Heavy Facilities Tension Prediction Using Flexible Neural Trees

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

Petr Fiala
VITKOVICE UAM a.s.,
Mezirka 775/1, 602 00 Brno, Czech Republic
petr.fiala@vitkovice.cz

***Abstract*—In this article we show the usage of soft-computing methods to solve the real problem of computation of tension in facilities working under very hard conditions in industrial environment. Because the classical mathematical approaches such as Finite Element Method (FEM) are very time consuming, the more progressive soft-computing methods are on the place. We have proposed two step algorithm based on Flexible Neural Tree (FNT) and Particle Swarm Optimization (PSO) which is more efficient then typical approach (FEM). Flexible neural tree is hierarchical neural network like structure, which is automatically created and optimized using evolutionary like algorithms to solve the given 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 good enough to be used in real environments.**

***Keywords*-Real industry environment, Soft-computing, Flexible Neural Tree, Particle Swarm Optimization**

I. INTRODUCTION

Energetics, chemical and food industry, rolling/mills and many other industries uses large amount of facilities under very hard conditions. All facilities undergo many changes during their lifetimes. Their operators need information on gradual damage of materials of such facilities. Gathered information are used in process of control to reduce possible damage and to achieve optimal lifetime of individual facilities. For example, a large enough time period after which the facility's owners are willing to pay necessary repairs and preventive services can play an important role in the lifetime of particular facility.

Diagnostic systems based on mathematical description of material damaging process may be used for collecting of necessary information about trends and degree of material and function damage. One of the parameter needed by this mathematical description is time flow of relative deformation ϵ and tension σ in construction points with maximal material damage due to load during running and shut downs. The major wears are losses of facility integrity, fatigue of materials, corrosion, and creep, etc., including their mutual interference.

Diagnostic systems can be used in *on-line* and *off-line* mode. Typically, an *off-line* diagnosis of measured data from stored files is performed once per day. This approach is used in such cases, when multiple working load cycles are performed during a day, e.g rolling mill, derrick and giant machinery in open pit mines. The first mentioned *on-line* diagnostic approach can be used in such case when, e.g., a machine works with limited and rated power for many days, even more that one month. In this case, *off-line* analysis provides a good result only if the working load cycles are closed. Then the *off-line* analysis is performed repeatedly, e.g. every week from January,1st of the current year till the evaluation day, and achieved results of defined periods usually update the results achieved during previous measurement.

The computation of the relative deformation ϵ and tension σ within selected construction points can be done by using simplified, but precise enough approach based on analytical simulations. These simulations try to express a dependence among input parameters (ϵ, σ) and measured parameters of working load like a temperature T, an overpressure p and a flow rate Q of running medium inside conduits, chambers and heat transfer tubes in energetics facilities.

Next, the computation of the relative deformation ϵ and tension σ with respect to working load is varying in time and represents a dynamic system. This is more important especially in case of complex geometric construction shapes of a particular facility. Then the computation can be performed by using the Finite Element Method (FEM) [15]. Unfortunately, FEM is very time-consuming and cannot be effectively used in continuous computation of ϵ and σ as a part of diagnostic system.

Because of very high time complexity of FEM, more progressive approach may be used. The proposed method is based on two-step algorithm. In the first step, a time flow of relative deformation ϵ and tension σ is computed by using FEM. These results represent an input set for the second step in the so called *learning* phase. Many soft-computing methods suitable for this problem exists, e.g. Artificial Neural Networks [7], Self Organized Maps [9], Flexible Neural Trees [3], [4], [5], [13], Fuzzy rules based system [8], etc. The paper is primary focused on the utilization of Flexible Neural Trees in the project *FR-*

TI1/86 New design approach of energetic components and steel structures with high utility parameters. Prepared soft-computing models can be used as a part of *on-line* and *off-line* diagnostic systems, respectively. The main contribution of such utilization consists in an evaluation of remaining lifetimes of various facilities.

Examples of the time flow calculation for relative deformation ϵ and tension σ in relation with typical working load changes T, p, Q using FEM is shown in the Figures 1 and 2. Both images show a connection of membrane wall tube and chamber body.

Rest of paper is organized as follows. Section II contains description of theoretical background about flexible neural trees and soft computing method used for FNT parameters tuning. Section III contains description of experimental data used, Section IV reviews achieved results and in the last section are conclusions and description of future work.

