

# An automatic selection of optimal recurrent neural network architecture for processes dynamics modelling purposes

Krzysztof Laddach<sup>a</sup>, Rafał Łangowski<sup>a,\*</sup>, Tomasz A. Rutkowski<sup>a</sup> and Bartosz Puchalski<sup>a</sup>

<sup>a</sup>*Department of Intelligent Control and Decision Support Systems, Gdańsk University of Technology, G. Narutowicza 11/12, 80-233 Gdańsk, Poland*

---

## ARTICLE INFO

### Keywords:

black-box model  
evolutionary algorithm  
neural modelling  
neural network architecture search  
pressurised water reactor

---

## ABSTRACT

A problem related to the development of algorithms designed to find the structure of artificial neural network used for behavioural (black-box) modelling of selected dynamic processes has been addressed in this paper. The research has included four original proposals of algorithms dedicated to neural network architecture search. Algorithms have been based on well-known optimisation techniques such as evolutionary algorithms and gradient descent methods. In the presented research an artificial neural network of recurrent type has been used, whose architecture has been selected in an optimised way based on the above-mentioned algorithms. The optimality has been understood as achieving a trade-off between the size of the neural network and its accuracy in capturing the response of the mathematical model under which it has been learnt. During the optimisation, original specialised evolutionary operators have been proposed. The research involved an extended validation study based on data generated from a mathematical model of the fast processes occurring in a pressurised water nuclear reactor.

---

## 1. Introduction


Nowadays, the use of advanced algorithms involved in the operation of widely understood industrial plants is very strongly related to the availability of accurate mathematical models of processes that occur in these plants. It can distinguish algorithms that perform the tasks of monitoring, diagnostics, estimation, or advanced control, etc. Therefore, the quality of performance of the above-mentioned algorithms tasks is closely related to the quality of mathematical models, but also to their time availability or the possibility of performing many simulations of the process with the required time regimes, as it is the case, for example, in a widely applied model predictive algorithm [60].

Typically, a mathematical model of a given process (system) can be devised in the phenomenological or experimental (behavioural) way [49]. In the first case resulting model is called a white-box model and is based on the conservation laws (physics, chemistry, biology, etc.), and in general, describes the phenomena occurring in a given system. Hence, the white-box model is derived analytically, and its abstract mathematical structure is fundamentally related to the physical structure of processes, and the model parameters have a physical meaning and interpretation. On the other hand, a mathematical model developed experimentally (behaviourally) is referred to as a black-box model and it is built based on observation of the behaviour of a given system. Thus, this model is derived experimentally, and its structure does not have to be essentially related to the structure of the process, and the model parameters do not have a physical interpretation. Naturally, each of these types of models has both advantages and disadvantages. However, because the complexity of some plants is significant as well as the phenomena occurring in them are sophisticated, the white-box modelling might become difficult, time-consuming, expensive, and in the worst-case impossible to perform. Moreover, in many cases, the values of white-box model parameters are not exactly known. Therefore, developing a black-box model or a certain kind of hybrid of white- and black-box models so-called grey-box model may be more justified and reasonable. In this paper, black-box modelling is considered.

The black-box model of a given system may be provided using various types of tools. They are commonly based on either a statistical analysis of time series or computational (artificial) intelligence. The first group includes the input-output models such as, e.g., linear autoregressive moving-average model with exogenous inputs (ARMAX) or non-linear autoregressive moving average model with exogenous inputs (NARMAX) model structure [10, 13]. The

---

\*Corresponding author

 [rafal.langowski@pg.edu.pl](mailto:rafal.langowski@pg.edu.pl) (R. Łangowski)

ORCID(s): 0000-0001-9122-2167 (K. Laddach); 0000-0003-1150-9753 (R. Łangowski); 0000-0001-8818-6126 (T.A. Rutkowski); 0000-0001-9834-6250 (B. Puchalski)

ARMAX and NARMAX models, approximate the input-output system behaviour by the linear or non-linear difference equations defined in the finite-dimensional linear or non-linear discrete-time domain, respectively. In general, the process of identifying a black-box model involves determining the structure of the unknown linear or non-linear difference equation, estimating its parameters, and finally checking or validating the resulting model to ensure that it describes the modelled system accurately. A wide class of dynamical systems can be approximated with sufficient accuracy only by the linear expressions involving the variables which characterise the system. However, there are many practical cases when a linear description of a process is not sufficient, and a global, more accurate non-linear model is required. Additionally, the determination of the structure of non-linear functional dependencies between inputs and outputs of the modelled system is not an obvious and trivial task. Typically, for the group of the linear black-box models, the polynomial model structures are used. For the non-linear models, to overcome mentioned limitations, the common practice approximates the unknown high-dimensional and non-linear functional dependencies by using well-known and well-suited for those purposes methodologies using polynomials, wavelets, neural networks, or hybrid neural-fuzzy estimators. With those technologies, the high-dimensional and non-linear function is approximated by a set of appropriately organised lower-dimensional functions. A large group of those technologies is classified as the black-box models inspired by artificial intelligence [14]. In this paper, a methodology based on artificial neural networks (ANNs) and their recurrent implementation, which are well-known to be universal approximators of non-linear dynamic systems, will be further considered [17, 27, 35, 40, 48]. In recent years, many interesting applications of ANN have been found in the literature, include classification tasks [19, 50, 51], image (pattern) recognition [1, 8, 34], smell recognition [12, 36], speech recognition [18], text generation [58], prediction purposes [3, 29, 59], modelling and control of dynamic systems [40, 42, 48, 66], state estimation, generating control signals and operating as diagnostic systems [24, 56, 57], fractional order operators approximations [45], and many others, e.g., [22, 43].

A common feature of practically all ANNs applications is the need to select their architecture (optimal structure/topology) so that their performance, in terms of accuracy and time complexity, will be as high as is possible and satisfactory for the user. In particular, attributes/components of ANN such as neuron model, the connections between neurons, the number of layers, the number of neurons in each layer, level of delays, and net input and activation functions should be determined. In a classic approach, manual selection of the optimal structure of ANN manually is not a trivial task - the exponential dependence of the number of parameters that the user should optimally set to the complexity of the ANN topology [26, 35]. At the next stage, it is still necessary to train the chosen ANN structure, with the following considerations: the problem of initialisation of the weights, the type of learning algorithm, the lack of definition of the optimal number of iterations for the learning algorithms, the size of the training set and the value of the learning rate coefficient. Overall, the impact of the aforementioned architecture-related ANN attributes and its learning process on the ANNs performance is verified using a trial-and-error approach. It should also be noticed that, in the case of the over-sized and under-sized ANNs structure, the over-fitting and under-fitting problems (poor generalization, trap in local solution) can take place. In this area, the approach based on dividing a data set into training, validation, and testing subsets and further evaluation of networks using all subsets to minimise over-fitting may be useful [61].

In general, the problem of neural neural network architecture search (NAS) is still an open issue. It is because there are no general and certain rules in this topic. Hence, experimental methods based on heuristics, experience and intuition are often in favour by various authors. For example, the NAS based on the trial-and-error method and in-depth analysis of the obtained results has been discussed in [30]. However, all these methods require considerable user involvement, and they are time-consuming. Therefore, a fully or partially automatic selection of ANNs architecture is an attractive alternative. One of the approaches enabling such operations is to automate the trial-and-error technique using a computing environment with dedicated optimisation tools (solvers) that meet especially a multi-objective optimisation requirement [20] (e.g. minimal ANN topology, and maximal ANN accuracy). Examples of such solvers are nature-inspired optimisation algorithms, which are commonly used to solve optimisation tasks for a wide class of real-world complex problems. An interesting and comprehensive review of this type of algorithms from the groups of evolutionary algorithms (e.g. genetic algorithm, evolutionary strategy, differential evolution), swarm intelligence (e.g. ant colony and particle swarm optimisation) or those inspired by the laws of physics and chemistry (e.g. simulated annealing), which were used for the optimal, partial or full selection of the ANN structure, can be found in [25]. In addition to references to individual publications, an overview of articles from over the last two decades, the study indicates [25]: (i) the range of automatically selected, optimised ANN parameter/parameters divided into seven groups: architecture and weights, connection and weights, hidden neurons, hidden neurons and hidden layers, hidden layers, hidden neurons and connections weights and bias; (ii) used optimisation algorithm; (iii) and its observed strengths

and weaknesses. The other papers related to using artificial intelligence that uses evolutionary algorithms to generate ANN can be found in the literature under the neuroevolution topic [21, 39, 52, 53, 55, 63, 65]. A wide overview of different aspects of neuroevolution methods and discusses their potential for application in the field of deep learning may be found in [53]. One well-known algorithm from this group is NeuroEvolution of Augmenting Topologies (NEAT) presented almost two decades ago [54, 55]. The basic NeuroEvolution of Augmenting Topologies (NEAT) is a Topological and Weight Evolving Artificial Neural Networks (TWEANN) method that enables the learning of the structure of ANNs at the same time it optimises their connectivity weights. The NEAT method is based on direct encoding to encode the phenotypes (ANNs structure) in the genotype, specialised operators of crossover, mutation and speciation (the introduction of the concept of species – sub-populations), and other crucial issue related to the identification of similar neural network structures evolving independently in population and assigning each structure its historical markings (innovation number). They act as chronological indicators that facilitate crossover by identifying homologous sections between different neural networks. The innovation number of each gene is inherited by the offspring, facilitating the retaining of its historical origin throughout evolution. Generally, neural networks are grouped in appropriate sub-population based on their topological similarities expressed as compatibility distance. The sub-populations protects topological innovations in the neural network structure, and such individuals compete within their own niche instead of the entire population. The NEAT algorithm starts the evolution with minimal structure and, in further iterations, introduce new nodes and connections via mutations (evolve to more complex networks structures) as long as they find useful after fitness evaluation. In the paper, [41], comprehensive categorisation of the NEAT algorithms successors found in the literature is described in detail. Historically, the introduction of species has been also introduced in earlier studies, e.g., [37]. Another approach is based on genetic algorithms, where ANNs are used to observe the state of the nuclear reactor for diagnostic purposes [6, 7]. In this approach, the "importance" of particular neurons, and thus the probability of their survival, depends on the influence of the particular neuron on the output of the entire network - the close link between the ANNs structures and the problems for which they are addressed is shown.

In this paper, the authors' neuroevolution methods inspired by various methods described in the literature [5, 25, 41], especially evolutionary algorithms (EAs) to build a black-box model of a non-linear dynamic process are presented. The authors developed and simulation-verified four NAS algorithms. In general, the proposed algorithms using the  $\mu + \lambda$  evolutionary strategy. The first two using roulette-based selection and elitism mechanism. In comparison to the first, the second one includes a specialised neuron mutation-deleting operator. In turn, the third and fourth algorithms incorporate the species concept. In comparison to the third, the fourth algorithm is distinguished because the weights of connections between neurons are selected using a gradient descent learning method (the hybrid algorithm, where EA still selects ANN structure) and not jointly by EA (ANN structure and weights). For all the above-mentioned algorithms, specialised evolutionary operators have been developed, i.e. mutation and crossover for weights and neurons (for new neuron formation or neuron death), respectively. All algorithms use the direct encoding type to store the network structure. Additionally, proposed algorithms satisfy the postulate of automatic selection of the neural network structure to various degrees, ranging from the algorithm that selects two parameters of ANN (number of neurons in the hidden layer and connection weights) to the algorithm that selects six parameters of ANN (number of hidden layers, number of neurons in the hidden layer, input delay level, output delay level (feedback), connection weights and learning method). As an application the dynamics of fast processes in a pressurized water reactor (PWR) is taken into account. A nuclear reactor is a non-linear, spatial, and non-stationary plant that belongs to the elements of critical infrastructure. Its processes are characterised by multi-scale and complex dynamics. Because of these reasons, there are many different mathematical models of nuclear reactors, which are used depending on the purpose, e.g., synthesis of control algorithms, modelling of physical processes, diagnostics, on-line monitoring, power demand scheduling, or fuel campaign planning. In this study, a mathematical model of PWR reactor that has been originally developed for diagnostics and control purposes has been used [38]. The model consists of a sub-model responsible for the description of point neutron kinetics in the reactor core with six groups of delayed neutron precursors, a sub-model responsible for the description of thermohydraulic phenomena related to heat exchange between the core and the coolant, a module for calculating reactivity feedbacks from fuel and coolant temperatures, and a sub-model of the actuator, which mimics the operation of control rod drive mechanism. The model uses a thermal-hydraulic structure consisting of a single fuel node - F and two coolant nodes - C (1F/2C) which is described in [31, 38, 46, 47]. The model of the actuator used is given in [47], whereas the overall mathematical model equation structures together with the necessary parameters are given in [46, 47]. The mathematical model of the PWR presented in the aforementioned works takes into account only fast processes taking place in the reactor, i.e. neutron kinetics and heat exchange between the fuel and coolant.

