Hybrid Flexible Neural-Tree-Based Intrusion Detection Systems

Yuehui Chen,[1,†] Ajith Abraham,[2,*] Bo Yang[1,‡]
[1]*School of Information Science and Engineering, Jinan University, Jinan 250022, P.R. China*
[2]*School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea*

An intrusion is defined as a violation of the security policy of the system, and, hence, intrusion detection mainly refers to the mechanisms that are developed to detect violations of system security policy. Current intrusion detection systems (IDS) examine all data features to detect intrusion or misuse patterns. Some of the features may be redundant or contribute little (if anything) to the detection process. The purpose of this study is to identify important input features in building an IDS that is computationally efficient and effective. This article proposes an IDS model based on a general and enhanced flexible neural tree (FNT). Based on the predefined instruction/operator sets, a flexible neural tree model can be created and evolved. This framework allows input variables selection, overlayer connections, and different activation functions for the various nodes involved. The FNT structure is developed using an evolutionary algorithm, and the parameters are optimized by a particle swarm optimization algorithm. Empirical results indicate that the proposed method is efficient.

1. INTRODUCTION

Traditional prevention techniques such as user authentication, data encryption, avoiding programming errors, and firewalls are used as the first line of defense for computer security. Recently, intrusion detection systems (IDS) have been used in monitoring attempts to break security, which provides important information for timely countermeasures. 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.

*Author to whom all correspondence should be addressed: e-mail: ajith.abraham@ieee.org.
†e-mail: yhchen@ujn.edu.cn.
‡e-mail: yangbo@ujn.edu.cn.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 22, 337–352 (2007)
 Published online in Wiley InterScience (www.interscience.wiley.com). • DOI 10.1002/int.20203

Various data mining techniques have been applied to intrusion detection because they have the advantage of discovering useful knowledge that describes a user's or program's behavior from large audit data sets. This article proposes a flexible neural tree (FNT)[1] 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, overlayer connections, and different activation functions for different nodes. In our previous work, the hierarchical structure was evolved using the probabilistic incremental program evolution algorithm (PIPE)[2] with specific instructions. In this work, the hierarchical structure is evolved using a tree-structure-based evolutionary algorithm. The fine-tuning of the parameters encoded in the structure is accomplished using particle swarm optimization (PSO).[3] The proposed method interleaves 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 fine-tunes the structure and rules' parameters. This loop continues until a satisfactory solution is found or a time limit is reached.

2. INTRUSION DETECTION SYSTEMS

Data mining approaches for intrusion detection were first implemented in mining audit data for automated models for intrusion detection.[4] The raw data are first converted into ASCII network packet information, which in turn is converted into connection level information. These connection level records contain connection features like service, duration, and so forth. Data mining algorithms are applied to these data to create models to detect intrusions.

Data mining approaches for intrusion detection were first implemented in Mining Audit Data for Automated Models for Intrusion Detection.[5] Raw data are converted into ASCII network packet information, which in turn is converted into connection level information. These connection level records contain within connection features like service, duration, and so forth. Data mining algorithms are applied to these data to create models to detect intrusions. Neural networks have been used both in anomaly intrusion detection and in misuse intrusion detection. In the first approach of neural networks[6] for intrusion detection, the system learns to predict the next command based on a sequence of previous commands by a user. Support vector machines (SVMs) have proven to be a good candidate for intrusion detection because of their training speed and scalability. Besides SVMs are relatively insensitive to the number of data points, and the classification complexity does not depend on the dimensionality of the feature space, so they can potentially learn a larger set of patterns and scale better than neural networks.[7] Neuro-fuzzy computing is a popular framework for solving complex problems. An adaptive neuro-fuzzy IDS is proposed by Shah et al.[8] Multivariate Adaptive Regression Splines (MARS) are an innovative approach that automates the building of accurate predictive models for continuous and binary dependent variables.[9] They excel at finding optimal variable transformations and interactions and the

complex data structure that often hides in high-dimensional data. An IDS based on MARS technology is proposed in Ref. 10. Linear Genetic Programming (LGP) is a variant of the conventional Genetic Programming (GP) technique that acts on linear genomes. Its main characteristics in comparison to tree-based GP lie in the fact that computer programs are evolved at the machine code level, using lower level representations for the individuals. This can tremendously hasten the evolution process as, no matter how an individual is initially represented, finally it always has to be represented as a piece of machine code, as fitness evaluation requires physical execution of the individuals. LGP based IDS is presented in Ref. 11.