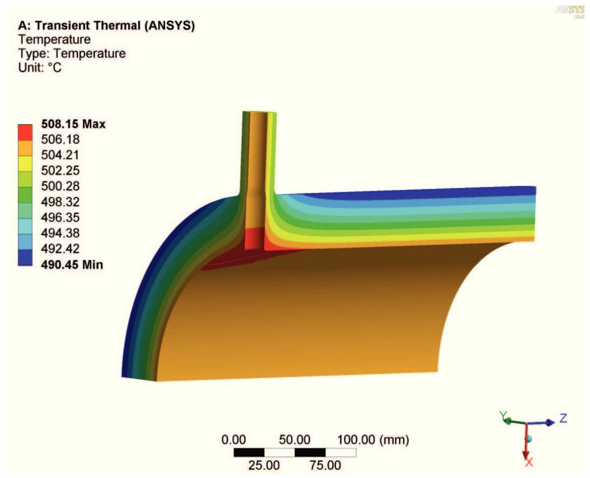

Figure 1. Temperature distribution T in connection of membrane wall tube and chamber body

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 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.

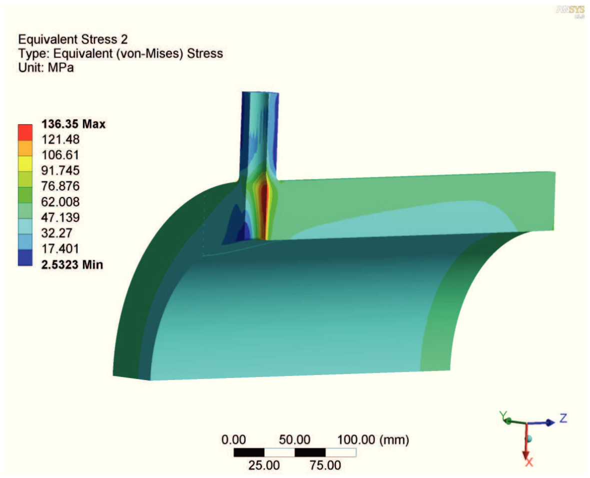

Figure 2. Tension intensity distribution i in connection of membrane wall tube and chamber body

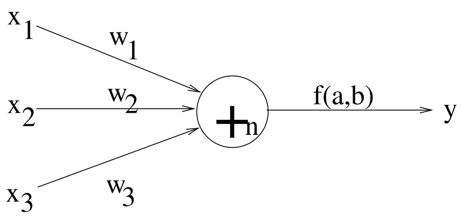

Figure 3. A flexible neuron operator (instructor)

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).

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

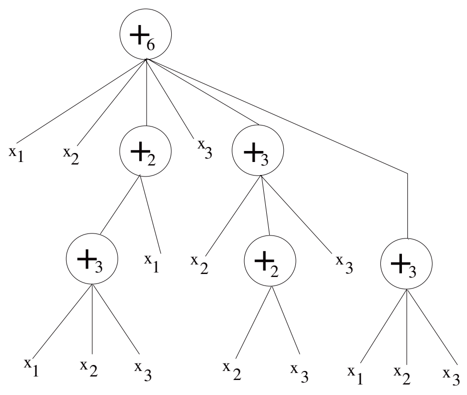

Figure 4. A typical representation of neural tree with function instruction set $F = \{+_2, +_3, +_4, +_5, +_6\}$, and terminal instruction set $T = \{x_1, x_2, x_3\}$

argument each. The output of a non-leaf node is calculated as a flexible neuron model (see figure 3). 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:

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

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

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

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

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

A typical evolved flexible neural tree model is shown as Figure 4. The overall output of flexible neural tree can be computed from left to right by depth-first method, recursively.

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):

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

or Root Mean Square Error ($RMSE$):

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

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.

A. 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 [3]:

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 Particle Swarm Optimization. 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 [3], [6].

Genetic Algorithms and FNT Structure Optimization

In this work, we use genetic algorithms [1], [2] for flexible neural tree structure 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.

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.

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 [12], [10], [14], [11].

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.