From the research point of view addressed in the paper, the nuclear reactor model used is certainly a non-trivial and challenging case. It should be noted also that the model has been used only to generate learning, validation, and test data, and mainly for this reason a detailed description of it is not presented in the body of the paper as modelling of processes taking place in a nuclear reactor is not the subject of the presented research.

To summarise the main aim of this work is to develop and verify the authors' algorithms of optimal artificial neural network architecture search for black-box (behavioural) modelling purposes. As a type of ANN, the recurrent network has been chosen - as a universal approximator of a non-linear dynamic system. Its architecture is automatically selected by solving an appropriately defined optimisation task. The obtained neural network operates as a black-box model of the fast processes in a PWR. As an input to the black-box model the position of control rods is used whereas the scaled thermal power of a PWR is an model output. The optimality has been understood as achieving the desired trade-off between the size of obtained ANN and the accuracy of black-box model responses. Hence, the main contributions of this paper are as follows:

- four algorithms of NASs using either EA based itself or EA with gradient descent learning methods, which create the desired recurrent neural network models are devised and verified based on the selected case study (fast processes in a PWR),
- the specialised evolutionary operators, i.e. mutation and crossover for weight, neuron (for new neuron formation or neuron death) and delays for proposed NAS algorithms are delivered and their performance is verified in comparison to the basic NEAT algorithm and exhaustive search algorithm,
- the black-box model of the selected processes in a PWR is obtained with delivered by authors NASs algorithms and verified based on the black-box models delivered by the basic NEAT algorithm and exhaustive search algorithm.

The paper is organised as follows. The problem formulation is presented in section 2. The four various authors' algorithms of artificial neural network architecture search for black-box modelling purposes are delivered in section 3. The obtained single-input single-output model (SISO) of the fast processes in PWR and the performance of its operation verified in a simulation way are described in section 4. The paper is concluded in section 5.

## 2. Problem statement

In general, a concept of ANNs is based on an investigation of processes taking place in biological neural networks [9, 28, 64]. Typically, ANNs can be classified in the following way [4, 11, 26]:

- feed-forward artificial neural networks (FNNs) – where signals are transmitted only in one direction, i.e. from the network input to its outputs through the network layers;
- recurrent artificial neural networks (RNNs) – these networks are characterised by an internal state; in other words there are feedbacks in the RNNs from the outputs of individual neurons to their inputs, or from the network outputs to its inputs; this causes that a change in the state of individual neuron can be transferred through feedback to the other neurons, invoking transient states and generally leading to another state of the network; thus thanks to feedbacks RNNs have their internal state, which allows them to model dynamic plants;
- cell artificial neural networks (CeNNs) – where connections between particular neurons occur only in the closest neighbourhood; these connections are generally non-linear and described by a system of differential equations; this type of networks is mainly used for clustering of input data, and often the method of teaching them, in which the teacher does not exist (unsupervised learning), is based on the Hebb rule; an example of CeNNs is a Kohonen's map of features [4].

As it has been mentioned in section 1 as an application the fast processes occurring in PWR are considered. Similarly to the vast majority of real plants, also PWR has internal feedbacks. Thus, the natural choice of a type of ANN has been RNN but it has been decided to use only feedback from the network output to its input in this paper. Searching for a network with such architecture is equivalent to looking for a dynamic function meeting the discrete equation (1):

$$y(k) = f(y(k-1), y(k-2), y(k-3), \dots, y(k-n), u(k), u(k-1), u(k-2), u(k-3), \dots, u(k-m)), \quad (1)$$

where:  $y(\cdot)$  is the output at the discrete-time instant specified by  $(\cdot)$ ;  $u(\cdot)$  denotes the input at the discrete-time instant specified by  $(\cdot)$ ;  $f(\cdot)$  signifies the function specified by  $(\cdot)$ ;  $k, m, n$  are the discrete-time instants.

In turn, an architecture of exemplary RNN is presented in Fig. 1.

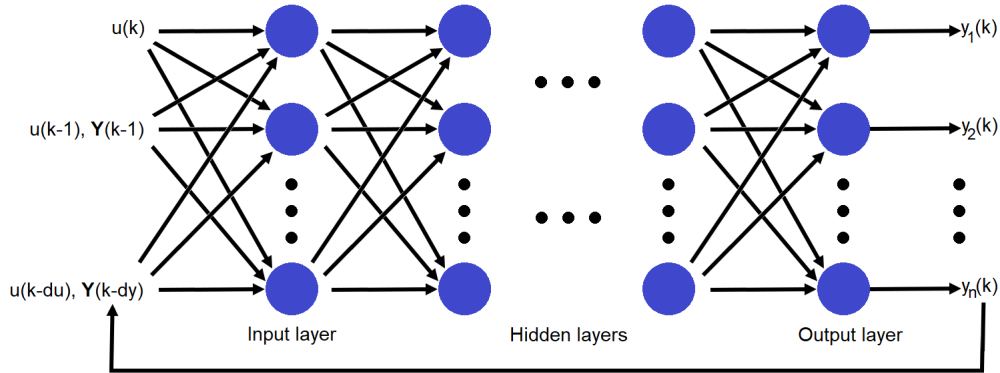


Figure 1: An architecture of exemplary RNN.

As it can be noticed the following layers are distinguished:

- an input layer, which is responsible for the normalisation of data (the most often it is re-scaling the input data to a given range),
- hidden layers, which are responsible for the signals processing, and,
- output layer, there are at least as many neurons in it as there are outputs from the network.

The particular symbols in Fig. 1 denote:

$Y(\cdot)$  – the vector of feedback outputs at the discrete-time instant specified by  $(\cdot)$ ;

$y_i(\cdot)$  – the outputs at the discrete-time instant specified by  $(\cdot)$ ,  $i = \overline{1, n}$ ;

$du$  – maximal level of input delay;

$dy$  – maximal level of feedback outputs delay.

The main task of the first layer (input layer) is re-scaling inputs data. However, in many cases, also in this work, the inputs data are already normalised, so the input layer is skipped, and it can be understood as transmitting input signals directly to the next (hidden) layers. Therefore, the developed algorithms enable selecting of the number of hidden layers and the number of neurons in these layers in a given RNN. Moreover, they also enable selecting of  $du$  and  $dy$  delays. In order to perform this task, it is necessary to define the neuron model. The simple perceptrons with the weighted sum, and with bipolar sigmoid in the hidden layers and linear activation function in the output layer are used. These activation functions are chosen primarily because of:

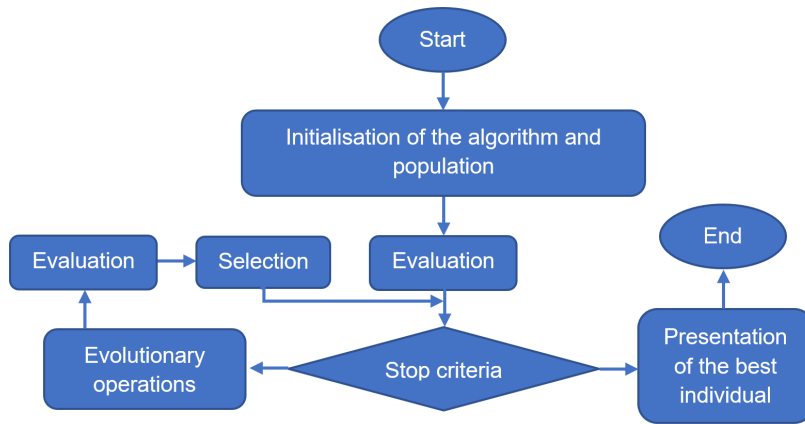
- the monotony of the derivative ensures the correct operation of gradient methods of ANN learning, which are used in one of the developed algorithms,
- the bipolarity of functions increases the acceptable field of searching for the optimal solution,
- since the task of neurons in the output layer is the linear transformation of the sum of the previous layer outputs the linear activation functions in this layer are selected.

In order to perform an automatic selection of the number of hidden layers, the number of neurons in these, the weights of neurons, and delays in a given RNN the EAs are used either independently or in combination with gradient-based methods for the network learning process. It is since an EA, i.e., enables solving non-trivial optimisation tasks [37, 53]. Undoubtedly searching for the optimal architecture of ANN belongs to a category of this type. It is because there are different types of variables involved in the task, i.e. real numbers, e.g., the values of ANN weights as well as integer and binary variables, e.g., the values of maximal levels of used delays and the number of layers and the number of neurons in each layer. A general scheme of an EA is presented in Fig. 2.



**Table 1**  
The summary of main features of proposed NAS algorithms.

Type of ANN architecture - RNN				
Component name	DNAS1	DNAS2	DNAS3	DNAS4
number of hidden layers	hyper-parameter	hyper-parameter	hyper-parameter	parameter
number of neurons in hidden layers	parameter	parameter	parameter	parameter
$du$	hyper-parameter	parameter	parameter	parameter
$dy$	hyper-parameter	parameter	parameter	parameter
connection weights	parameter	parameter	parameter	parameter
learning methods	hyper-parameter	hyper-parameter	hyper-parameter	parameter
number of neurons in output layer	hyper-parameter	hyper-parameter	hyper-parameter	hyper-parameter
model of neuron	hyper-parameter	hyper-parameter	hyper-parameter	hyper-parameter
net input function	hyper-parameter	hyper-parameter	hyper-parameter	hyper-parameter
activation function in hidden layers	hyper-parameter	hyper-parameter	hyper-parameter	hyper-parameter
activation function in output layer	hyper-parameter	hyper-parameter	hyper-parameter	hyper-parameter



**Figure 2:** A general scheme of an evolutionary algorithm.

The development of evolutionary operators has a crucial meaning for black-box modelling when RNNs are involved. It is well-known that, in general, two types of evolutionary operators can be distinguished, i.e. mutation and crossover [26, 33, 37, 55]. The mutation operator is primarily responsible for exploring the field of solutions whereas the crossover operator is responsible for exploiting this field. In the further part of this paper, the developed (dedicated) mutation and crossover operators are described in detail.

As it has been mentioned above, four NAS algorithms have been proposed using EAs. The first developed NAS algorithm (DNAS1) enables only the selection of the number of neurons in the hidden layer and the values of their weights. In contrast, the second developed NAS algorithm (DNAS2) and third developed NAS algorithm (DNAS3) allow the selection of the maximal levels of delays  $du$  and  $dy$  also, using an EA with dedicated evolution operators. In turn, in the fourth developed NAS algorithm (DNAS4) an EA is used for searching for the number of hidden layers, the number of neurons in these layers,  $du$  and  $dy$  and one of the three gradient-based RNNs learning methods. To summarise, the classification of the NAS algorithms proposed in the paper, from their main features point of view, is presented in table 1. Thus, each individual in the population represents a RNN architecture that is related to searched black-box model.

### 3. Designed NAS algorithms

During the developing of each NAS algorithm a certain set of input parameters has been determined. These parameters are listed in table 2.

**Table 2**

A set of input parameters for proposed NAS algorithms.

Input parameters	
Symbol	Description
$maxLay$	the maximal number of hidden layers
$maxNinLay$	the maximal initial number of neurons in each hidden layer
$du$	the maximal initial level of input delay
$dy$	the maximal initial level of feedback outputs delay
$popSize$	the initial number of the population individuals
$pCross$	the probability of crossover
$p_i$	the values of weights in a fitness function
$minDelta$	the minimal module of changing the weights in mutation operator
$maxDelta$	the maximal module of changing the weights in mutation operator
$pMutW$	the probability of weights mutation
$pMut$	the probability of mutation in DNAS4
$pMutNewN$	the probability of generation of a new neuron
$pMutD$	the probability of mutation of delays
$pMutDelN$	the probability of delete of neuron
$minW$	the minimal initial weights values
$maxW$	the maximal initial weights values
$hmBest$	the number of the best individuals subject to elitism
$pRetrain$	the probability of selecting an individual for retraining

### 3.1. DNAS1 – DNAS3 algorithms

In this section the DNAS1 – DNAS3 algorithms, which operate according to general scheme presented in Fig. 2 are described. Firstly the algorithms initialisation, fitness function evaluation, selection and dedicated evolutionary operators, i.e. mutation and crossover are presented. Next, the unique features of a given algorithm are discussed.

