2015 International Nuclear Atlantic Conference - INAC 2015
São Paulo, SP, Brazil, October 4-9, 2015
ASSOCIAÇÃO BRASILEIRA DE ENERGIA NUCLEAR - ABEN
ISBN: **978-85-99141-06-9**

ANT COLONY OPTIMIZATION AND NEURAL NETWORKS APPLIED TO NUCLEAR POWER PLANT MONITORING

Gean Ribeiro dos Santos, Delvonei Alves de Andrade and Iraci Martinez Pereira

Instituto de Pesquisas Energéticas e Nucleares (IPEN / CNEN - SP)
Av. Professor Lineu Prestes 2242
05508-000 São Paulo, SP
gean@usp.br
delvonei@ipen.br
martinez@ipen.br

ABSTRACT

A recurring challenge in production processes is the development of monitoring and diagnosis systems. Those systems help on detecting unexpected changes and interruptions, preventing losses and mitigating risks. Artificial Neural Networks (ANNs) have been extensively used in creating monitoring systems. Usually the ANNs created to solve this kind of problem are created by taking into account only parameters as the number of inputs, outputs, and hidden layers. The result networks are generally fully connected and have no improvements in its topology. This work intends to use an Ant Colony Optimization (ACO) algorithm to create a tuned neural network. The ACO search algorithm will use Back Error Propagation (BP) to optimize the network topology by suggesting the best neuron connections. The result ANN will be applied to monitoring the IEA-R1 research reactor at IPEN.

1. INTRODUCTION

The use of sensors has proven indispensable in many areas of the industry such as automation process, manufacturing, robotics, experimental engineering, energy industry, etc [1]. They are used to measure physical quantities and to monitor faults. When a change occurs in one of the values read by a sensor, actuators take action in order to control the system.

The need of quality, reliability, and safety in production processes has stimulated researches in the area of monitoring and fault diagnosis [2]. The measuring system based on sensors is of great importance because it provides data for manual and automatic operation. However, for the process to take place without control problems, it is necessary to validate the information received from the monitoring agents. This increases security and system's availability.

Failures in sensors can impact a system's performance. The effect of these faults depends on the stage where they are discovered and include: decreased availability; economic losses;

physical impacts to people and facilities. Aiming to mitigate these risks, it is necessary to implement control systems reliable and tolerant to faults.

In order to minimize problems of failures in sensors, monitoring systems use redundancy. Traditionally, this redundancy is created physically by using two or more sensors for reading a parameter [2,3]. Although this technique allows the identification of a flawed agent, it can only be applied in systems where there's space for the installation of redundant sensors [2,3]. In addition, installing multiple monitoring agents increases project costs [2].

An alternative to physical redundancy is the analytical redundancy (also known as software redundancy). It predicts the value of a signal through a model of the system. The model can be built from the mathematical equations that describe the real phenomenon or from the data. The predictions are then compared with the actual values of the system sensors [2,4]. The advantages of the analytical redundancy as well as advances in computing increased the number of monitoring systems based on this new technique [2].

The procedures based on analytical redundancy are classified in two groups: those that are based in the mathematical model and those who are obtained by procedures of artificial intelligence (AI). One of the most popular AI techniques is the one based on Artificial Neural Networks (ANN) [2,3]. When creating a model to predict values in classification and regression problems using ANN, usually it is taken into account only parameters as the number of inputs, outputs, and hidden layers. This leads to networks that are generally fully connected and have no improvements in its topology.

This work intends to use Ant Colony Optimization (ACO), an algorithm that has been used to solve NP-hard problems, to create a neural network with a tuned topology. The artificial ants will be used to find the best connections between neurons on different layers so that the resulting model can perform better. The ACO search algorithm, which we named ACONN, will use Back Error Propagation (BP) to optimize the neural network topology by suggesting the best neuron connections. The result ANN was applied to predict the value of variables of the IEA-R1 research reactor at IPEN.

2. ANT COLONY OPTIMIZATION

In the early 1990s it was developed a method of resolution of complex combinatorial problems based on "artificial ants". Since then, researchers have used this technique to find solutions to problems classified as NP-hard [5]. Because of its inspiration from real ants, the algorithm was named Ant Colony Optimization (ACO) and has been intensively used in solving problems such as sequencing tasks (scheduling) [5], routing [6,7], and optimization [8].