Particle Swarm Optimization and Parameters Tuning

The particle swarm optimization (PSO) conducts searches using a population of particles which correspond to individuals in evolutionary algorithm (EA). A population of particles is randomly generated initially. Each particle represents a potential solution and has a position represented by a position vector x_i. A swarm of particles moves through the problem space, with the moving velocity of each particle represented by a velocity vector v_i. At each time step, a function f_i representing a quality measure is calculated by using 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 p_i. Furthermore, the best position among all the particles obtained so far in the population is kept track of as 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, p_i, and the global best position, $p_g(t)$, a new velocity for particle i is updated by:

$$v_i(t+1) = \omega v_i(t) + c_1\phi_1(p_i(t) - x_i(t)) + c_2\phi_2(p_g(t) - x_i(t)) \quad (7)$$

where c_1, c_2 (acceleration) and ω (inertia) are positive constant and ϕ_1 and ϕ_2 are uniformly distributed random number in [0,1]. The term v_i is limited to the range of $\pm v_{max}$. 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, p_i, and global best position, p_g. Based on the updated velocities, each particle changes its position according to the following equation:

$$x_i(t+1) = x_i(t) + v_i(t+1). \quad (8)$$

The particles in this case represent encoded vectors of flexible neural tree parameters and encoded vectors of tree weights respectively.

III. Testing data

Data for experiments were extracted from the real measured data. Three input parameters were used - temperature, overpressure and flow rate. The data are stored real time when a large enough change was made. Therefore, the parameters are always stored in the same time. The first step in data preparation is to convert the data into the same time axis. This was made by linear interpolation of the values into the defined time grid. The density of the grid defines the number of input values. We used nonlinear grid, because of nature of the data.

Our task is to approximate tension during several states of the power plant generator. The first state is start-state, when the generator started after it was off for a long time. At the end of the start state the generator has the maximal power - is in the full-state. When the generator is off less than one hour then this situation is called very hot state. A hot state means, that the generator is off less then three hours and the warm state means that the generator was off less than twelve hours. The final state - cold state - means that the generator was off more than twelve hours and have to be started. Normal sequence of the states is start state, several very hot, hot and warm states interleaved by long time on full state and finally a cold state. Between start state and the sequence of other states the generator is of the full power without significant change for a long time and therefore no data are measured and stored into the data files. Interpolated data in these situations will lead into long sequences of non-changing data flow, which are not needed during the model learning. This is the reason for selection of non-linear time grid.

In our experiments, we used a long sequence of data which contains start state, very hot state, hot state, warm state and cold state. Between start and cold state and other states is always a full-state. The collection contains 808 records - each record contains temperature, overpressure and flow rate. The training data are depicted in Figure 5 (temperature), Figure 6 (overpressure) and Figure 7 (flow rate).

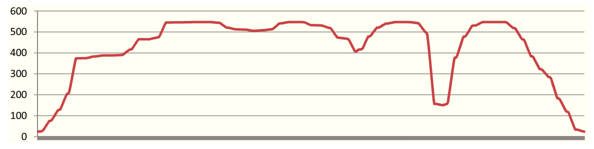

Figure 5. Temperature data used for learning [°C]

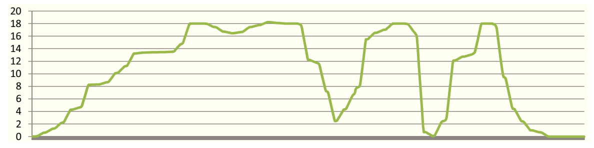

Figure 6. Overpressure data used for learning [MPa]

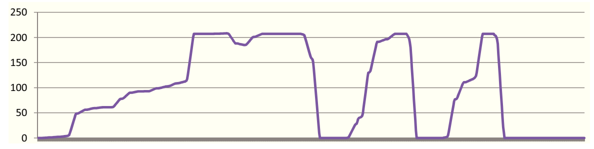

Figure 7. Flow rate data used for learning [t/h]

Table I
PARAMETERS USED IN THE FLEXIBLE NEURAL TREE MODEL

Parameter	Initial value
Initial population size	100
Overall mutation probability	40%
Overall crossover probability	50%
Elitism	30%
Initial connection weights	rand[0,1]
Initial parameters, a_i and b_i	rand[0,1]
PSO - Particles count	40
PSO - Acceleration constant c_1	0.5
PSO - Acceleration constant c_2	0.3
PSO - Inertial constant ω	0.9
PSO - Maximum number of iterations	1000

As may be seen, the full-states are not visible, because we use non-linear time grid and because these data are not useful during the learning. This data will be used in the learning phase of the algorithm. Because we need to compare our model with a more sophisticated approach, we computed the tension using FEM. The calculation for satisfactory precision takes several hours using this method.