#### Initialisation

At the beginning of operation of the DNAS1 – DNAS3 algorithms each individual in an initial population is created in accordance with the following procedure. Firstly, the values of  $du$  and  $dy$  are randomly taken from the ranges  $(0, duMax)$  and  $(0, dyMax)$ , respectively or they are assumed to be fixed (see tables 2 and 3). It is worth adding that, the assumed large initial values of  $du$  and  $dy$ , on the one hand, can improve the mapping of target responses, but on the other hand, it directly degrades the individual fitness evaluation. Next, the number of neurons for the hidden layer is randomly selected from the range of  $(1, maxNinLay)$ . In turn, in the output layer, only one neuron occurs because the network has only one output – SISO model of PWR. Then, for each neuron, a random value for appropriate number of  $(N_{rw})$  is drawn within the range  $(minW, maxW)$ . The appropriate number of random weights is understood here as the number of inputs to each neuron that results from the network structure. For the first layer, it is:

$$N_{rw} = u_k + du + dy + b, \quad (2)$$

where:  $u_k = 1$  denotes current input sample;  $b = 1$  stands for the bias.

Whereas for the next layers, the number of needed weights equals the number of neurons in the previous layer increased by one, i.e. the weight corresponding to the bias. The value of the bias weight can be understood as a direct value of the bias.

#### Mutation

As it has been aforementioned, during operation of EAs mutation is mainly responsible for exploring the field of decision space. In the DNAS1 – DNAS3 algorithms, the mutation of individual weights are designed and programmed in the following way. The probability of mutation for each weight, as the value of one of the algorithm parameters, has been set to  $pMutW$ . Then a random decision is taken as to whether or not there will be a mutation of a given weight. The weight is changed by adding to it the corresponding value from the  $\Delta w$  vector. The values of  $\Delta w$  vector depends on the value of the fitness evaluation of the individual for whom the mutation occurred. The formula linking

the value of the fitness evaluation with the  $\Delta w$  has been developed so that the weights of better-adapted individuals change by smaller values. Assuming that  $fit$  is a vector of the values of the individual fitness evaluation, the procedure for calculating the  $\Delta w$  vector is as follows:

$$fit = -fit, \quad (3)$$

$$fit = fit + |min(fit)|, \quad (4)$$

$$minFit = min(fit), \quad (5)$$

$$maxFit = max(fit), \quad (6)$$

$$\Delta w = \frac{(maxDelta - minDelta)(fit - minFit)}{(maxFit - minFit)} + minDelta. \quad (7)$$

The vector  $fit$  is first inverted, i.e. multiplied by -1, in order to assign the highest value resulting from the fitness evaluation to the individual who is the least fit. Then it is ensured that all its elements are positive, and next, the maximum and minimum values from the  $fit$  vector are found. In the last line of the procedure, the  $\Delta w$  vector is calculated. Hence, it is a linear mapping of the value of the  $fit$  vector into a range of  $(minDelta, maxDelta)$ . The results of this procedure are that for the best-fit individual the module of change is  $minDelta$ , and for the least fit  $maxDelta$ . These values are the parameters of a given algorithm, just like the probability of weights mutation  $pMutW$  (see table 2). If a mutation occurred in one or more weights in a given individual, a new individual containing this new weight or new weights are added to the population of mutated individuals. A functional diagram of the weight mutation operator is shown in Fig. 3. The dashed blue box in Fig. 3 marks the part of the diagram that is shared with the following figures (Figs. 4 and 5). Thus, this part is not repeated on them.

It should be noticed that a feature of such evaluation of the weights adjustment is the fact that the adjustment for the most poorly fitted individuals is bigger. Moreover, this mechanism reduces the risk that an individual who is close to the optimum will jump over it because the change in weights will be smaller for him. Compared to conventional solutions, in which the weights changes values depending on the generation number, i.e. the longer the algorithm operates, the weights change by smaller value, the proposed approach eliminates the problem of marginal weight changes in the final stage of the algorithm operation [37]. Moreover, it allows the small value to change the more accurate exploitation of the most promising areas - the weights of the most-fitted individuals. The proposed solution also has an advantage over approach, where the value of the changes depends on the average value of the fitness function of individuals of the entire population [37]. It is because, it is not necessary to determine a level of satisfaction, i.e. the value of the fitness function at which the changes would reach the minimum level.



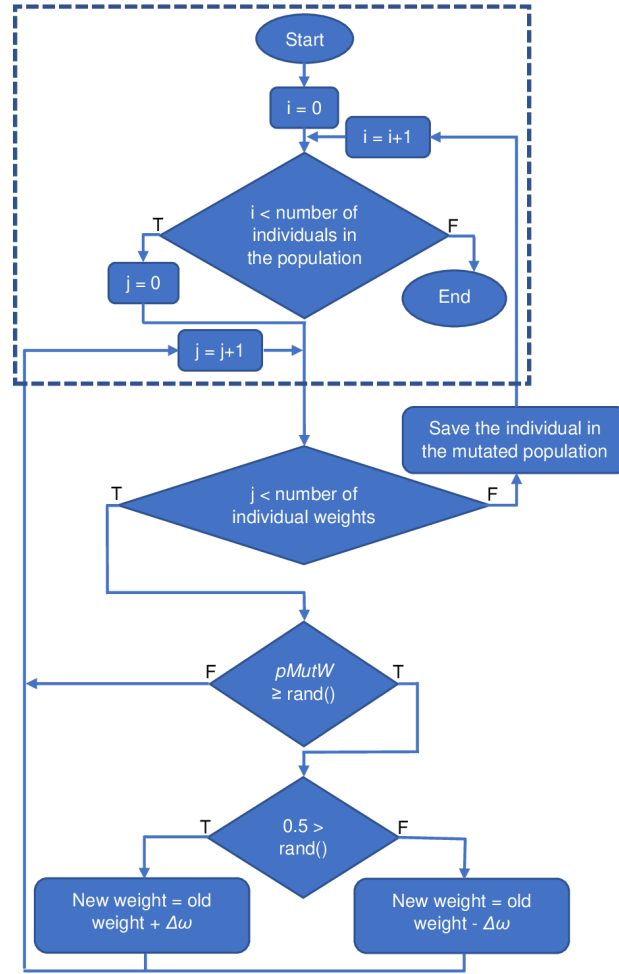


Figure 3: A functional diagram of the weight mutation operator.

It is well-known that every human being during its life is subject to a continuous learning process. It should be noticed that learning is not only improving what the brain already knows (improving network's weights – weights mutation), and what once had to be learned, but also learning completely new things. This process occurs due to the building of new connections between existing neurons and by creating new neurons that build new paths for nerve signals, thus allowing the network to extend its capability. This fact in neuro-science is known under the broad concept of neuroplasticity [11]. A similar feature may be required in the studied ANNs, which even after achieving optimal weights for a given structure, may not sufficiently map learning data. According to Kolmogorov's theorem [35], the solution to this problem is to increase the number of neurons in the network structure. The above ideas have been implemented in devised algorithms in the following way. For each individual in the population, the number of free places, where the neuron could occur, is calculated according to the formula:

$$N_{fp} = \max NinLay - N_{an}, \quad (8)$$

where:  $N_{fp}$  denotes the number of free places;  $N_{an}$  is an actual number of neurons in a given layer.

Then, for each free place with a probability equal to  $pMutNewN$ , a draw takes place to determine whether a new neuron is to be created in it. If the draw is successful a new neuron with random weights and bias is generated in the same way as it has been described in the initialisation part of this section. The newly formed neuron is always located in the first free place in a given layer. Of course, the creation of a new neuron requires modification of neurons' weights in the next layer. Hence, for each neuron in the next layer, a new random weight is added at the place corresponding to the connection with the newly formed neuron. Each addition of a neuron to an individual creates a new individual in the mutated population. The operation of the described procedure is presented in Fig. 4.

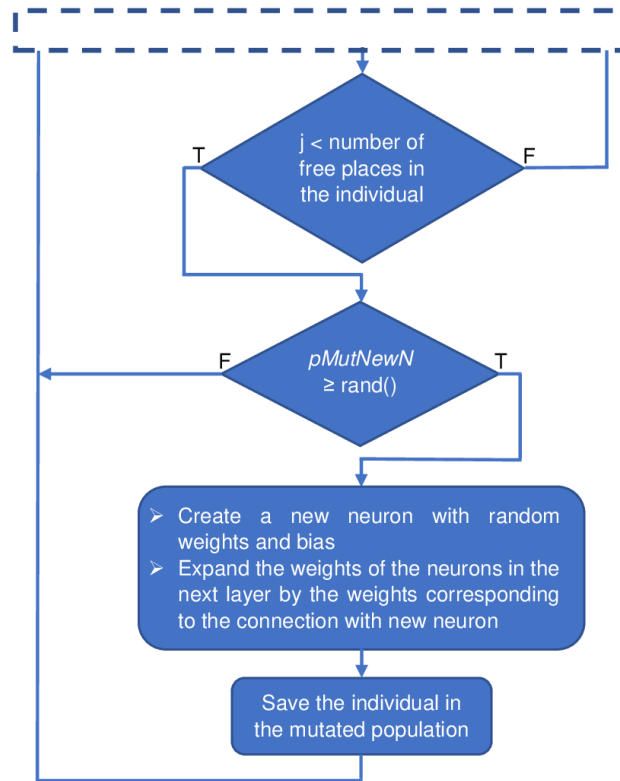


Figure 4: A functional diagram of the neuron mutation operator.

In opposition to the above, there is also the natural phenomenon of neuron death. This phenomenon may be compared to the creation of individuals with fewer neurons in a given layer of ANNs. In designed NAS algorithms, the possibility of neuron death is given to every existing neuron in the hidden layer for each individual in the population with probability equal to  $pMutDelN$ . Of course in the next layer, the weights corresponding to the removed neuron are also removed. This evolutionary operator works very similar to the procedure illustrated in Fig. 4.

The last part of the mutation operator in the developed NAS algorithms is the delays mutation. The mutation of delays consists of randomly increasing or decreasing the delay by one time instance. The fact of change of delay is drawn with a probability equal to  $pMutD$  for each type of delays -  $du$  and  $dy$  separately. In the case of decreasing the level of delay, the corresponding input with its weights is removed from the network. In turn, in the case of increasing the level of delay, the set of saved inputs is enlarged and the weights corresponding to this input are drawn from the range  $[minW, maxW]$ . The probability of increase and decrease is equal to 50% for each. Of course, each delay mutation generates a new individual, i.e. a new individual has at most one delay difference than the source individual. A functional diagram of the delays mutation operator is shown in Fig. 5.