The ACO algorithm is based on the ability of ants to find, through the use of pheromone, an optimized way from their nest to a food source [6,9]. Pheromone is a chemical used by ants to mark paths so that the paths with the highest concentration of this substance have a greater chance of being followed [6,7].

One of the applications of ACO heuristic is to solve problems involving minimization on graphs [9,10]. Its first use was proposed by Marco Dorigo in the Traveling Salesman Problem (TSP) [11,12], which used artificial ants to find solutions to the TSP through the technique

known as positive reinforcement [5]. This technique is based on the analogy with the behavior of some species of ants that lay pheromone on the paths to the food source, thus enhancing the most followed paths (which can be optimal) [1].

The algorithms based on the ACO are created with artificial ants, which are probabilistic procedures based on artificial pheromone and heuristic [13]. Pheromones are represented numerically and are modified at each iteration, reflecting the search process [12,13]. The first search technique based on this algorithm and which was used in solving the TSP was called Ant System (AS) [12,14]. The AS uses a graph representation where each edge has a measure of pheromone, updated at runtime by artificial ants [10]. This graph consists normally of two numerical information: a fixed (established in the problem definition) and a variable [5,12]. These two pieces of information are independent of each other and are related to the connection between the j and k nodes of the graph. Distance between nodes and time to perform operations are examples of fixed information [5]. At each iteration, the ants add components to obtain the problem's solution.

In the AS heuristic, each complete path between the beginning and the end of the graph is offered as a possible solution to the problem. Spreading pheromone on the edges in an amount proportional to the quality of the solution (relative to the solutions obtained by others), the ants choose the best paths. Through probabilistic data, the insects choose the next node of the graph to be visited based on heuristic obtained by the distance between the nodes and the trail pheromone [10]. In order to not getting stuck into a local minimum solution, the algorithm applies the pheromone evaporation technique, which makes this marker element to dissipate over time [9,15].

In Figure 1 two paths are shown connecting an ants' nest to a food source. Pheromone levels (lengths of arrows) are balanced at the beginning of the process (Fig. 1a). Thus, the probability of selecting each path is equal, so that an equivalent number of ants pass the two sections. However, the insects that choose the largest path take a longer time to return, so that there is greater evaporation of the chemical. Then, the ratio of the amount of pheromone on the longer path decreases relative to that on the shortest path (Fig. 1b). After a few iterations, the shortest path stands (Fig. 1c). [1]

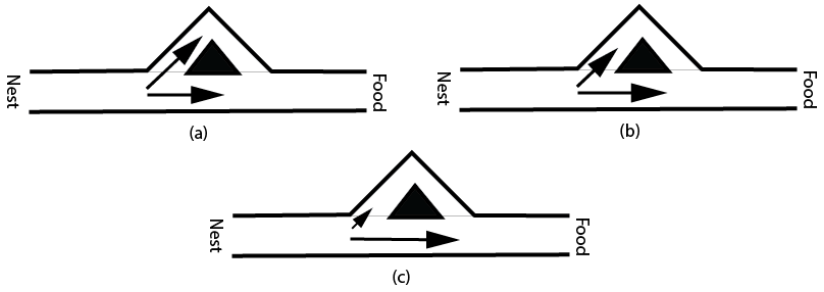

Figure 1 - Pheromone evolution. The length of the arrows is proportional to the probability of the route to be chosen.

In the AS algorithm proposed by Dorigo [11], an ant k has a memory which stores the nodes that were visited and, when standing on the node r of the graph, chooses the next node u to visit with a probability given by the equation (1)

$$s = \begin{cases} \underset{u \notin M_k}{argmax} \{ [\tau (r,u)]^{\alpha} . [\eta (r,u)]^{\beta} \} & if\ q \leq q_0 \\ S\ otherwise \end{cases} \quad (1)$$

In equation (1) $\tau\ (r, u)$ is the amount of pheromone in the edge that goes from node r to node u; $\eta\ (r, u)$ is the heuristic function (inverse of the distance between r and u); α and β are parameters that represent the weights given to the global intelligence and heuristic function respectively; q is a randomly chosen value with uniform probability in the interval [0,1]; $q0$ is a parameter ($0 \leq q0 \leq 1$); S is a random variable selected according to the probability distribution described in equation (2) [11]:

$$p_k(r,s) = \frac{[\tau (r,s)]^{\alpha} . [\eta (r,s)]^{\beta}}{\sum_{u \notin M_k} [\tau (r,u)]^{\alpha} . [\eta (r,u)]^{\beta}} \quad (2)$$

In equation (2) $p_k(r,s)$ is probability of an k choosing to follow the edge that goes from node r o node s; M_k is the set of nodes yet not visited by ant k.