IV. EXPERIMENTS AND RESULTS

The aim of this work was to train flexible neural tree on real data acquired from nuclear power plant. As was mentioned in Section III there are three input parameters and one output parameter - tension. The flexible neural tree was generated according to algorithm described in Section II. The structure of tree was optimized using the genetic algorithms and the weights of the tree nodes as well as flexible parameters were optimized using particle swarm optimization technique. In addition, the parameters used for experiment is listed in Table I.

The best resulting tree (with best fitness reached) was achieved in population 14. The total number of tree nodes is 34. Same data was used for training as well as for testing. The root mean squared error (RMSE) was used to study the performance of learned tree. **The RMSE for tested data is 0.030111**. Figure 8 depicts the result of our experiment.

V. CONCLUSIONS

In this paper, we presented a Flexible Neural Tree approach to approximate tension during several states of the power plant generator. We have used genetic algorithms for tree structure optimization and Particle Swarm Optimization for connection weights and activation function parameters tuning. All the experiments were done on real environment data collection and the results show good prediction accuracy. We have proved that the Flexible Neural Tree model can be used in a real industry environment with good approximation results. In future work we would like to compare this approach toward the other methods such

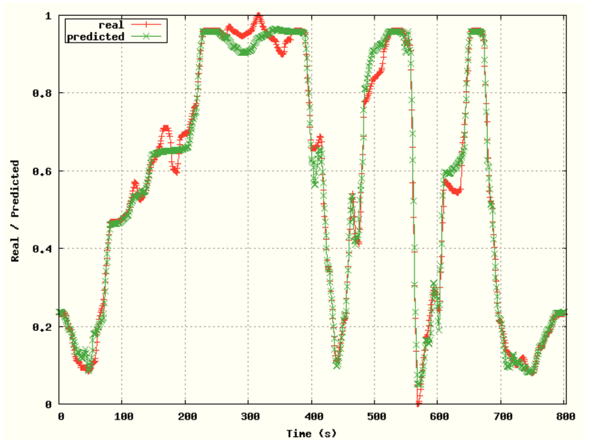

Figure 8. Comparison of real and predicted tension.

as Self Organizing Maps, Adaptive Neuro-Fuzzy Inference Systems and others.

Acknowledgment

This work was supported by the Ministry of Industry and Trade of the Czech Republic, under the grant no. FR-TI1/086.

References

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

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

[3] Y. Chen, B. Yang, J. Dong, and A. Abraham: Time-series forecasting using flexible neural tree model, Information Sciences, Volume 174, Issues 3-4, 11, Pages 219-235, 2005.

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

[5] 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.

[6] Y. Chen, B. Yang and A. Abraham: Flexible Neural Trees Ensemble for Stock Index Modeling, Neurocomputing, Vol. 70, pp. 697-703, 2007.

[7] H. Debar, M. Becker, and D. Siboni: A Neural Network Component for an Intrusion Detection System. Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 240-250, 1992.

[8] J. S. R. Jang: ANFIS: Adaptive-Network-Based Fuzzy Inference System. IEEE Transactions on Systems and Cybernetics SMC, Vol. 23, No. 3, pp. 665, 1993.

[9] T. Kohonen: Self-Organizing Maps. Springer Series in Information Sciences, Vol. 30, Edition 3rd, pages ALL, 2001.

[10] 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.

[11] Z. Michalewicz, Genetic Algorithmsq + Data Structuress Evolution Programs, Springer-Verlag, New York, 1996.

[12] 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.

[13] T. Novosád, J. Platoš, V. Snášl and Abraham, A.: Fast Intrusion Detection System based on Flexible Neural Tree, Sixth International Conference on Information Assurance and Security (IAS), USA, IEEE, ISBN 978-1-4244-7408-0, pp. 142-147, 2010.

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

[15] O. C. Zienkiewicz and R. L. Taylor, The Finite Element Method for Solid and Structural Mechanics. Elsevier Butterworth Heinemann, 2005.