Because the amount of audit data that an IDS needs to examine is very large even for a small network, analysis is difficult even with computer assistance because extraneous features can make it harder to detect suspicious behavior patterns. IDS must therefore reduce the amount of data to be processed. This is very important if real-time detection is desired. Reduction can occur in one of several ways. Data that are not considered useful can be filtered, leaving only the potentially interesting data. Data can be grouped or clustered to reveal hidden patterns; by storing the characteristics of the clusters instead of the data, overhead can be reduced. Finally, some data sources can be eliminated using feature selection. In the literature, there are some related works for feature reduction in IDS. A support vector machine technique is used to select the important features.[12] Feature deduction using the Markov blanket model and decision trees is presented in Ref. 13.

The novelty of this article is in the usage of the flexible neural tree model for selecting the important features and for detecting intrusions.

3. THE FLEXIBLE NEURAL TREE MODEL

In this research, a tree-structure-based encoding method with a specific instruction set is selected for representing a flexible neutron tree (FNT) model. The reason for choosing the representation is that the tree can be created and evolved using the existing or modified tree-structure-based approaches, that is, GP, PIPE, ant programming (AP), and so forth.

3.1. Flexible Neuron Instructor

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

$$S = F \bigcup T = \{+_2, +_3, \ldots, +_N\} \bigcup \{x_1, \ldots, x_n\} \tag{1}$$

where $+_i$ $(i = 2,3,\ldots,N)$ denote nonleaf nodes' instructions and take i arguments. $x_1, x_2, \ldots, x_n$ are leaf nodes' instructions and take no other arguments. The output of a nonleaf node is calculated as a flexible neuron model (see Figure 1). From this point of view, the instruction $+_i$ is also called a flexible neuron operator with i inputs.

In the process of creating a neural tree, if a nonterminal instruction, that is, $+_i$ $(i = 2,3,4,\ldots,N)$ is selected, i real values are randomly generated and used for

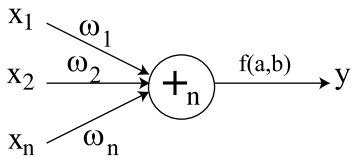

Figure 1. A flexible neuron operator.

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. Some examples of flexible activation functions are shown in Table I.

For developing the IDS, the following flexible activation function is used:

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

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

$$net_n = \sum_{j=1}^{n} w_j * x_j \tag{3}$$

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

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

A typical flexible neuron operator and a neural tree model are illustrated in Figures 1 and 2. The overall output of the flexible neural tree can be computed from left to right by the depth-first method, recursively.

3.2. Fitness Function

A fitness function maps FNT to scalar, real-valued fitness values that reflect the FNT's performances on a given task. First, the fitness functions should clearly reflect the classification error measures. A secondary non-user-defined objective for which algorithm always optimizes FNTs is the size of FNT usually measured by number of nodes. Among FNTs having equal fitness values, smaller FNTs are

Table I. The activation functions.

Gaussian function	$f(x) = \exp(-((x-a)^2/b^2))$
Flexible unipolar sigmoid function	$f(x,a) = 2\lvert a\rvert/(1+e^{-2\lvert a\rvert x})$
Flexible bipolar sigmoid function	$f(x,a) = (1-e^{-2xa})/a(1+e^{-2xa})$

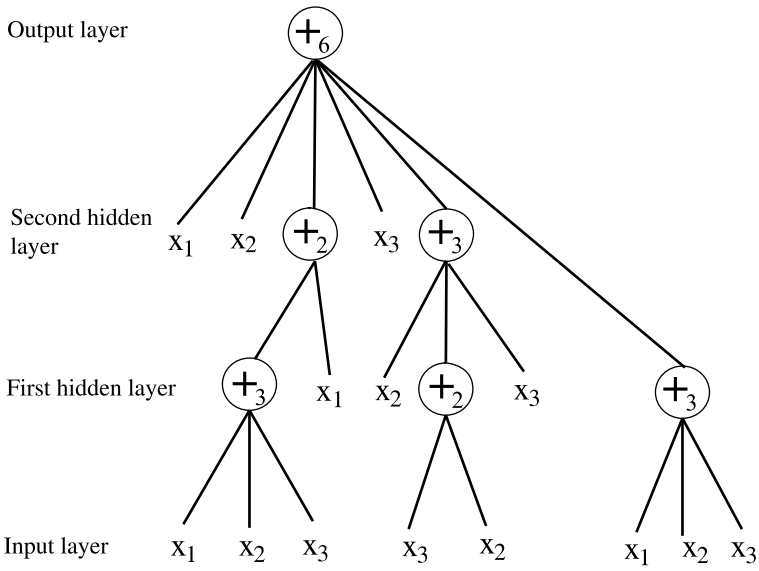

Figure 2. A typical representation of a neural tree with function instruction set $F = \{+_2, +_3, +_4, +_5, +_6\}$ and terminal instruction set $T = \{x_1, x_2, x_3\}$.

always preferred. A fitness function formulating the sum of positive and negative classification errors is used to design the IDS.