Each time an edge is selected by an ant, its pheromone is updated through a process called local update. The purpose of this update is to prevent an edge with a large amount of pheromone to be chosen by all ants [11]. The local modification suggested by Dorigo is:

$$\tau (r,s) \leftarrow (1 - \rho) . \tau (r,s) + \rho . \tau_0 \quad (3)$$

where τ_0 and ρ are parameters chosen by the user.

In the AS algorithm, in addition to the local update, there is also the global update of pheromone whose goal is to highlight the edges that are part of the shortest paths. When all of the ants have find a solution to the problem, the one that found the best path deposits pheromone on the edges that compose its path. [11]. The amount of chemical deposited is inversely proportional to the distance of the solution. The global update is proposed in the AS is:

$$\tau (r,s) \leftarrow (1 - \rho) . \tau (r,s) + \rho . \Delta\tau(r,s) \quad (4)$$

In (4) $\Delta\tau(r, s)$ is defined as $1/Lk$ and Lk is the distance of the shortest path found in the current iteration.

The global update shows that the ants deposit on each edge of its solution a quantity of pheromone inversely proportional to the distance, that is, the shorter the path, the greater the amount of pheromone deposited on the edges. This update is similar to the technique known as learning by reinforcement, in which the best solutions are given priority.

There are several advantages to using the ACO meta-heuristics. One is the fact that it is a self-adaptive paradigm and is able to run both local and global search in a large and dynamic space [15]. Furthermore, because of its parallelizable background, this technique can be programmed in graphics processing units (GPU) [9,10]. Another positive aspect of optimization by ant colony is its good scalability and the low need of global information about graph's status. [15]

3. ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANN) are inspired by the biological neural network and are designed to solve problems in the areas like decision-making, categorization, function approximation, optimization, prediction and control. They can be seen as parallel and distributed processing systems, consisting of a large number of simple, interconnected processors [16]. The kind of ANN known as feed-forward neural networks (FFNN) is one of the most widely applied techniques in pattern classification. Usually FFNN are built in a three-layer topology: input, hidden and output layers. Usually every neuron in each layer is connected to all neurons on the next layer [17].

In a FFNN every neuron i computes a weighted sum if its r input signals and generates an output o_i:

$$net_i = \sum_{i=1}^{r} w_{ij} o_j + b_i\,,\ o_i = f(net_i) = \frac{1}{1+ e^{-net_i}} \tag{5}$$

where w_{ij} is the synapse weight associated with the connection between neuron j and neuron i, o_j is the output of a neuron in the previous layer, b_i represents a neuron's self-bias and f is an activation function (shown in the equation as the sigmoid function) [17][16].

A FFNN with n inputs and m output neurons can be trained based on a set of examples τ. Each training example p is applied to the input layer and the signal propagates through the hidden layers until it reaches the output layer. Then, the network's output (denoted y') is compared to the output defined on the training example (denoted y) to determine the network's error (denoted Ep) [17]. The most widely used error function is the sum of squared error: $E_p = \frac{1}{2}\sum_{i=1}^{m}(y - y')^2$ [17,18].

Training a fixed-topology neural network can be handled as multi-dimensional function minimization problem. This is due to the fact that, although the network error depends on the training set, topology, weights, and biases, if we keep the first two as fixed, the error function will depend only on the weights and biases. So, training the neural network becomes the problem of minimizing a mathematical function [17].

One of the most widely-used neural network training algorithm is the gradient descent based Backward Error Propagation (BP) algorithm. When using this technique, the derivative of the error vector (E) is computed with respect to each component of the vector w (the weights vector). This vector derivative is called *gradient* of E with respect to w [19].

