31	0.03	0.65	78	60 832
Attack	98.57	0.27	1.15	297	296 413

Table IV
FNT SIZE AND EXTRACTED VARIABLES

Class	FNT Size	Important variables
DoS	7	x_2, x_9, x_{12}, x_{25}, x_{34}, x_{41}
U2R	3	x_9, x_{41}
R2L	3	x_9, x_{25}
Probe	20	x_2, x_4, x_9, x_{15}, x_{18}, x_{25}, x_{27}, x_{30}, x_{31}, x_{34}, x_{41}
Attack	11	x_2, x_4, x_9, x_{11}, x_{12}, x_{18}, x_{25}, x_{27}, x_{34}, x_{41}

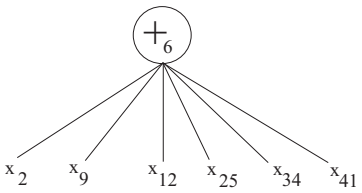

Figure 3. Best evolved tree for Denial of Services collection.

VI. CONCLUSIONS

In this paper, we presented a Flexible Neural Tree approach to detect different types of network intrusions. We have used genetic algorithms for tree structure optimization as well as for connection weights and activation function parameters tuning. All the experiments were done on new large data collection and the results show very good classification accuracy which was not worse than 98% of correctly classified records in all data collections. We have proved the flexible neural tree is also very suitable for real-time decision making and great for dimension reduction.

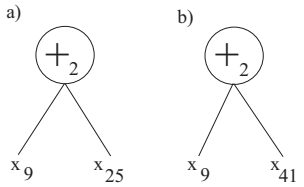

Figure 4. Best evolved tree for a) Remote to User collection and b) User to Root collection.

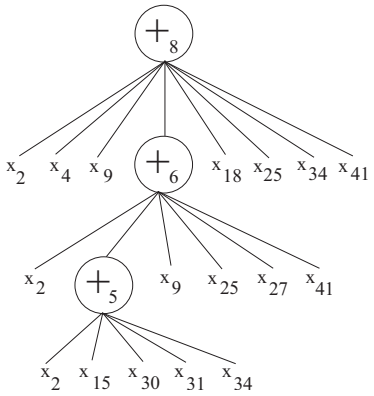

Figure 5. Best evolved tree for Probing collection.

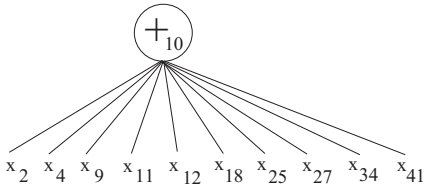

Figure 6. Best evolved tree for Attack collection.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Industry and Trade of the Czech Republic, under the grant No. FR-TI1/420 and by SGS, VŠB–Technical University of Ostrava, Czech Republic, under the grant No. SP/2010196.

REFERENCES

[1] A. Abraham: Evolutionary Computation in Intelligent Web Management, Evolutionary Computing in Data Mining, A. Ghosh and L. Jain (Eds.), Studies in Fuzziness and Soft Computing, Springer Verlag Germany, Chapter 8, pp. 189-210, 2004.

[2] T. Bäck, D. B. Fogel and Z. Michalewicz: Evolutionary Computation 1: Basic Algorithms and Operators, Institut of Physics Publishing, (eds.) (2000a).

[3] T. Bäck, D. B. Fogel and Z. Michalewicz: Evolutionary Computation 2: Advanced Algorithms and Operators, Institute of Physics Publishing, (eds.) (2000b).

[4] T. Brugger: Data Mining Methods for Network Intrusion Detection. University of California at Davis, 2004.

[5] S. Chebrolu, A. Abraham, J. P. Thomas: Feature Detection and Ensemble Design of Intrusion Detection Systems, Computers and security, In press.

[6] Y. Chen, A. Abraham, B. Yang: Hybrid flexible neural-tree-based intrusion detection systems. Internat. J. Intell. Systems 22, 337352. 2007.

[7] Y. Chen, B. Yang, J. Dong, and A. Abraham: Time-series Forcasting using Flexible Neural Tree Model, Information Science, In press.

[8] Y. Chen, B. Yang, J. Dong: Nonlinear system modeling via optimal design of neural trees, Int. J. Neural Syst. 14 (2), pp. 125137, 2004.

[9] Y. Chen, B. Yang, J. Dong: Evolving flexible neural networks using ant programming and PSO algorithm, International Symposium on Neural Networks (ISNN04), Lecture Notes on Computer Science, vol. 3173, pp. 211216, 2004.

[10] M. Debar, D. Becke, and A. Siboni: A Neural Network Component for an Intrusion Detection System. Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, 1992.

[11] Lincoln Laboratory: Darpa intrusion detection evaluation, February 2010. [Online]. Available: http://www.ll.mit.edu/mission/communications/ist/corpora /ideval/index.html

[12] G. F. Miller, P. M. Todd, and S. U. Hegde: Designing neural networks using genetic algorithms, in Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, pp. 379384, 1989.

[13] D. Montana and L. Davis; Training feedforward neural networks using genetic algorithms, in Proc. 11th Int. Joint Conf. Artificial Intelligence. San Mateo, CA: Morgan Kaufmann, pp. 762767, 1989.

[14] S. Mukkamala, A.H. Sung and A. Abraham: Intrusion Detection Using Ensemble of Soft Computing Paradigms, Advances in Soft Computing, Springer Verlag, Germany, pp. 239-248, 2003.

[15] S. Mukkamala, A. H. Sung, A. Abraham, and V. Ramos: Intrusion detection systems using adaptive regression splines, in ICEIS (3), pp. 2633, 2004.

[16] J. Platoš, V. Snášel, P. Krömer, and A. Abraham: Socioeconomic and Legal Implications of Electronic Intrusion, 1st ed. Information Science Reference, ch. Designing Light Weight Intrusion Detection Systems: Non-negative Matrix Factorization Approach, pp. 216229, April 2009.

[17] K. Shah, N. Dave, S. Chavan, S. Mukherjee, A. Abraham and S. Sanyal: Adaptive Neuro-Fuzzy Intrusion Detection System, IEEE International Conference on ITCC'04, Vol. 1, pp. 70-74, 2004.

[18] V. Snášel, J. Platoš, P. Krömer, and A. Abraham: Matrix factorization approach for feature deduction and design of intrusion detection systems, in IAS 2008 PROCEEDINGS, M. Rak, A. Abraham, and V. Casola, Eds., pp. 172-179, 2008.

[19] The UCI KDD Archive: Kdd cup data, February 2010. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[20] I. Vondrák: Genetic Algorithm and Neural Network Adaptation, Neural Network World, Vol. 4, Num. 2. IDG Czechoslovakia - VSP International Science Publishers, Netherlands, pp. 211-217, 1994.

[21] X. Yao: Evolving artificial neural networks. Proceedings of the IEEE 87(9), pp. 14231447, 1999.