3.3. Tree Structure Optimization

Finding an optimal or near optimal neural tree is formulated as a product of evolution. A number of neural tree variation operators are developed as follows:

3.3.1. Mutation

Four different mutation operators were employed to generate offspring from the parents. These mutation operators are 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 all the terminal nodes: Select each and every terminal node in the neural tree and replace it with another terminal node.
(3) Growing: Select a random leaf in the hidden layer of the neural tree and replace it with a newly generated subtree.
(4) Pruing: Randomly select a function node in the neural tree and replace it with a terminal node.

Following the work of Chellapilla,[14] the neural tree operators were applied to each of the parents to generate an offspring using the following steps: (a) A

Poission random number N, with mean λ, was generated. (b) N random mutation operators were uniformly selected with replacement from above four mutation operator set. (c) These N mutation operators were applied in sequence one after the other to the parents to get the offspring.

3.3.2. Crossover

Select two neural trees randomly and select one nonterminal node in the hidden layer for each neural tree randomly, and then swap the selected subtree. The crossover operator is implemented with a predefined probability 0.3 in this study.

3.3.3. Selection

Evolutionary programming (EP) style tournament selection was applied to select the parents for the next generation.[14] Pairwise comparison is conducted for the union of μ parents and μ offspring. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's fitness is no smaller than the opponent's, it receives a selection. Select μ individuals out of parents and offspring that have the most wins to form the next generation. This is repeated for each generation until a predefined number of generations is reached or when the best structure is found.

3.4. Parameter Optimization with PSO

Particle swarm optimization (PSO)[3,15] conducts searches using a population of particles that correspond to individuals in an evolutionary algorithm. A population of particles is randomly generated initially. Each particle represents a potential solution and has a position represented by a position vector $\mathbf{x_i}$. A swarm of particles moves through the problem space, with the moving velocity of each particle represented by a velocity vector $\mathbf{v_i}$. At each time step, a function f_i representing a quality measure is calculated by using $\mathbf{x_i}$ as input. Each particle keeps track of its own best position, which is associated with the best fitness it has achieved so far in a vector $\mathbf{p_i}$. Furthermore, the best position among all the particles obtained so far in the population is kept track of as $\mathbf{p_g}$. In addition to this global version, another version of PSO keeps track of the best position among all the topological neighbors of a particle.

At each time step t, by using the individual best position, $\mathbf{p_i}$, and the global best position, $\mathbf{p_g}(\mathbf{t})$, a new velocity for particle i is updated by

$$\mathbf{v_i}(\mathbf{t}+\mathbf{1}) = \mathbf{v_i}(\mathbf{t}) + c_1\phi_1(\mathbf{p_i}(\mathbf{t}) - \mathbf{x_i}(\mathbf{t})) + c_2\phi_2(\mathbf{p_g}(\mathbf{t}) - \mathbf{x_i}(\mathbf{t})) \tag{5}$$

where c_1 and c_2 are positive constant and ϕ_1 and ϕ_2 are uniformly distributed random numbers in $[0,1]$. The term $\mathbf{v_i}$ is limited to the range of $\pm\mathbf{v_{max}}$. If the velocity violates this limit, it is set to its proper limit. Changing velocity this way enables the particle i to search around its individual best position, $\mathbf{p_i}$, and global

best position, $\mathbf{p_g}$. Based on the updated velocities, each particle changes its position according to the following equation:

$$\mathbf{x_i(t+1)} = \mathbf{x_i(t)} + \mathbf{v_i(t+1)} \tag{6}$$

Based on Equations 7 and 8, the population of particles tend to cluster together with each particle moving in a random direction. Most attempts to improve the velocity update formula Equation 7 can be captured by the following formula[3]:

$$\mathbf{v_i(t+1)} = \chi(\omega \mathbf{v_i(t)} + c_1\phi_1(\mathbf{p_i(t)} - \mathbf{x_i(t)}) + c_2\phi_2(\mathbf{p_g(t)} - \mathbf{x_i(t)})) \tag{7}$$

where two new parameters, χ and ω, are also real numbers. The parameter χ controls the magnitude of $\mathbf{v}$, whereas the inertia weight ω weights the magnitude of the old velocity $\mathbf{v_i(t)}$ in the calculation of the new velocity $\mathbf{v_i(t+1)}$.