Because the gradient specifies the direction of the steepest increase of E, the training rule for the gradient descent is:

$$w \leftarrow w + \Delta w \quad (6)$$

where

$$\Delta w = -\eta \nabla E(w)$$

In equation (6) η is a positive number called the learning rate and determines the step size in the gradient descent search [19].

4. IEA-R1 RESEARCH REACTOR

IEA-R1 is a nuclear research reactor at IPEN. It is a pool-type reactor and uses water in its cooling and moderation systems. Built by the company "Babcok & Wilcox", it uses graphite and beryllium as neutron reflectors. Its first criticality happened in September 16th, 1957, when its operation power was set as 2MW [2]. An improvement was made in 1997, increasing its power to 5 MW. Figure 1 depicts a diagram of IEA-R1 research reactor.

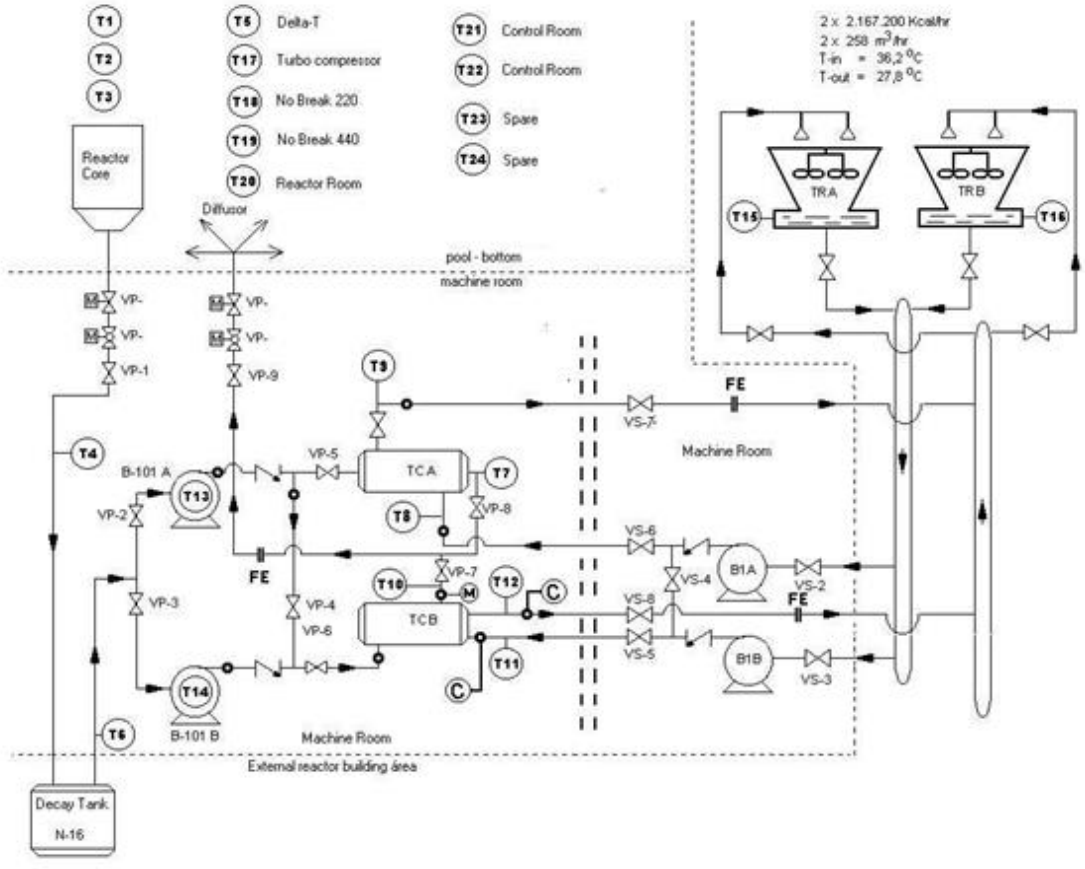

Figure 2 - Schematic diagram of IEA-R1 research reactor at IPEN

4.1 IEA-R1 Data Acquisition System (DAS)

The IEA-R1 reactor has a system that monitors 58 operational variables. This system is called Data Acquisition System (DAS) and the variables that it supervises include temperature, flow, level, pressure, nuclear radiation, nuclear power and rod position (Table 1). The DAS keeps track of the temporal history of all variables and does not interfere with the reactor control [2].