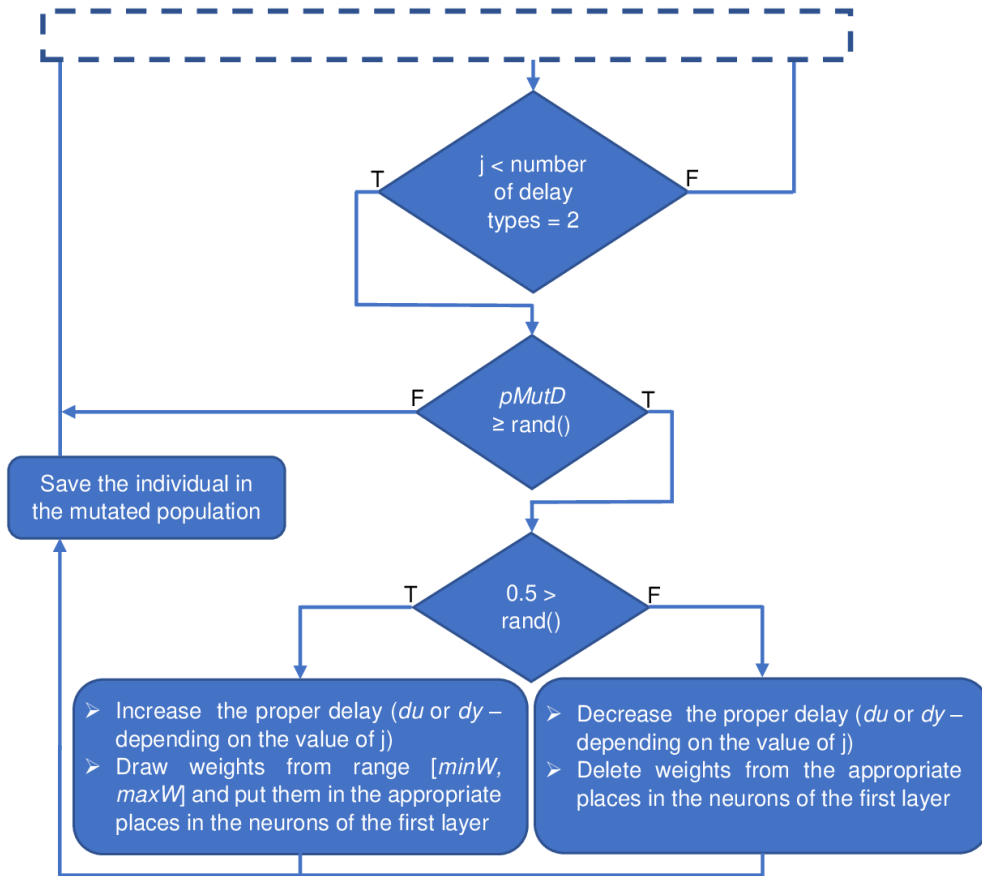


Figure 5: A functional diagram of the delays mutation operator.

**Crossover**

In general, in a short scale of time, an essential part of evolution is a result of crossover (recombination) primarily [37]. The main advantage of that method of reproduction is the diversity of offspring (children). Among others for this reason, as it has been aforementioned, during operation of EAs crossover operator is mainly responsible for exploiting the field of solutions. In the developed NAS algorithms, the crossover operator ensures the exchange of all network’s architecture features. A general diagram of the crossover is shown in Fig. 6.

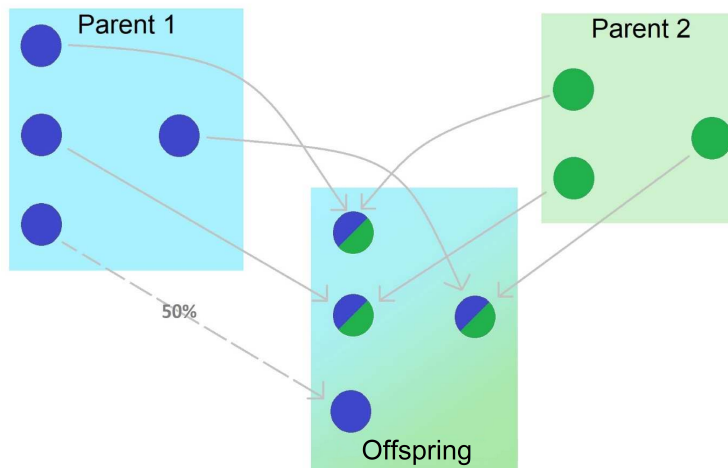


Figure 6: A general diagram of the crossover.

In the first step, parents with  $pCross$  probability are drawn from a given population. It is assumed that each individual of a given population can be selected on a parent only once in a given generation of the algorithm, and participate in creating only one offspring. Next, the pairs are drawn from the parents' group. Each neuron of RNN, which represents parent 1 is crossed with the corresponding neuron of RNN, which represents parent 2. However, parents in a pair may differ not only by weight values but also by other RNN structure components. This leads to a situation where the corresponding neuron from parent 1 neuron may not exist in the parent 2. This problem is resolved by introducing a 50% probability of copying to a child a neuron found only in one parent. The copied neuron always occupies the first free place after the last neuron in a given RNN layer (see Fig. 6). Hence, in the situation when crossover occurs involving parents with different architectures, the described procedure produces a child with a number of neurons that is not greater than the maximum number of neurons in one of the parents. It is also possible to generate offspring with fewer number of neurons that parents have. On the one hand, this is an advantage of the developed operator, because it allows to find the ANN with the least number of neurons. On the other hand, this action in the initial phase of the algorithm's operation can lead to the elimination of all individuals with larger structures. As a consequence, in the further phase of the algorithm's operation, it may not be possible to create individuals who will adapt to the learning data with satisfactory accuracy. The predominance of individuals with a smaller number of neurons in the initial phase of the algorithm's operation, where due to low weight matching, network's responses are far from perfect. This results from the smaller penalty part responsible for the size of the network (see *fitness function* in this section). In order to prevent the phenomenon of bad quality of mapping caused by an insufficient number of neurons, an additional evolutionary operator is introduced responsible for adding new neurons to the network – the neuron mutation operator (see Fig. 4).

Besides the different numbers of neurons in the layers, individuals differ in the levels of delays, i.e. the number of weights of neurons in the first hidden layer. The devised NAS algorithms enable that only the weights corresponding to the same inputs are crossed. For example, the weight from parent 1 responsible for the input signal delayed in the second level  $u(k-2)$  can be crossed only with the corresponding weight from parent 2, i.e. the weight responsible for input signal delayed in the second level. Hence, in the first step of the crossover operator delays of parents are crossed as follows:

$$du_{new} = round(rdu_{p1} + (1-r)du_{p2}), \quad (9)$$

$$dy_{new} = round(rdy_{p1} + (1-r)dy_{p2}), \quad (10)$$

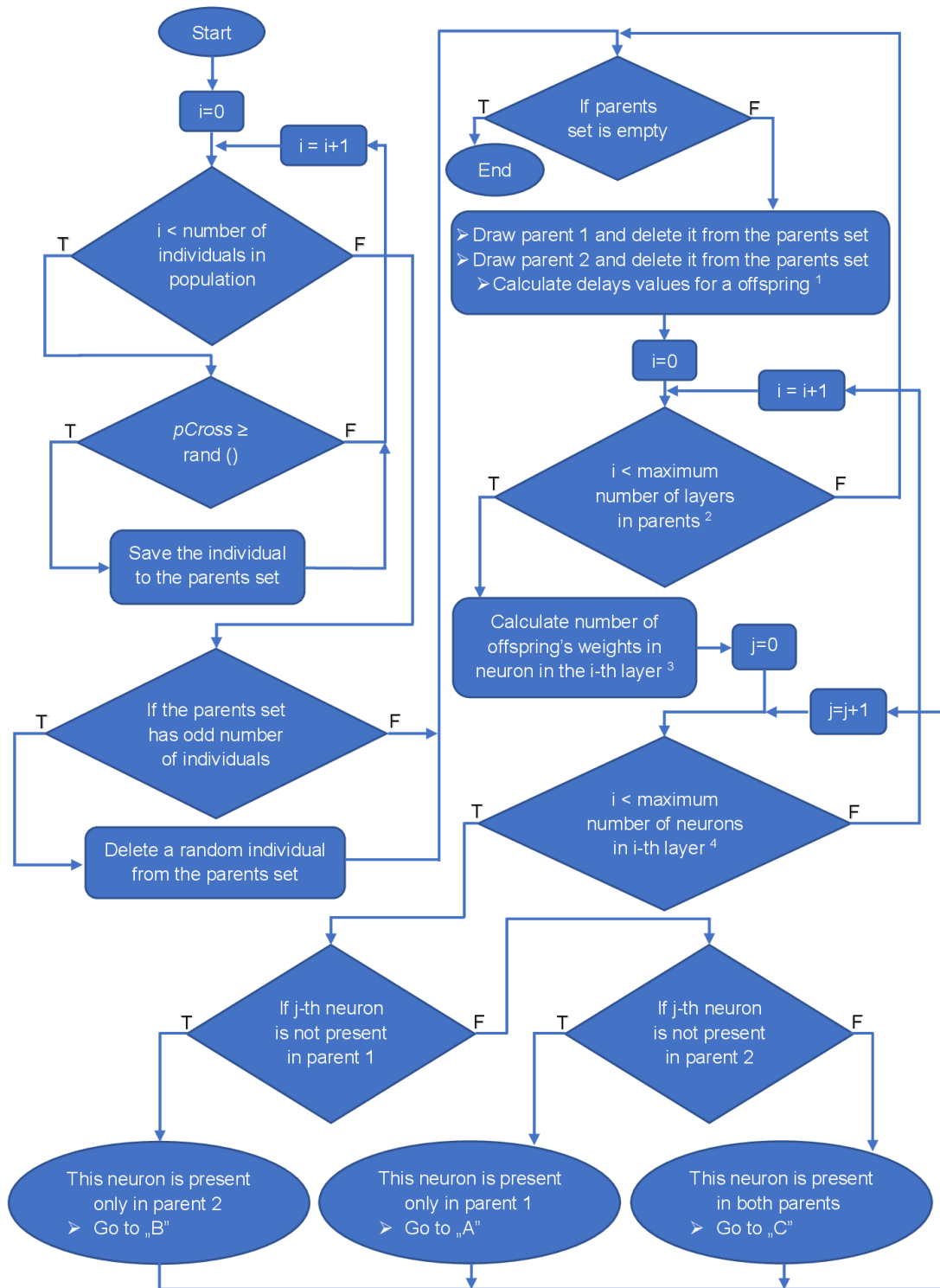
where:  $du_{new}, dy_{new}$  denote input delay level and recursive delay level of offspring, respectively;  $du_{p1}, du_{p2}$  are levels of input delays of parents 1 and 2;  $dy_{p1}, dy_{p2}$  signifies levels of recursive delays of parents 1 and 2;  $r$  stands for the random number in the range  $[0,1]$ , drawn separately for each equation.

Differences in the structures and delays levels of parents' also lead to a different number of weights in the corresponding neurons. If a given weight exists in both parents, the weight in the child is calculated from the following linear combination of parents' weights [37]:

$$w_{new} = rw_{p1} + (1-r)w_{p2}, \quad (11)$$

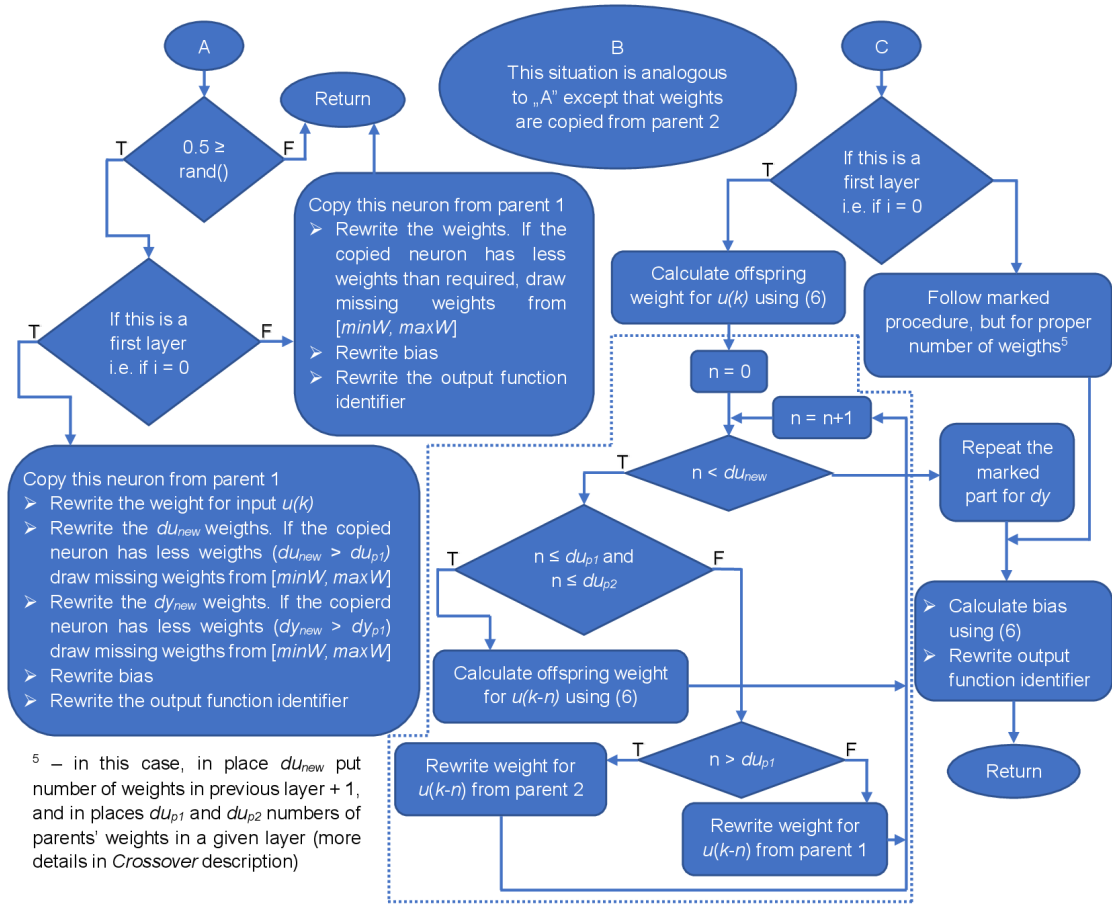
where:  $w_{new}$  denotes weight of offspring;  $w_{p1}, w_{p2}$  are weights of parents 1 and 2.