3.5. Procedure of the General Learning Algorithm

The general learning procedure for constructing the FNT model can be described as follows.

(1) Create an initial population randomly (FNT structures and its corresponding parameters).
(2) Structure optimization is achieved by the neural tree variation operators as described in Subsection 3.3.
(3) If a better structure is found, then go to step 4; otherwise go to step 2.
(4) Parameter optimization is achieved by the PSO algorithm as described in Subsection 3.4. In this stage, the architecture of FNT model is fixed, and it is the best tree developed during the end of run of the structure search. The parameters (weights and flexible activation function parameters) encoded in the best tree formulate a particle. The PSO algorithm works as follows:
   (a) Initial population is generated randomly. The learning parameters c_1 and c_2 in PSO should be assigned in advance.
   (b) The objective function value is calculated for each particle.
   (c) Modification of search point. The current search point of each particle is changed using Equations 7 and 6.
   (d) If the maximum number of generations is reached or no better parameter vector is found for a significantly long time (100 steps), then stop; otherwise go to step (b).
(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 a satisfactory solution is found, then the algorithm is stopped; otherwise go to step 2.

4. FEATURE SELECTION AND CLASSIFICATION USING FNT PARADIGMS

4.1. The Data Set

The data for our experiments were prepared by the 1998 DARPA intrusion detection evaluation program by MIT Lincoln Lab. The data set contains 24 attack types that could be classified into four main categories, namely, *Denial of Service* (*DOS*), *Remote to User* (*R2L*), *User to Root* (*U2R*), and *Probing*. The original

data contain 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 from attacks. These features are either nominal or numeric. Some features examine only the connection in the past two seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, and so forth. These are called same host features. Some features examine only the connections in the past two seconds that have the same service as the current connection and are called same service features. Some other connection records were also stored by the destination host, and features were constructed using a window of 100 connections to the same host instead of a time window. These are called host-based traffic features. R2L and U2R attacks do not have any sequential patterns like DOS and Probe because the former attacks have the attacks embedded in the data packets whereas the later attacks have many connections in a short amount of time. So some features that look for suspicious behavior in the data packets such as number of failed logins are constructed and these are called contents features. The data for our experiments contain 11,982 randomly generated records having 41 features.[16] The labels of the 41 features and their corresponding networks data features are shown in Table II.

This data set has five different classes, namely *Normal*, *DOS*, *R2L*, *U2R*, and *Probe*. The training and test comprise 5092 and 6890 records, respectively. All the IDS models were trained and tested with the same set of data. As the data set has five different classes, we performed a five-class binary classification. The

Table II. Network data feature labels.

Label	Feature	Label	Feature
x_1	*duration*	x_2	*protocol-type*
x_3	*service*	x_4	*flag*
x_5	*src_bytes*	x_6	*dst_bytes*
x_7	*land*	x_8	*wrong_fragment*
x_9	*urgent*	x_{10}	*hot*
x_{11}	*num_failed_logins*	x_{12}	*logged_in*
x_{13}	*num_compromised*	x_{14}	*root_shell*
x_{15}	*su_atempted*	x_{16}	*num_root*
x_{17}	*num_file_creations*	x_{18}	*num_shells*
x_{19}	*num_acess_files*	x_{20}	*num_outbound_cmds*
x_{21}	*is_host_login*	x_{22}	*is_guest_login*
x_{23}	*count*	x_{24}	*srv_count*
x_{25}	*serror_rate*	x_{26}	*srv_serror_rate*
x_{27}	*rerror_rate*	x_{28}	*srv_rerror_rate*
x_{29}	*smae_srv_rate*	x_{30}	*diff_srv_rate*
x_{31}	*srv_diff_host_rate*	x_{32}	*dst_host_count*
x_{33}	*dst_host_srv_count*	x_{34}	*dst_host_same_srv_rate*
x_{35}	*dst_host_diff_srv_rate*	x_{36}	*dst_host_same_srv_port_rate*
x_{37}	*dst_host_srv_diff_host_rate*	x_{38}	*dst_host_serror_rate*
x_{39}	*dst_host_srv_serror_rate*	x_{40}	*dst_host_rerror_rate*
x_{41}	*dst_host_srv_rerror_rate*		

Table III. Parameters used in the flexible neural tree model.