Table 1. IEA-R1 DAS variables.

Z1	Control rod position [0 a 1000 mm]
Z2-Z4	Safety rod position 1, 2 and 3[0 a 999 mm]
N2-N4	% power (safety channel 1, 2 and 3) [%]
N5	Logarithm Power (log channel) [%]
N6-N8	% power [%]
F1M3	Primary loop flowrate [gpm]
F2M3	Secondary loop flowrate [gpm]
C1-C2	Pool water conductivity [μmho]
L1	Pool water level [%]
R1M3-R14M3	Nuclear dose rate [mR/h]
T1-T3	Pool water temperature [º C]
T4 and T6	Decay tank inlet and outlet temperature [º C]
T5	(T4-T3) [º C]
T7	Primary loop outlet temperature (heat exchanger A) [º C]
T8-T9	Secondary loop inlet and outlet temperature (heat exchanger A) [º C]
T10	Primary loop outlet temperature (heat exchanger B) [º C]
T11-T12	Secondary loop inlet and outlet temperature (heat exchanger B) [º C]
T13-T14	Housing pump B101-A and B102-A temperature [º C]
T15-T16	Cooling tower A and B temperature [º C]
T17	Housing turbo compressor temperature [º C]
T18-T19	NO-BREAK temperature –220V and 440V [º C]
T20-T24	Room temperature [º C]

5. ACONN ALGORITHM

Many neural networks are built by using one input layer, at least one hidden layer, and one output layer. In this design, each layer has full connectivity with the subsequent layer. This work uses an ant colony algorithm to search on a graph built with the neurons of a three-layer network. The search algorithm suggests the best neuron connections, including links from input to output neurons. This creates network topologies with arbitrary connections.

The first step in our proposed ACO algorithm is to construct the graph that contains the solution components. Each ant will build a candidate solution by exploring the search space and suggesting a network topology with the selected connections between the neurons. Three types of connections will be allowed: connections between input and hidden neurons;

connections between hidden and output neurons; connections between input to output neurons. Each potential connection $c = i \rightarrow j$, connecting neurons i and j, is associated with two solution components: D_c^{true}, and D_c^{false}. These two components represent, respectively, the decision to include or not to include the connection in the current candidate network topology [17].

The number of input neurons (N_i) and output neurons (N_o) were determined based on the characteristics of our dataset. We decided to work with three-layer networks only and we set the number of neurons on the hidden layer (N_h) to be the sum of input and output neurons

$$N_h = N_i + N_o$$

The overall process of our ACONN is shown in Algorithm 1. Initially, 0.5 is assigned as the amount of pheromone to the solution components of each edge of the graph (line 3). This means that, for each connection, the probability of including it in the topology is equal to the probability of not including it [17]. Inside the inner loop (lines 6-12), each ant creates a candidate solution NN_i (line 7). Then, in line 8, the quality of the solution is calculated. In line 13, the pheromone trail is updated based on the quality of $NN_{thebest}$ (the best topology suggested during the current iteration). Next, the iteration best solution is compared with the best-so-far solution (lines 14-16), keeping the best solution created during the algorithm execution [17].

These steps are repeated until the same solution is generated for an amount of consecutive times, defined on parameter *conv_iterations* or until a maximum number of iterations is reached (line 18) [17]. In our experiments, *max_iterations* was set to 200, and *colony_size* (line 6) was set to 10.

In line 19, the best-so-far topology is used to train (using standard Backward Error Propagation) a final neural network to be returned. At this step, using the connections suggested by the ants, the weights and biases of the neural network are learned. The learning rate and momentum were both set to 0.01 and the number of epochs was set to 1000.

Algorithm 1. Pseudo-code of ACONN.