If the weight corresponding to an input to a neuron exists only in one of the parents, it is rewritten. Because each neuron has a bias, the value of the offspring bias is always calculated according to (11). In order to make the above description more transparent, a scheme of the crossover operator is shown in Fig. 7.



<sup>1</sup> – the levels of offspring's delays are calculated according to the equations (4) and (5); <sup>2</sup> – number of layers in the parent who has more of them; <sup>3</sup> – the equations (2) describe number of neurons in first layer (when  $i = 0$ ). The number of required weights in next layers equals number of neuron in previous layer + 1 (for bias); <sup>4</sup> – the number of neurons in a  $i$ -th layer in parents, who has more of them; in other words maximum number of neurons in the  $i$ -th layers in both parents;

a: A scheme of crossover operator (continuation, i.e. "go to sections": A, B or C is presented in Fig. 7b)



b: A scheme of crossover operator (continuation of the scheme from Fig. 7a)

Figure 7: A scheme of crossover operator.

It should be noticed that this way of exchanging information between parents (recombination) is justified. The weight of a given input in parent 1 is crossed with the weight corresponding to a given input in parent 2 or is completely rewritten. The same operation happens with entire neurons. If both parents have a neuron in a given place of the network, they are crossed with each other. However, if only one parent has a neuron in a given place, then the child inherits it with a probability of 50%, and this neuron is located in the first free place in the given layer. Moreover, thanks to the neuron placement in the first free place in a given layer the transfer of significant neurons, i.e. those with the main positive effect on the network response, to the upper positions of the layers is ensured. This makes it possible to transfer their characteristic features (set of weights) to the neurons of other individuals. This action also saves the memory space used for the algorithm, because the gaps created after neurons are not saved. It is worth adding that copying of whole groups of neurons is not considered. In the ANNs of relatively small size, this is not a disadvantage. However, in the case of creating networks of large sizes, an algorithm should be developed that allows crossing networks based on entire groups of neurons.

### Fitness function

In order to evaluate the fitness of an ANN, it is necessary to determine its response to the learning data, and then with selected measure check their fit to target response. In the RNN the calculation of the first samples of the network's responses requires the completion of the delayed input and output samples. It depends on the considered problem, and in this work under these values the nominal position of the control rods and the nominal thermal power of the reactor (see section 1) scaled to the value 1, have been written. The value of the fitness function is calculated as follows:

$$f_{\text{adapt},i} = 10 - p_1 \bar{e}_i - p_2 N_i - p_3 D_i, \quad (12)$$



where:  $f_{\text{adapt},i}$  is the value of fitness function for  $i$ th individual;  $p_1, p_2, p_3$  denote the values of weights in the fitness function;  $D_i = du + dy$  stands for the sum of levels of delays occurring in the  $i$ th individual;  $N_i$  signifies the total number of neurons in the  $i$ th individual;  $\bar{e}_i$  is the mean error (represents the fit to the target response) for the  $i$ th individual calculated as:

$$\bar{e}_i = \frac{1}{n} \sum_{k=1}^n |y_{i,k} - y_{\text{ref},k}|, \quad (13)$$

where:  $n$  is the number of samples in target response;  $y_{i,k}$  denotes subsequent samples of the  $i$ th individual response;  $y_{\text{ref},k}$  stands for subsequent target response samples.

The value of the fitness function depends not only on the quality of the RNN fit but also on its architecture. This fact is included in the penalty part of (12), i.e. fitness function is decreased depending on the number of neurons present in the network and the number of delays in the input and output signals. The values of weights in this part, i.e.  $p_1, p_2$  and  $p_3$  are 1, 0.01 and 0.0001, respectively. They have been chosen so that a network containing one more neuron and the same degree of delays have to generate a response whose average error would be at least 0.01 lower. The value 0.01 results from the specifics of the problem under consideration. Clearly, by dividing the original PWR reactor heat output (target response) by its nominal power the response can be understood as the percentage power output. The value of 1 corresponds to the nominal power of the reactor, i.e. 100% of its load. It has been subjectively assumed that a network with fewer neurons can be wrong by 1% more of power on each signal sample. Similar reasoning is used to determine the value of  $p_3$ , except that a network having one degree of delay less can have 0.01% greater error on each sample. This value is smaller than for  $p_2$ , because the computational cost introduced due to a longer delay is less than the computational cost associated with introduction of additional neuron.

It is worth mentioning that the values of fitness function are evaluated in every developed NAS algorithm in the same way according to (12). This allows for a direct comparison of the results obtained by the investigated algorithms.

### Selection

It is well-known that in nature the well-fitted individuals, and thus those with a high-value fitness function, live longer. Moreover, they have a greater chance of multiplication, generating at the same time a greater number of offspring who inherit their features. This operator, such as in nature provides a greater likelihood of reproduction for more fitted individuals. It is because these individuals are more likely to survive to the next population, where they will have the chance to be crossed again. The selection operator  $\mu + \lambda$  type is used in the devised NAS algorithms. Clearly, the chance of transition to a new population is given to both individuals formed in a given population and in the previous one, and in detail the roulette method with elitism is used [37].

#### 3.1.1. DNAS1 algorithm

The domain of searched RNN architecture is limited by setting the  $du$  and  $dy$  delays of all individuals in a given population to a constant value (see table 1) of 5 in the DNAS1 algorithm. Moreover, in order to enable the building of RNNs of larger sizes and ensuring better fit to learning data the neuron mutation-deleting operator is not used. The selection operator consists of the roulette method as well as elitism. The *hmBest* number of the best individuals is directly transferred to the next population. In turn, the remaining individuals, i.e.  $popSize - hmBest$  are drawn using the roulette method.

#### 3.1.2. DNAS2 algorithm

In the DNAS2 algorithm the domain of searched RNN architecture also includes adjustment of the  $du$  and  $dy$  values (see table 1). These values are changed in the mutation and crossover operators. Other operating conditions of DNAS2 algorithm are identical to DNAS1 algorithm.

#### 3.1.3. DNAS3 algorithm

In order to limit random events (e.g. drawing of missing weights during the crossover) and reasonable evaluation time of the DNAS2 algorithm, subsequent mechanisms inspired by nature have been introduced. These include biogeographical zones, ecological divergence and competitive exclusion [23]. Algorithm DNAS2 developed in this way is the third NAS algorithm, so-called DNAS3 algorithm. An exemplary effect of using the above mechanisms is the crossover of individuals with similar features (species). Thus, features that improve the fitness of an individual are

transferred more often. The consequence of this is that individual features in a population are isolated and strengthened, and dominate over others. Hence, considering the problem of missing weights, the RNNs with a different number of neurons in the hidden layer are treated as separate species. This approach described in [11, 32], utilise the segregation of individuals of the population concerning the similarity of their structure. Therefore, a crossover operator can operate only on individuals (networks) belonging to one species and only within it (intra-species crossover). Such operation is aimed at faster convergence to the optimal RNN weights for a given structure. However, it cannot be guaranteed that in a given population, e.g., the initial one, there will be at least one individual of the optimal species, i.e. a network with the appropriate number of neurons in the hidden layer. Also, in order to avoid the evolution operations for species that differ significantly from the currently optimal species, the evolution operators have been provided, that is responsible for creating species slightly different from the original. It enables reaching from one species (e.g. not optimal) to another even significantly different (e.g. optimal) through numerous small changes, which is consistent with the theory of gradualism [23].

The mechanisms presented above have been implemented in DNAS3 algorithm in the following way. The population is initialised with *popSize* number of individuals, allowing all of them to crossover with all of the rest. When a given population is dominated by one species, only the *hmBest* number of the best individuals from the dominant species, and the same number of individuals from two secondary species are allowed to develop further through the intra-species crossover. The secondary species have one neuron more or less than the dominant species in the hidden layer. The intra-species crossover means that the offspring can be formed only from parents belonging to one species. At the same time, due to the small number of individuals of the dominant species and of the secondary species, the probability of crossover *pCross* rate increases to 1. The used crossover operator does not differ from the one used in DNAS2 algorithm (see section 3.1). A given species becomes dominant if the *hmBest* number of the most fit individuals in the population belongs to it. If a species loses domination in the population, the inter-species crossover is restored, *pCross* is then reduced to 0.2, and the population's size is increased to a maximum number expressed by *popSize*, giving the chance to dominate the population by the new species. The creation of new species has been implemented using the described function of creating new neurons and deleting already existing neurons (see section 3.1 - the neuron mutation operator). Keeping in mind that, new species can occur as a result of the crossover. In comparison to DNAS1 and DNAS2 algorithms, the roulette method is not used to speed up the convergence of the algorithm. The next population receives a certain number of best-fitness individuals ("full elitism") depending on whether the population is dominated or not. In the case of a non-dominated population, this is the maximum number (*popSize*) of best-fitness individuals. Because the number of all existing individuals in a given population may be smaller, then it is possible that all individuals will pass to the next population. In the situation of dominance to the next population, as it has been mentioned above, passes *hmBest* number of individuals of each species separately, i.e. *hmBest* number of individuals of the dominant species and *hmBest* number of each of the two secondary ones, in total  $3hmBest$  of individuals. A general scheme of the designed DNAS3 algorithm is presented in Fig. 8.

### 3.2. DNAS4 algorithm

The idea of the DNAS4 algorithm is different from the algorithms presented previously. It is because in DNAS4 algorithm the classic gradient descent methods are used to change the weights, whereas the EA is used to select the number of hidden layers, the number of neurons in every layer, levels of delays  $du$  and  $dy$ , and a type of gradient descent algorithms. The direct motivation to develop this approach is related to the concluded tests of DNAS1 – DNAS3 algorithms. In details, it has been observed during the tests that the optimal or suboptimal RNN structure is mainly determined in the initial phases of activity of the NAS algorithm. Whereas the subsequent RNN improvement through the mutation of weights takes a large number of generations. Taking this into account, the DNAS4 algorithm using the common gradient learning methods has been developed.