Parameter	Initial values
Population size *PS*	100
Crossover probability	0.3
Opponent q in tournament selection	30
Maximum local search steps	2000
Terminate steps in local search	100
Initial connection weights	rand[−1,1]
Initial parameters a_i and b_i	rand[0,1]

normal data belong to class 1, *Probe* belong to class 2, *DOS* belong to class 3, *U2R* belong to class 4, and *R2L* belong to class 5.

The initial parameters used for each experiment are listed in Table III.

4.2. Feature/Input Selection with FNT

It is often a difficult task to select variables (features) for the classification problem, especially when the feature space is large. A fully connected NN classifier usually cannot do this. In the perspective of the FNT framework, the nature of the model construction procedure allows the FNT to identify important input features in building an IDS that is computationally efficient and effective.

The mechanisms of input selection in the FNT constructing procedure are as follows. (1) Initially the input variables are selected to formulate the FNT model 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 generation 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.

4.3. Modeling IDS Using FNT with 41 Input-Variables

For this simulation, the original 41 input variables are used for constructing an FNT model. An FNT classifier was constructed using the training data, and then the classifier was used on the test data set to classify the data as an attack or normal data. The instruction sets used to create an optimal FNT classifier are $S = F \bigcup T = \{+_5, \ldots, +_{20}\} \bigcup \{x_1, x_2, \ldots, x_{41}\}$, where x_i $(i = 1,2,\ldots,41)$ denotes the 41 features.

The required number of iterations for structure and parameter optimization for each of the FNT classifiers are listed in Table IV. The optimal FNTs for classes 1–5 are shown in Figures 3–5. It should be noted that the important features for constructing the FNT model were formulated in accordance with the procedure mentioned in the previous section. These important variables are shown in Table V. Table VIII, below, depicts the detection performance of the FNT by using the original 41 variable data set.

Table IV. Iterations for structure and parameter optimization.

Class	Structure optimization	Parameter optimization
Class 1	95	1789
Class 2	89	1602
Class 3	91	1539
Class 4	48	1892
Class 5	64	1920

4.4. Modeling IDS with Input Variables Selected by the Decision Tree Approach

The important variables for intrusion detection were decided by their contribution to the construction of the decision tree.[13] Variable rankings were generated in terms of percentages. The variables that had 0.00% rankings and were considered only the primary splitters were eliminated.[13] This resulted in a reduced 12-variable data set with x_3, x_5, x_6, x_{12}, x_{23}, x_{24}, x_{25}, x_{28}, x_{31}, x_{32}, x_{33}, and x_{35} as variables. Further, the FNT classifier was constructed using the 12-variable data set (training data), and then the test data were passed through the save trained model. The instruction sets used to create an optimal neural tree model are $S = F \bigcup T = \{+_2, \ldots, +_{10}\} \bigcup \{x_3, x_5, x_6, x_{12}, x_{23}, x_{24}, x_{25}, x_{28}, x_{31}, x_{32}, x_{33}, x_{35}\}$.

The iterations for structure and parameter optimization for each of the FNT classifiers are listed in Table VI. The optimal FNTs for classes 1–5 are shown in Figures 6–8. It should be noted that the important features for constructing the FNT model were recognized automatically one more time. The important variables selected by the FNT model are shown in Table VII. Table VIII depicts the performance of the FNT by using the reduced 12-variable data set.

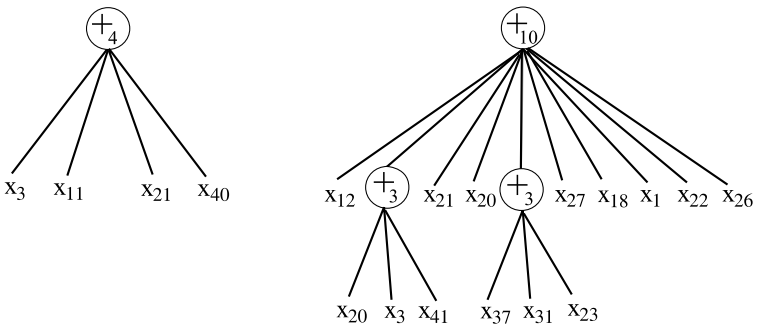

Figure 3. The evolved FNT for class 1 and class 2 with 41 input variables.