1: **Begin**
2: $NN_{best\text{-}so\text{-}far} = \varnothing$; $t = 1$;
3: initialize_pheromone();
4: **repeat**
5: $NN_{thebest} = \varnothing$; $Q_{thebest} = 0$
6: **for** i = 1 $\rightarrow$ colony_size do
7: $NN_i = ant_i.create\_solution();$
8: $Q_i = EvaluateQuality(NN_i);$
9: **if** $Q_i > Q_{thebest}$ **then**
10: $NN_{thebest} = NN_i$; $Q_{thebest} = Q_i$;
11: **end if**
12: **end for**
13: update_pheromone();
14: **if** $Q_{thebest} > Q_{bsf}$ **then**
15: $NN_{best\text{-}so\text{-}far} = NN_{thebest}$; $Q_{best\text{-}so\text{-}far} = Q_{thebest}$;
16: **end if**

17: $t = t + 1$
18: **until** $t = max\_iterations$ or *Convergence*(conv_iterations);
19: $NN_{final} = post\_processing(\ NN_{best\text{-}so\text{-}far}\ )$;
20: **return** NN_{final};
21: **End**

The process of creating of a candidate solution (line 7) starts with an edge-less graph whose connections will be chosen during the procedure. For each connection in the available set of connections, the ant decides whether to include it the candidate topology or not. This is done by either selecting D_c^{true} or D_c^{false}, based on the following probabilistic state transition equation [17]:

$$p(D_c^a) = \frac{\tau(D_c^a)}{\tau(D_c^{true}) + \tau\left(D_c^{false}\right)} \qquad (7)$$

where $p(D_c^a)$ is the probability of selecting decision D^a for connection c, and $\tau(D_c^a)$ is the current amount of pheromone associated with the component D_c^a (where $a = true$ or $a = false$).

After an ant finds a solution, it calculates its quality by training a neural network using Backward Error Propagation (line 8) with some optimized parameter values (the amount of training epochs was set to only 10). In order to avoid overfitting, the training set is split into two parts: the learning set, containing 80% of the training set; the validation set, containing 20% of the training set. The validation set is also used to calculate the quality Q_i of a candidate solution. The quality is measured by using the *correlation coefficient*, a statistical method for evaluating numeric predictions. The correlation coefficient measures the correlation between the predicted values on the instances and their actual values. It ranges from 1 for perfectly correlated results, through 0 when there is no correlation, to -1 when the results are perfectly correlated negatively [18].

3. RESULTS

We evaluated the performance of the ACONN algorithm by using a dataset with 7500 instances. This dataset contains values of the IEA-R1 variables *N2*, *N3*, *N4* and *T1*. A neural network with 3 inputs, 6 hidden neurons, and 1 output neuron was created to predict the value of *T1* based on the other three variables (Figure 3a). The ACONN algorithm was used to tune its topology and its performance was compared with an equivalent fully connected neural network. Our tests we executed with the following parameters:

- *max_iterations*: 500
- *conv_iterations*: 10
- *colony_size*: 10

The fully connected topology is shown in Figure 3a. The best connections, suggested by the ant colony algorithm, are displayed in Figure 3b.

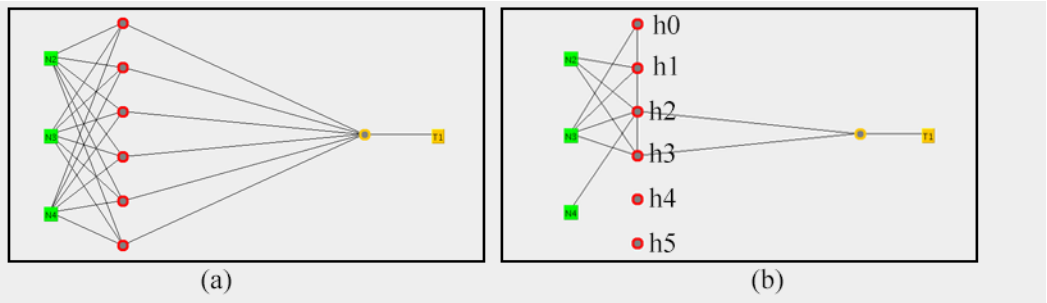

Figure 3: Topology suggested by the ACONN algorithm to predict T1 based on N2-N4

In Figure 3b, neuron h0 is connected to h3, and neuron h1 is connected to h2.

The predictions of the data used for validating the fully connected network are shown in Figure 4.