As with DNAS1 – DNAS3 algorithms, the general scheme of EA in DNAS4 algorithm is in line with Fig. 2. However, the selection, mutation and crossover operators have been changed compared to DNAS1 – DNAS3 algorithms. Moreover, retraining is used in this algorithm. The retraining is repeated learning, i.e. reselection of RNN weights of a given individual with other initial values of weights. Nevertheless, this operation does not always ensure the improvement of fitness of a given individual. Therefore, in every generation, only selected individuals are given to retraining. These individuals are randomly selected with the *pRetrain* probability. The well-known gradient methods have been chosen for learning purposes. These are Levenberg-Marquardt algorithm [22, 43], backward propagation of Bayesian regularisation [15] and backward propagation of scaled gradient [55]. The  $\mu + \lambda$  selection is applied again, and the maximum *popSize* of the best-fitness individuals (elitism) is selected. Hence, the selection operator has been

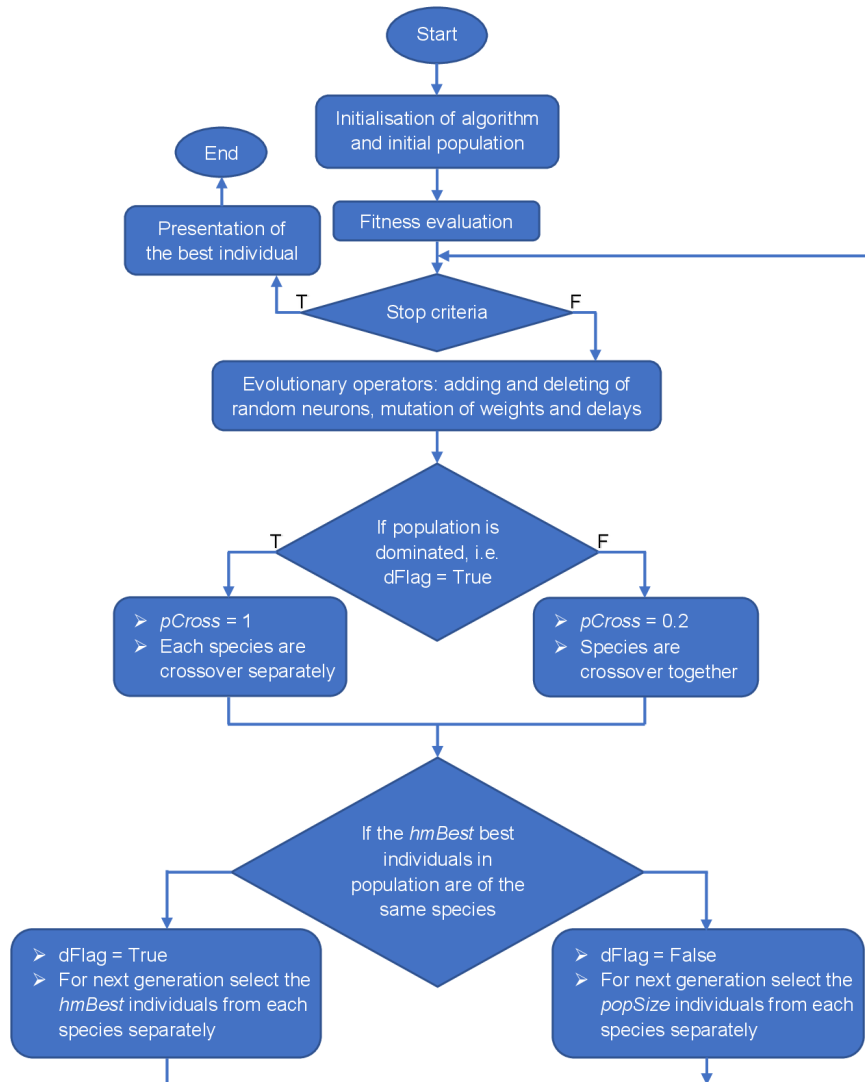


Figure 8: A general scheme of the designed DNAS3 algorithm.

modified in such a way that only the best individual with a given architecture, i.e. only the best individual of a given species is transferred to the next population. The individuals belong to the same species when they have the same number of layers, neurons in each layer,  $du$ ,  $dy$  and method of training. Thanks to this, the variable population size has been obtained, which reduces the number of calculations and accelerates the result's achievement.

As it has been aforementioned, the mutation in DNAS4 algorithm differs from this operator in the algorithms presented previously. It is because the RNN weights are not mutated. In turn, the structure and the rest of the RNN parameters represented by each of the individuals from the population has a chance to mutate with a probability of  $pMut$ . If an individual is mutated, it is equally likely that the number of hidden layers, the number of neurons in each layer, both levels of delays or the learning method will be mutated. Nevertheless, only one feature for a given individual is changed during one generation of the algorithm. The mutation of the number of hidden layers involves a random removal of a layer or inserting a new layer with a random number of neurons in a random place of the network. In turn, the neurons' mutation involves the deletion or the addition of one neuron to each layer. A random draw is done separately for each individual's layer, and neuron's addition or removal, and is equally likely to 50% chance of addition or deletion. In the situation, when the last neuron is removed from a layer, the entire layer is deleted. The mutation of levels of delays  $du$  and  $dy$  is the equally likely to increase or decrease for each level by one. Whereas the mutation of

the learning method is based on the equally probable selection from among all available methods, except that which has been already assigned to the individual at the time of the mutation.

The crossover operator consists of the exchange of individuals' features concerning the RNN structure, and learning method. In contrast to DNAS1 – DNAS3 algorithms, weights are not crossed, which results in the required training of new individuals. In the first step, similarly to DNAS1 – DNAS3 algorithms, the pairs of parents are randomly selected. Next, the following features of offspring are calculated:

- the number of hidden layers as:

$$L_{\text{new}} = \text{round} (rL_{p1} + (1 - r)L_{p2}), \quad (14)$$

where:  $L_{\text{new}}$  is the number of offspring hidden layers;  $L_{p1}, L_{p2}$  denote the number of hidden layers in parent 1 and 2, respectively;

- the number of neurons in the  $i$ th hidden layer of offspring as:

$$N_{i,\text{new}} = \text{round} (rN_{i,p1} + (1 - r)N_{i,p2}), \quad (15)$$

where:  $N_{i,\text{new}}$  is the number of neurons in  $i$ th offspring hidden layer;  $N_{i,p1}, N_{i,p2}$  denote the number of neurons in  $i$ th hidden layers in parent 1 and 2, respectively;

It should be noticed that if a given layer occurs only in one parent, then the number of neurons is calculated on the assumption that the number of neurons corresponding to the second parent equals zero.

- the level of the  $i$ th delay as:

$$d_{i,\text{new}} = \text{round} (rd_{i,p1} + (1 - r)d_{i,p2}), \quad (16)$$

where:  $d_{i,\text{new}}$  is the level of the  $i$ th delay for offspring;  $d_{i,p1}, d_{i,p2}$  denote the levels of  $i$ th delays in parent 1 and 2, respectively;

- the offspring inherits learning method from one of the parents, where each parent has a 50% chance to transfer his method.

## 4. Case study

In general, the DNAS1 – DNAS4 algorithms have been simulation-verified in an analogous way. However, for the DNAS1 – DNAS3 algorithms, 100 calls of a particular algorithm have been made and each of them counted 100 generations. In turn, the DNAS4 algorithm has been called 46 times with 25 generations. This was due to the long computation time of this algorithm. As it has been mentioned above, the aim has been to find an RNN with an optimal (minimum) architecture while at the same time obtaining satisfactory accuracy of its response. Such a network is a desirable SISO black-box model of the fast processes in PWR. The values of the particular input parameters of devised NAS algorithms are shown in table 3.

### 4.1. Framework

In order to solve a given task using the EAs, the task can be either transferred to a proper form for the algorithm or the algorithm can be adapted to suit the task [2, 37]. The classic genetic algorithms are associated with the first approach. In turn, the use of evolutionary algorithms allows for omitting ANN's architecture encoding and decoding for operators, which can speed up EA operation [2, 37]. However, the adaptation of evolutionary operators to the NAS problem requires a proper way of saving data. In other words, a proper manner of coding solutions. It is because the efficiency of a given algorithm depends largely on this operation [37].

The devised algorithms have been implemented in the Matlab environment. All variables and constants in these algorithms have been saved in the decimal floating-point format. Whereas the structure of RNNs is saved in the cell arrays. These are a data type with indexed data containers called "cells". Each cell can contain every data type, including other cell or cell array. This feature is used to repeatedly nest subsequent cell arrays. The authors are aware that this way of saving the structure of RNNs does not ensure the fastest possible operation of algorithms. However, this has been done because the additional aim of the implementation has been to make the code easy to interpret by

**Table 3**  
The values of the input parameters for DNAS1 – DNAS4 algorithms.

Values of input parameters					
Symbol	Value				Comment
	DNAS1	DNAS2	DNAS3	DNAS4	
<i>maxLay</i>	1	1	1	1	-
<i>maxNinLay</i>	20	20	20	20	this value has been chosen experimentally and it depends on the correlation between the input and output (target) data; during numerical experiments, it has been observed that if the smaller correlation appears then the bigger number of neurons is required (see also [30])
<i>du</i>	5	(1, <i>duMax</i> )	(1, <i>duMax</i> )	(1, <i>duMax</i> )	the value <i>duMax</i> has been chosen experimentally and it is equal to 50
<i>dy</i>	5	(1, <i>dyMax</i> )	(1, <i>dyMax</i> )	(1, <i>dyMax</i> )	the value <i>dyMax</i> has been chosen experimentally and it is equal to 50
<i>popSize</i>	50	50	50	50	in DNAS3 and DNAS4 algorithms the size of the population are changing during their work
<i>pCross</i>	0.8	0.8	0.8	0.8	-
<i>p<sub>i</sub></i>	1, 0.01, 0.0001	1, 0.01, 0.0001	1, 0.01, 0.0001	1, 0.01, 0.0001	see section 3.1
<i>minDelta</i>	0.0001	0.0001	0.0001	0.0001	-
<i>maxDelta</i>	0.1	0.1	0.1	0.1	-
<i>pMutW</i>	0.2	0.2	0.2	0.2	-
<i>pMut</i>	0.2	0.2	0.2	0.2	-
<i>pMutNewN</i>	0.2	0.2	0.2	0.2	-
<i>pMutD</i>	-	0.2	0.2	-	-
<i>pMutDelN</i>	-	-	0.2	-	-
<i>minW</i>	-1	-1	-1	-	(*)
<i>maxW</i>	1	1	1	-	(*)
<i>hmBest</i>	5	5	5	5	-
<i>pRetrain</i>	-	-	-	0.2	-
(*) the values [-1,1] of the initial weights [ <i>minW</i> , <i>maxW</i> ] are because: a) this range is bipolar and symmetrical with respect to zero, therefore ensures that negative values for the activation function are accounted for; b) the range of the input signals to the designed ANN is [-2.196, 0] whereas the range of the target output signal is [0, 1.184] (see section 4.2), thus the weights in considered interval should ensure calculating (re-scaling) input signal to the desirable levels of the output signal;					

other users. Moreover, this approach allows relatively easy transfer of prepared software to open source environments, e.g., Python. As it has been aforementioned, in developed algorithms populations consist of individuals representing the architectures of RNNs. In turn, the RNNs are composed of layers, and the layers consist of neurons. Each neuron consists of a set of weights (with bias) and an identifier of the activation function. Therefore, a table of cells is available in which under the next indexes individuals of populations are placed. Each individual consists of the cell array, wherein each next array corresponds to the next level of penetration of the network, and a numeric array, in which two non-negative integers are placed determining the level of inputs and recursive outputs delays. The first array corresponds to the whole network and contains as many elements as the given network has layers. Each layer is built from neurons, so each array, which corresponds to the given layer, is built from next cell arrays, which correspond to neurons. Each neuron array consists of two numerical arrays. In the first numerical array are determined the neuron weights and bias whereas in the second a numeric value takes place, which identifies the activation function of a given neuron. The

above description is illustrated in Fig. 9 to ensure its transparency.

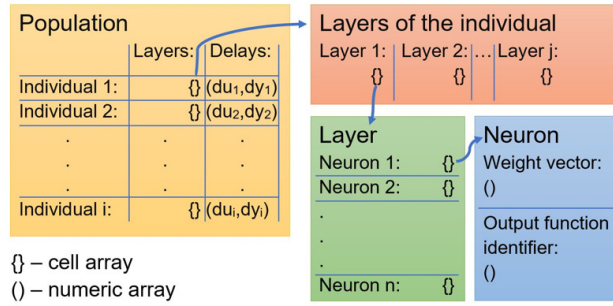


Figure 9: The way of saving of a given RNN architecture in the devised algorithms.

## 4.2. PWR

In accordance with the considerations presented before, the DNAS1 – DNAS4 algorithms have been developed for creating the SISO black-box model of the fast processes in PWR. Therefore, the position of the control rods and the thermal power of the PWR are the selected input and output signals, respectively. They are presented in Figs. 10 and 11 [44, 47]. It should be added that the trajectory in Fig. 11 is scaled. Clearly, the physical sense of this reference response is the average thermal power of the PWR scaled by dividing it by the nominal average thermal power of the reactor. This means that the value 1 corresponds to the nominal thermal power (see table 4).

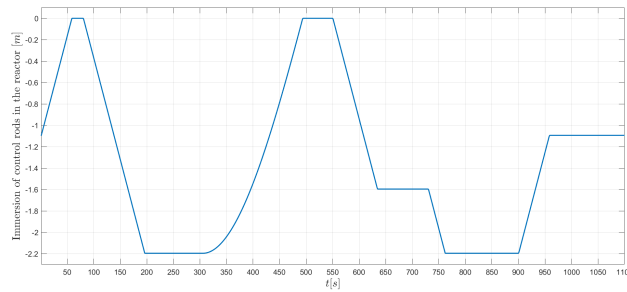


Figure 10: The target position of the control rods in a PWR.