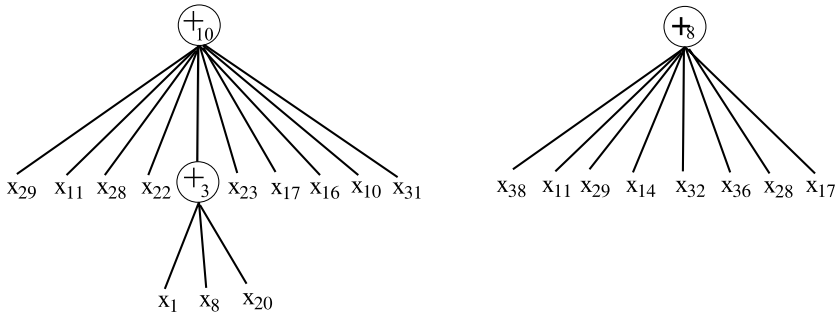

Figure 4. The evolved FNT for class 3 and class 4 with 41 input variables.

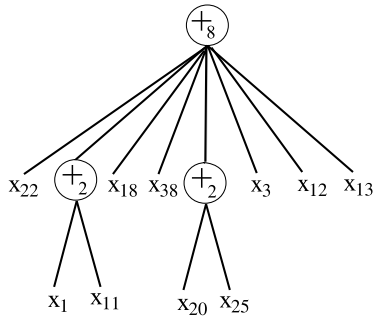

Figure 5. The evolved FNT for class 5 with 41 input variables.

Table V. The important features selected by the FNT algorithm.

Class	Important variables
Class 1	$x_3, x_{11}, x_{21}, x_{40}$
Class 2	$x_1, x_3, x_{12}, x_{18}, x_{20}, x_{21}, x_{23}, x_{26}, x_{27}, x_{31}, x_{37}, x_{41}$
Class 3	$x_1, x_8, x_{10}, x_{11}, x_{16}, x_{17}, x_{20}, x_{12}, x_{23}, x_{28}, x_{29}, x_{31}$
Class 4	$x_{11}, x_{14}, x_{17}, x_{28}, x_{29}, x_{32}, x_{36}, x_{38}$
Class 5	$x_1, x_3, x_{11}, x_{12}, x_{13}, x_{18}, x_{20}, x_{22}, x_{25}, x_{38}$

Table VI. The iterations in structure and parameter optimization.

Class	Structure optimization	Parameter optimization
Class 1	45	1745
Class 2	49	1403
Class 3	31	1547
Class 4	28	1678
Class 5	54	1834

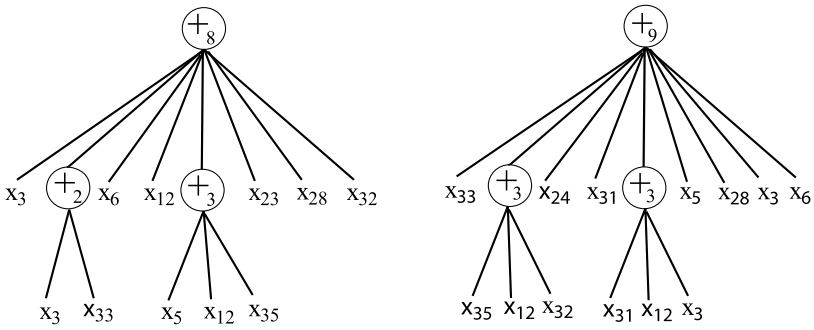

Figure 6. The evolved FNT for classes 1 and 2 with 12 input variables.

4.5. Modeling IDS Using Neural Networks without Input Variable Selection

For comparison purposes, a neural network classifier trained by a PSO algorithm with flexible bipolar sigmoid activation functions was constructed using the same training data sets, and then the neural network classifier was used on the test data set to detect the different types of attacks. All the input variables were used for the experiments.

Before describing details of the algorithm for training NN classifier, the issue of coding is presented. Coding concerns the way the weights and the flexible activation function parameters of NN are represented by individuals or particles. A float point coding scheme is adopted here. For NN coding, suppose there are M nodes in a hidden layer and one node in the output layer and n input variables; then the number of total weights is $n * M + M * 1$, the number of thresholds is $M + 1$,