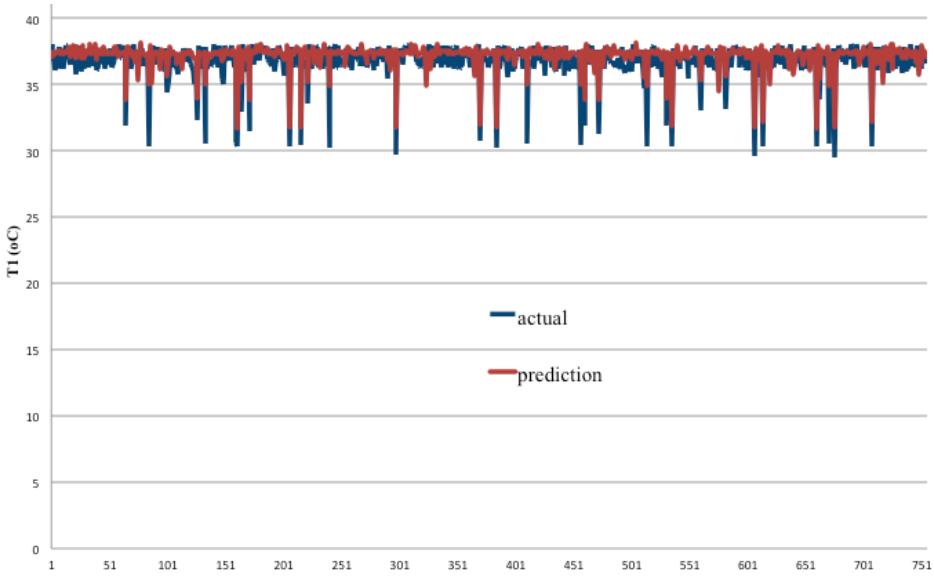

Figure 4 - T1 predictions made by the fully connected neural network

Figure 5 presents the predictions made for the validation data of the tuned neural network.

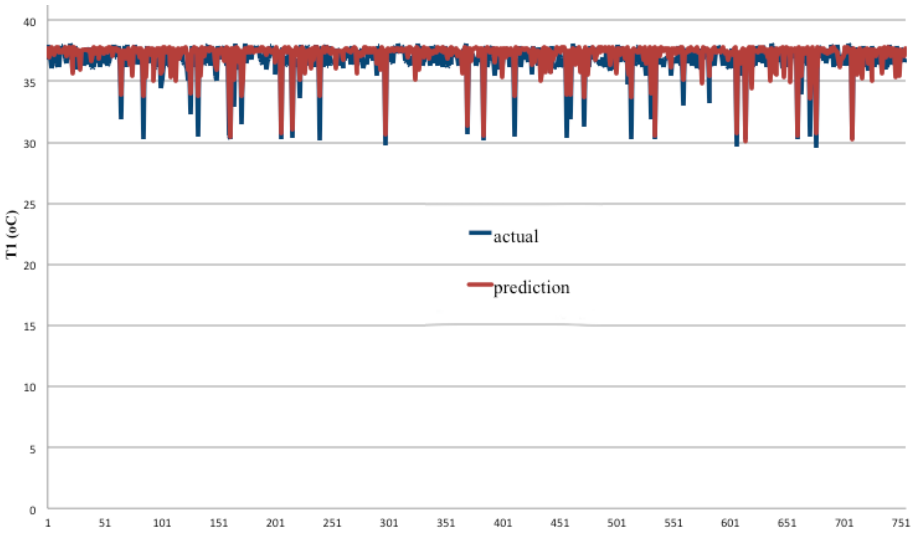

Figure 5 - T1 predictions made by the tuned neural network

The models that predict the T1 variable were designed by training two neural networks: one with full connections (Figure 3a); one with only the connections suggested by the ACONN algorithm (Figure 3b). In order to measure their performance we calculated the correlation coefficient using tenfold cross-validation. The performance comparison is summarized in Table 3.

Table 3: performance of tuned and fully connected neural networks

	Fully connected	ACONN
Correlation coefficient	0.4845	0.7982
Mean absolute error	0.8066	0.6452
Root mean squared error	1.2378	0.8257

4. CONCLUSIONS

Our results indicate that the performance of a feed-forward neural network can be improved by choosing the best connections between the neurons instead of using a fully connected topology. In future work we would like to improve the ACONN algorithm to include connections between neurons in the hidden layer. The methodology developed will be used in future studies to determine the values of other IEA-R1 variables. Also we will construct neural networks with different input variables. Furthermore, we would like to improve the quality evaluation function in order to find better solutions.