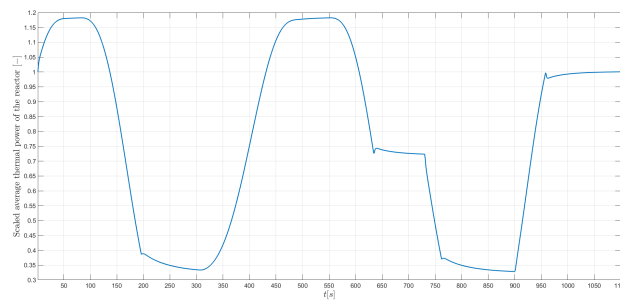


Figure 11: The target trajectory of thermal power in a PWR.

## 4.3. Results

In this section, the simulation results illustrating the performance of the proposed algorithms are presented. First, the learning (training) phase using trajectories from Figs. 10 and 11 is shown. Next, the verification phase is discussed. Moreover, the results obtained by the proposed algorithms are presented against the results generated by the NARX-based exhaustive search algorithm from Matlab [62] and the basic NEAT algorithm implemented in Python [16, 55]. During the performed experiment with NARX-based exhaustive search algorithm the following parameters were set: number of hidden layers = 1, number of neurons in hidden layers =  $max25$ ,  $du = max50$ ,  $dy = max50$ , learning



**Table 4**

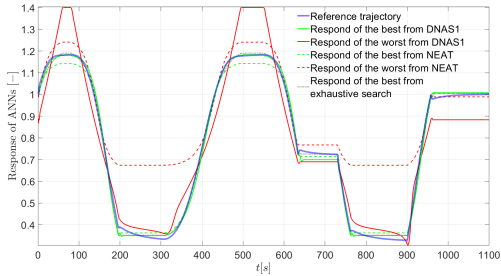
The values of PWR parameters.

Values of PWR parameters		
Name	Value	Unit
The minimal location of the control rods	-2.196	m
The nominal location of the control rods	-1.098	m
The maximal location of the control rods	0	m
The nominal thermal power of the reactor	3 436	MW

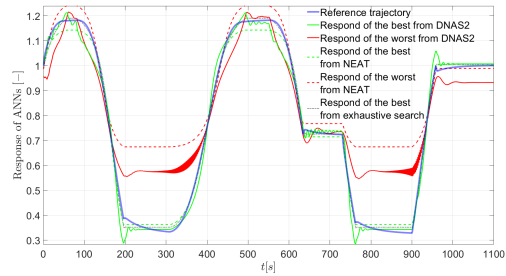
method - Levenberg-Marquardt algorithm. The experiment was repeated 10 times. Whereas, during the performed experiment with the basic NEAT algorithm, its task was simplified because the penalty part dependent on the delay levels ( $du$  and  $dy$ ) was not considered in the objective function. This was since the NEAT algorithm can create feedbacks of different types [16, 55]. The stop condition was taken as 100 generations and algorithm was called 100 times.

**4.3.1. Learning phase**

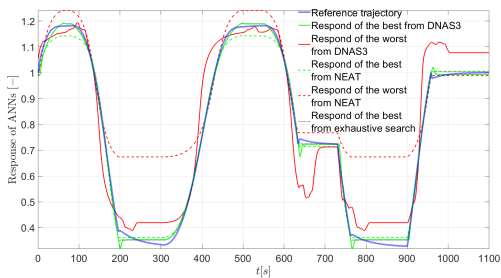
The simulation results showing the generated trajectories of the average thermal power of the reactor by the best and worst individual from the set of best individuals (relative to the value of the fitness function) for the DNAS1 – DNAS4 algorithms are shown in Fig. 12. Whereas the trajectories representing the average value of the fitness functions of the whole population and the average value of the mean errors are illustrated in Fig. 13. In turn, the distributions of the number of best individuals with a given number of neurons in the hidden layer (with a given RNN structure), obtained during calls of the particular algorithms are presented in Fig. 14. It should be noticed that over each bar, there are two numbers. The first of them specify the average value of the fitness function whereas the second denotes the mean error calculated according to (13). Thus, for example for DNAS1 algorithm, the highest number of best individuals (23) possess 4 neurons in the hidden layer.



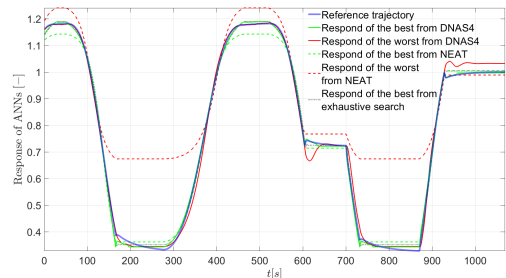
**a:** The average thermal power from DNAS1 algorithm



**b:** The average thermal power from DNAS2 algorithm



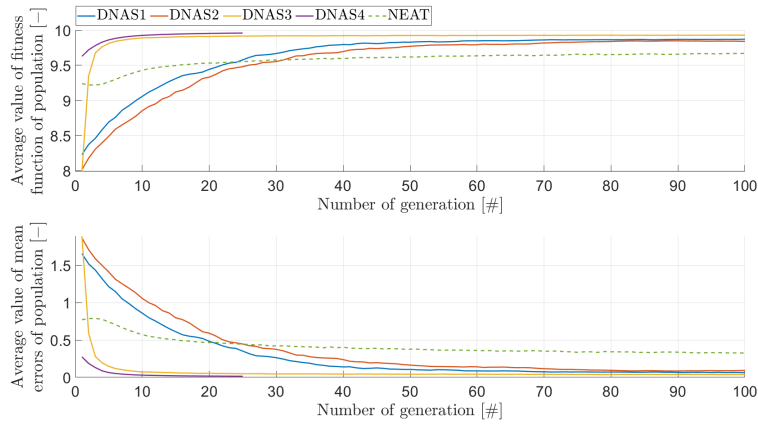
**c:** The average thermal power from DNAS3 algorithm



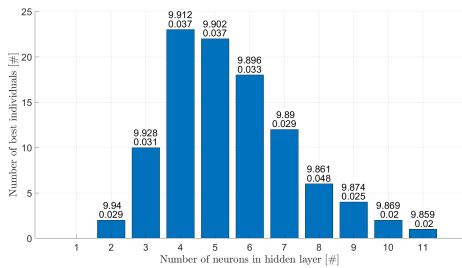
**d:** The average thermal power from DNAS4 algorithm

**Figure 12:** The generated trajectories of the average thermal power of the reactor in the learning phase.

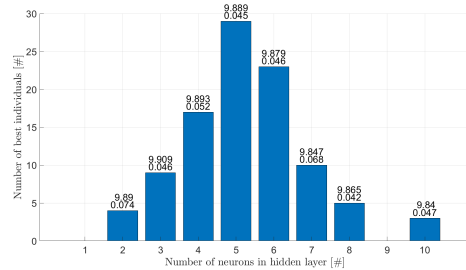
## An automatic selection of optimal recurrent neural network architecture



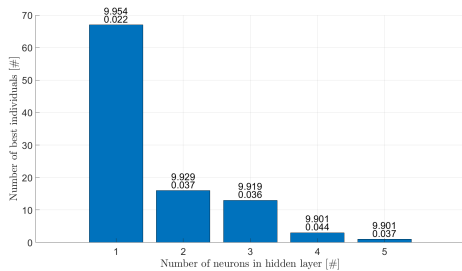
**Figure 13:** The trajectories of the average value of the fitness functions and the mean errors.



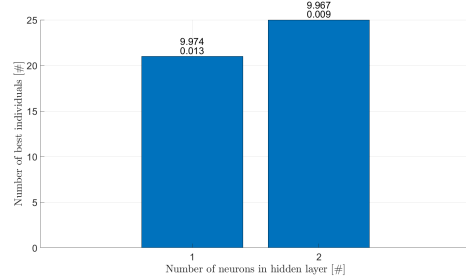
**a:** The distributions of the number of best individuals in DNAS1 algorithm



**b:** The distributions of the number of best individuals in DNAS2 algorithm



**c:** The distributions of the number of best individuals in DNAS3 algorithm



**d:** The distributions of the number of best individuals in DNAS4 algorithm

**Figure 14:** The distributions of the number of best individuals in the learning phase.

Analysing Figs. 12 – 14 it can be noticed that the best performance of the generated responses by the devised RNNs while having the smallest network architecture is provided by the DNAS3 and DNAS4 algorithms. Clearly, the obtained RNNs from DNAS3 and DNAS4 algorithms seem to be promising candidates for the black-box model of the fast processes in a PWR. Hence, the optimality which is understood as achieving the desired trade-off between the size of obtained RNN and the accuracy of black-box model responses may be ensured. In order to further illustrate this, several indicators are summarised in table 5. Taking into account the values of indicators from table 5 it can be observed that the performance of the generated response by the DNAS2 algorithm is worse than the results obtained with the DNAS1 algorithm while significantly increasing the computation time. This is due to the incomparably larger space of searched solutions in the DNAS2 algorithm. This problem is eliminated in DNAS3 algorithm, where the search space is the same as in DNAS2 algorithm. However, the efficiency of this algorithm as measured by the computation time as well as the accuracy of the response it generates is significantly higher. Moreover, the size of the provided

**Table 5**

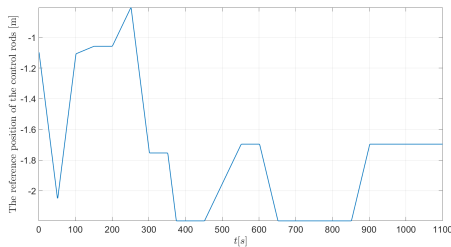
The values of indicators for DNAS1 – DNAS4 algorithms.

Values of indicators				
Indicator	DNAS1	DNAS2	DNAS3	DNAS4
The average value of mean errors	0.0343	0.0500	0.0270	0.0108
The average value of the number of neurons in the hidden layer	5.43	5.26	1.55	1.54
The average value of the $du$	$const = 5$	25.21	24.10	25.35
The average value of the $dy$	$const = 5$	23.43	20.41	11.00
The average value of the computational time for a hundred generations	37 min	68 min	18 min	912 min*
* the duration of calculations for 100 generations have been calculated from the proportion using the registered calculation times for 25 generations				

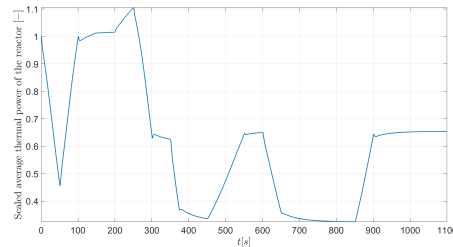
network is much smaller than in DNAS1 and DNAS2 algorithms. As it can be noticed above completely different is DNAS4 algorithm, which gets its advantage in the performance of RNN response through the use of gradient selection of weights, but it pays off with the longest calculation time. Of course, the final test for the developed algorithms is the verification phase, where the input data have not been used during the learning phase. The results obtained are presented in the next section.

**4.3.2. Verification phase**

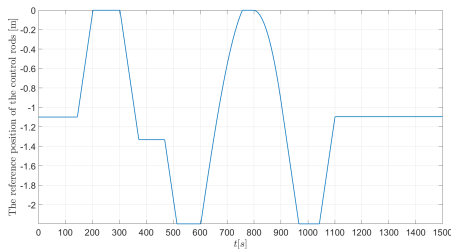
The reference trajectories using during the verification phase are shown in Fig. 15. As it can be noticed the two data sets have been used during this phase. It is worth adding that those trajectories are prepared based on considerations contained in [47]. The simulation results illustrating the generated trajectories of the average thermal power of the reactor by the best and worst individual from the set of best individuals (relative to the value of the fitness function) for the DNAS1 – DNAS4 algorithms are given in Figs. 16 and 17 for the first and second data set, respectively. In turn, the average value of mean errors of the response of the best individuals achieved in subsequent calls of particular algorithms for the data sets 1 and 2 are presented in table 6.



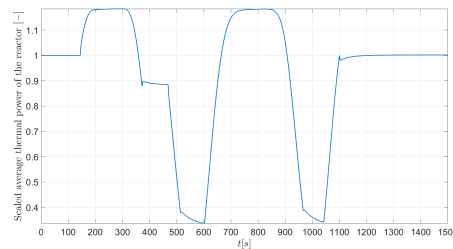
**a:** The reference position of the control rods in a PWR – data set 1.