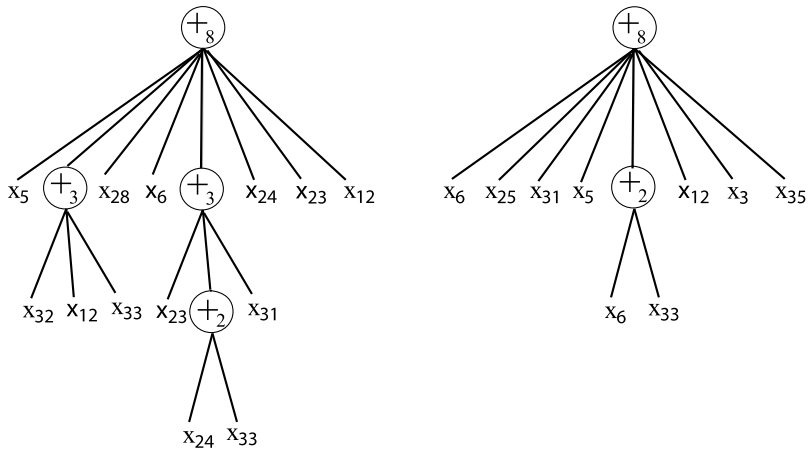

Figure 7. The evolved FNT for class 3 and class 4 with 12 input variables.

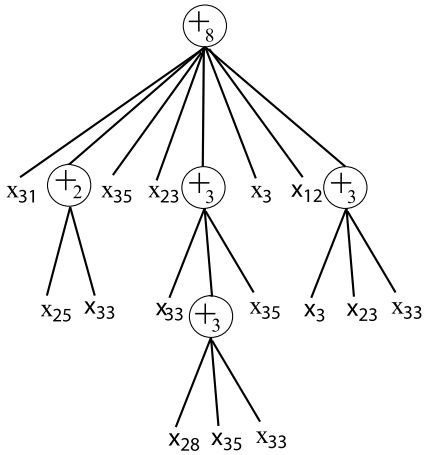

Figure 8. The evolved FNT for class 5 with 12 input variables.

and the number of flexible activation function parameters is $M + 1$; therefore the total number of free parameters in an NN to be coded is $n * M + M + 2(M + 1)$. These parameters are coded into an individual or particle orderly.

The simple loop of the proposed training algorithm for neural network is as follows:

S1 Initialization. Initial population is generated randomly. The learning parameters c_1 and c_2 in PSO should be assigned in advance.

S2 Evaluation. The objective function value is calculated for each particle.

S3 Modification of search point. The current search point of each particle is changed using Equations 7 and 6.

S4 If maximum number of generations is reached or no better parameter vector is found for a significantly long time (100 steps), then stop; otherwise go to step **S2**.

Table VII. Important features for 12-variable data set.

Class	Important variables
Class 1	$x_3, x_5, x_6, x_{12}, x_{23}, x_{28}, x_{32}, x_{33}, x_{35}$
Class 2	$x_3, x_5, x_6, x_{12}, x_{24}, x_{28}, x_{31}, x_{32}, x_{33}, x_{35}$
Class 3	$x_5, x_6, x_{12}, x_{23}, x_{24}, x_{28}, x_{31}, x_{32}, x_{33}$
Class 4	$x_3, x_5, x_6, x_{12}, x_{25}, x_{31}, x_{33}, x_{35}$
Class 5	$x_3, x_{12}, x_{23}, x_{25}, x_{28}, x_{31}, x_{33}, x_{35}$

Table VIII. Detection performance using FNT model with 41 and 12 input variables.

Attack class	41-variable data set	12-variable data set
Normal	**99.19%**	97.98%
Probe	**98.39%**	97.46%
DOS	**98.75%**	94.63%
U2R	99.70%	**99.76%**
R2L	**99.09%**	98.99%

Table IX depicts the performance of the neural network by using the original 41-variable data set and the 12-variable reduced data set.

5. CONCLUSIONS

In this article, we presented a Flexible Neural Tree (FNT) model for Intrusion Detection Systems (IDS) with a focus on improving the intrusion detection performance by reducing the input features and hybrid approaches for combining base classifiers. We have also demonstrated the performance using different reduced data sets. As evident from Tables VIII and IX, the proposed flexible neural tree approach seems to be very promising. The FNT model was able to reduce the number of variables to 4, 12, 12, 8, and 10 (using 41 input variables) and 9, 10, 9, 8, and 8 (using 12 input variables) for classes 1–5, respectively. Using 41 variables, the FNT model gave the best accuracy for the detection of most of the classes (except U2R). Using 41 input variables, although the hybrid model seems to work very well for most of the attack classes, the direct NN classifier outperformed the FNT approach for U2R attack. For the 12-variable reduced data set, the direct NN approach outperformed the FNT model for DOS, U2R, and R2L attacks.

As suggested in the literature,[17] by increasing the weight of false negative errors, the detection accuracy could be improved. In our research, we avoided any such bias toward any particular error. Instead a 0.5 bias was provided for both false positive and negative errors just to measure the neutral performance of the

Table IX. Detection performance using the NN classifier with 41 and 12 input variables.