ACKNOWLEDGMENTS

The authors would like to thank the project CAPES / ELETRONUCLEAR “Avaliação de Instalações Nucleares: física de reatores, termo-hidráulica, monitoração e diagnóstico, segurança e análise de acidentes e engenharia de fatores humanos”.

REFERENCES

1. ARENY, Ramón Pallás. Sensores y acondicionadores de señal: prácticas. Vol. 2. Marcombo, Barcelona, 2004.
2. BUENO, Elaine Inácio; GONÇALVES, Iraci Martinez Pereira. "Estudo comparativo entre GMDH e redes neurais aplicados na monitoração de sensores." Revista do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (2010): 182
3. GARCÍA, Efraín Alcorta. Detectando fallas mediante redundancia analítica. Ingenierías, vol.4, pags. 43-48.
4. CHOW, E., and A. Willsky. "Analytical redundancy and the design of robust failure detection systems." Automatic Control, IEEE Transactions on 29.7 (1984): 603-614
5. TAVARES NETO, Roberto Fernandes; GODINHO FILHO, Moacir. Otimização por colônia de formigas para o problema de sequenciamento de tarefas em uma única máquina com terceirização permitida. Gest. Prod., São Carlos , v. 20, n. 1, Mar. 2013 . Disponível em <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-530X2013000100006&lng=en&nrm=iso>. Acesso em 25 Out. 2013. http://dx.doi.org/10.1590/S0104-530X2013000100006.
6. BALUZ, Rodrigo Augusto Rocha Souza. “Uma aplicação de sistemas inteligentes híbridos ACO-Fuzzy para a otimização do desempenho em redes de sensores sem fio”, Tese (Mestrado), Universidade de Fortaleza, 90p., Fortaleza, 2013.
7. BORGES, Marcelo Eduardo. "Insetos Sociais como sistemas complexos", Tese (Mestrado), Universidade Federal do Paraná, Curitiba, 2012.
8. SILVA, R. M. A., and GL RAMALHO. Otimização Baseada em Colônia de Formigas Aplicada ao Problema da Cobertura de Conjuntos. Diss. Tese de Doutorado apresentada ao CIn-UFPE, 2003
9. ANGELO, Jaqueline S.; Augusto, Douglas A.; Barbosa, Helio J. C. Ant Colony Optimization - Techniques and Applications. Rijeka: InTech, 2013
10. CECILIA, José M., et al. "Enhancing data parallelism for ant colony optimization on gpus." Journal of Parallel and Distributed Computing 73.1 (2013): 42-51.
11. DORIGO, M.; COLORNI, A.; MANIEZZO, V. “Positive feedback as a search-strategy”. Milão, Itália, 1991.
12. CASTILLO, Oscar et al. Dynamic Fuzzy Logic Parameter Tuning for ACO and Its Application in the Fuzzy Logic Control of an Autonomous Mobile Robot. Int J Adv Robotic Sy, v. 10, n. 51, 2013.
13. STÜTZLE, Thomas et al. Parameter adaptation in ant colony optimization. In: Autonomous Search. Springer Berlin Heidelberg, 2012. p. 191-215
14. SWAMINATHAN, Santhosh. "Rule induction using ant colony optimization for mixed variable attributes." Tese (Mestrado), Texas Tech University, Lubbock, 2006.
15. VENDRAMIN, Ana Cristina Barreiras Kochem. Cultural GrAnt: um protocolo de roteamento baseado em inteligência coletiva para redes tolerantes a atrasos. 2012. 195 f.

Tese (Doutorado em Engenharia Elétrica e Informática Industrial) – Universidade Tecnológica Federal do Paraná, Curitiba, 2012.

16. JAIN, Anil K. Artificial Neural Networks: A Tutorial
17. SALAMA, Khalid, and Ashraf M. Abdelbar. "A novel ant colony algorithm for building neural network topologies." Swarm Intelligence. Springer International Publishing, 2014. 1-12.
18. WITTEN, Ian H., and Eibe Frank. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2005.
19. MITCHEL, Tom M. Machine learning. McGraw-Hill, 1997.