Attack class	41-variable data set (original data)	12-variable data set (decision tree reduced data)
Normal	**95.69%**	95.59%
Probe	**95.53%**	95.08%
DOS	90.41%	**100%**
U2R	100%	100%
R2L	98.10%	**99.25%**

Table X. The false positive/negative errors using the 41-variable data set by the FNT algorithm.

Attack class	False positive error	False negative error
Normal	0.0581%	0.7837%
Probe	1.3943%	0.2160%
DOS	0.6241%	0.6241%
U2R	0.2177%	0.0726%
R2L	0.7547%	0.1597%

classification algorithm. The achieved false positive/negative errors using the 41-variable data set by the FNT algorithm are depicted in Table X.

One of the problems with most of the current intrusion detection systems is the alarming rate of false positives. It is to be noted that the proposed FNT model could detect the normal mode within 99.19% accuracy by using only four input variables. This field is developing continuously. More data mining techniques should be investigated and their efficiency should be evaluated as intrusion detection models.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (NSFC), Project No.69902005, and Provincial Natural Science Foundation of Shandong, Project No. Y2001G09.

References

1. Chen Y, Yang B, Dong J, Abraham A. Time-series forcasting using flexible neural tree model. Inform Sci 2005;174:219–235.
2. Salustowicz RP, Schmidhuber J. Probabilistic incremental program evolution. Evol Comput 1997;2:123–141.
3. Kennedy J. Particle swarm optimization. In: Proc IEEE Int Conf on Neural Networks, vol IV; 1995. pp 1942–1948.
4. Barbara D, Couto J, Jajodia S, Wu N. ADAM: A testbed for exploring the use of data mining in intrusion detection. SIGMOD Record 2001;30:15–24.
5. Lee W. A data mining framework for constructing features and models for intrusion detection systems. PhD thesis, Computer Science Department, Columbia University, June 1999.
6. Fox KL, Henning RR, Reed JH, Simonian R. A neural network approach towards intrusion detection. In: Proc 13th National Computer Security Conf, Washington, DC; 1990. pp 125–134.
7. Mukkamala S, Sung AH, Abraham A. Intrusion detection using ensemble of soft computing paradigms. In: Proc Third Int Conf on Intelligent Systems Design and Applications, Advances in Soft Computing. Berlin: Springer Verlag; 2003. pp 239–248.
8. Shah K, Dave N, Chavan S, Mukherjee S, Abraham A, Sanyal S. Adaptive neuro-fuzzy intrusion detection system. In: IEEE Int Conf on Information Technology: Coding and

Computing (ITCC'04), vol 1. Los Alamitos, CA: IEEE Computer Society Press; 2004. pp 70–74.
9. Friedman JH. Multivariate adaptative regression splines. Ann Stat 1991;19:1–141.
10. Mukkamala S, Sung AH, Abraham A, Ramos V. Intrusion detection systems using adaptive regression splines. In: Seruca I, Cordeiro J, Hammoudi S, Filipe J, editors. Sixth Int Conf on Enterprise Information Systems, ICEIS'04, Portugal, Enterprise Information Systems VI. Berlin: Springer-Verlag; 2006. pp 211–218.
11. Mukkamala S, Sung AH, Abraham A. Modeling intrusion detection systems using linear genetic programming approach. In: Orchard R, Yang C, Ali M, editors. Proc 17th Int Conf on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Innovations in Applied Artificial Intelligence. Lecture Notes in Computer Science 3029. Berlin: Springer Verlag; 2004. pp 633–642.
12. Mukkamala S, Sung AH. Feature selection for intrusion detection using neural networks and support vector machines. Transport Res Rec 2003;1822:33–39.
13. Chebrolu S, Abraham A, Thomas JP. Feature detection and ensemble design of intrusion detection systems. Comput Secur 2005;24:295–307.
14. Chellapilla K. Evolving computer programs without subtree crossover. IEEE Trans Evol Comput, 1997;1:209–216.
15. Yoshida H, Kawata K, Fukuyama Y, Takayama S, Nakanishi Y. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Trans Power Syst 2000;15:1232–1239.
16. KDD cup 99. Available at: http://www.kdnuggets.com/datasets/kddcup.html.
17. Joo D, Hong T, Han I. The neural network models for IDS based on the asymmetric costs of false negative errors and false positive errors. Expert Syst Appl 2003;25:69–75.