**b:** The reference trajectory of thermal power in a PWR – data set 1.



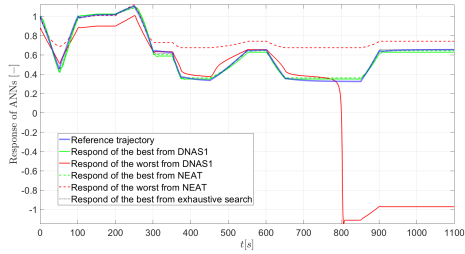
**c:** The reference position of the control rods in a PWR – data set 2.



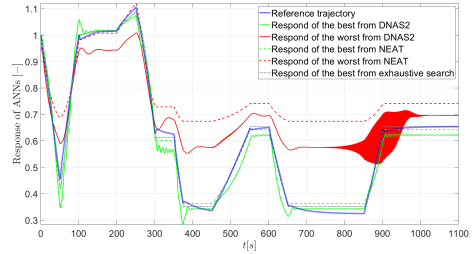
**d:** The reference trajectory of thermal power in a PWR – data set 2.

**Figure 15:** The reference trajectories of position of the control rods and thermal power in a PWR during the verification phase.

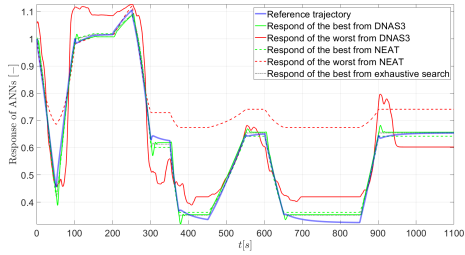
## An automatic selection of optimal recurrent neural network architecture



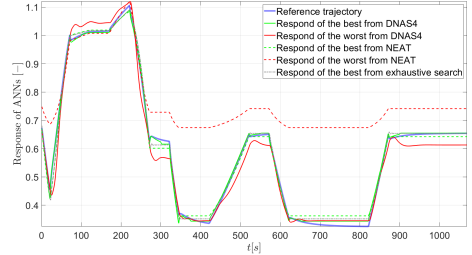
**a:** The average thermal power from DNAS1 algorithm – data set 1



**b:** The average thermal power from DNAS2 algorithm – data set 1

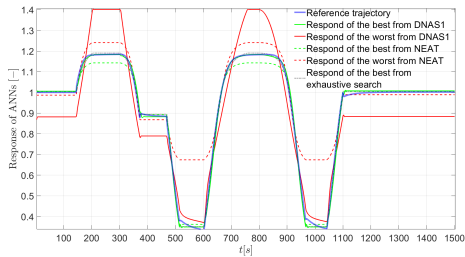


**c:** The average thermal power from DNAS3 algorithm – data set 1

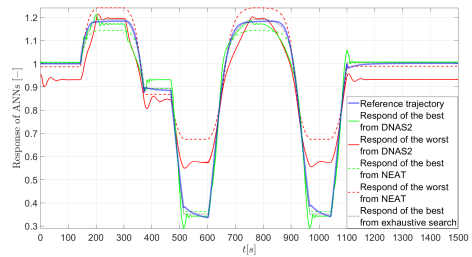


**d:** The average thermal power from DNAS4 algorithm – data set 1

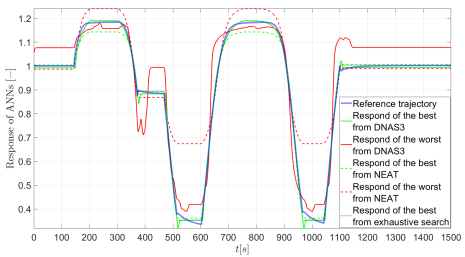
**Figure 16:** The generated trajectories of the average thermal power of the reactor in the verification phase for the first data set.



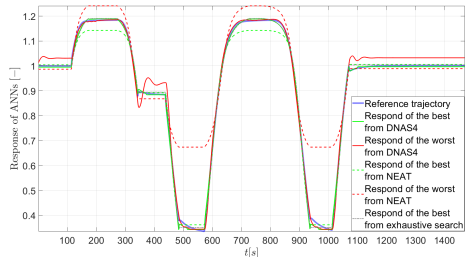
**a:** The average thermal power from DNAS1 algorithm – data set 2



**b:** The average thermal power from DNAS2 algorithm – data set 2



**c:** The average thermal power from DNAS3 algorithm – data set 2



**d:** The average thermal power from DNAS4 algorithm – data set 2

**Figure 17:** The generated trajectories of the average thermal power of the reactor in the verification phase for the second data set.

**Table 6**

The average value of mean errors for DNAS1 – DNAS4 algorithms during the verification phase.

The average value of mean errors						
Name	DNAS1	DNAS2	DNAS3	DNAS4	NEAT	Exhaustive search
The average value of mean errors – set 1	0.0379	0.0788	0.0308	0.0255	0.0410	0.0117*
The average value of mean errors – set 2	0.0341	0.0487	0.0237	0.0117	0.0620	0.0083*
* - values for the obtained best individual relative to the fitness function						

Analysing the trajectories presented in Figs. 16 and 17 and the average value of mean errors from table 6 and taking into account conclusions of the learning phase (see section 4.3.1), it can be stated that the best developed NAS algorithm is DNAS3 algorithm. Moreover, similarly good results are obtained with the DNAS4 algorithm; however, their cost understood as the computing time is very high. Therefore, the following is stated:

- the developed DNAS3 and DNAS4 algorithms may provide at least a good basis for NAS algorithms for the behavioural (black-box) modelling purposes,
- the RNNs obtained by the DNAS3 and DNAS4 algorithms can be at least a prototype of the SISO black-box model of the fast processes in a PWR.

Moreover, analysing the results obtained from exhaustive search algorithm it can be concluded that they confirm the results obtained with the DNAS1 – DNAS4 algorithms. More specifically, in the framework of the exhaustive search algorithm performed, the best results are obtained by the network that has one neuron in the hidden layer, with negligible impact on the quality of its performance of the values  $du$  and  $dy$ . Hence, the exhaustive search algorithm performed after the fact, using the knowledge gained from the research conducted by the authors, confirms the effectiveness of the developed DNAS1 – DNAS4 algorithms to perform the role of a black-box model of the fast processes in a PWR. Also, it can be concluded that the basic NEAT algorithm provides neural networks with response performance significantly worse than the developed algorithms, primarily DNAS3 and DNAS4 in the verification phase. Thus, the obtained black-box model of the considered processes in a PWR using the basic NEAT algorithm would be worse than the one obtained using the developed DNAS1 – DNAS4 algorithms.

## 5. Conclusions

In this paper, the problem of developing algorithms of artificial neural network architecture search for black-box modelling purposes has been investigated. In particular, four algorithms of artificial neural network architecture search using an evolutionary algorithm and also gradient descent methods have been devised to create the desired models. The specialised evolutionary operators have been delivered to ensure the proper activity of particular algorithms. Finally, it enabled devising the single-input single-output model black-box model of the fast processes in a pressurized water reactor. This model provides the desired trade-off between the size of the network structure and the accuracy of black-box model responses. The proposed algorithms have been implemented in Matlab environment and obtained results yield satisfying performance of the generated output trajectories.

The fact that the created artificial neural networks are not large in size raises the question of whether manual methods using gradient learning would not be enough to search for their optimal structures. The authors claim that no because the number of parameters defining even a small network is too large. Moreover, the developed solutions are further developed both for other types of artificial neural networks, i.e. spiking neural networks and deep neural networks, and for building MIMO models of processes in a pressurized water reactor, e.g., changes in coolant and fuel temperature.

## References

- [1] Agarwal, M., Jain, N., Kumar, M., Agrawal, H., 2010. Face recognition using eigen faces and neural networks. *International Journal of Computer Theory and Engineering* 2, 2–7. doi:10.7763/IJCTE.2010.V2.213.
- [2] Ahmadizar, F., Soltanian, K., Akhlaghiantab, F., Tsoulos, I., 2015. Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. *Engineering Applications of Artificial Intelligence* 39, 1–13. doi:10.1016/j.engappai.2014.11.003.

- [3] Ahn, B.S., Cho, S.S., Kim, C.Y., 2000. Integrated methodology of rough set theory and artificial neural network for business failure prediction. *Expert Systems with Applications* 18, 65–74. doi:10.1016/S0957-4174(99)00053-6.
- [4] Alom, M.Z., Taha, T.M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M.S., Hasan, M., Van Essen, B.C., Awwal, A.A.S., Asari, V.K., 2019. A state-of-the-art survey on deep learning theory and architectures. *Electronics* 8, 292–358. doi:https://doi.org/10.3390/electronics8030292.
- [5] Azzini, A., Tettamanzi, A.G.B., 2011. Evolutionary ANNs: State of the art survey. *Intelligenza Artificiale* 5, 19–35. doi:10.3233/IA-2011-0002.
- [6] Bartlett, E., Basu, A., 1991. A dynamic node architecture scheme for backpropagation neural networks, in: *Proceedings of the Artificial Neural Networks in Engineering*, pp. 559–564.
- [7] Basu, A., 1992. Nuclear Power Plant Status Diagnostics Using an Artificial Neural Network with Dynamic Node Architecture. Ph.D. thesis. Iowa State University. Ames, Iowa, US.
- [8] Bhattacharyya, D., Kim, T.H., Lee, G.S., 2011. Use of artificial neural network in bengali character recognition. *Communications in Computer and Information Science* 260, 140–152. doi:10.1007/978-3-642-27183-0\_15.
- [9] Bhushan, B., 2009. Biomimetics: Lessons from nature - an Overview. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367, 1445–1486. doi:10.1098/rsta.2009.0011.
- [10] Billings, S.A., 2013. *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-temporal Domains*, 1st Edition. John Wiley & Sons, Inc., London, UK.
- [11] Bishop, C.M., 2006. *Pattern Recognition and Machine Learning*. Springer.
- [12] Brudzewski, K., Osowski, S., Markiewicz, T., 2004. Classification of milk by means of an electronic nose and SVM neural network. *Sensors and Actuators, B: Chemical* 98, 291–298. doi:10.1016/j.snb.2003.10.028.
- [13] Candy, J.V., 2006. *Model-Based Signal Processing*. John Wiley & Sons, Inc., Hoboken, New Jersey, US.
- [14] Chen, C.H., 1996. *Fuzzy Logic and Neural Network Handbook*. McGraw-Hill, Michigan, US.
- [15] Chua, L.O., Yang, L., 1988. Cellular neural networks: Applications. *IEEE Transactions on Circuits and Systems* 35, 1273–1290. doi:10.1109/31.7601.
- [16] CodeReclaimers, Accessed on: Apr. 08, 2021. Python implementation of NEAT. URL: <https://neat-python.readthedocs.io/en/latest/>.
- [17] Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2, 303–314. doi:https://doi.org/10.1007/BF02551274.
- [18] Dede, G., Sazli, M.H., 2010. Speech recognition with artificial neural networks. *Digital Signal Processing: A Review Journal* 20, 763–768. doi:10.1016/j.dsp.2009.10.004.
- [19] Dreiseitl, S., Ohno-Machado, L., 2002. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics* 35, 352–359. doi:10.1016/S1532-0464(03)00034-0.
- [20] Ellefsen, K.O., Huizinga, J., Torresen, J., 2020. Guiding neuroevolution with structural objectives. *Evolutionary Computation* 28, 115–140. doi:https://doi.org/10.1162/evco\_a\_00250.
- [21] Floreano, D., Dürr, P., Mattiussi, C., 2008. Neuroevolution: From architectures to learning. *Evolutionary Intelligence* 1, 47–62. doi:10.1007/s12065-007-0002-4.
- [22] Fukuda, T., Shibata, T., 1992. Theory and applications of neural networks. *IEEE Transactions on Industrial Electronics* 39, 472–489. doi:10.1007/978-1-4471-1833-6.
- [23] Futuyma, D.J., Kirkpatrick, M., 2018. *Evolution*, 4th Edition. Oxford University Press, Oxford, UK.
- [24] Gadoue, S.M., Giaouris, D., Finch, J.W., 2009. Sensorless control of induction motor drives at very low and zero speeds using neural network flux observers. *IEEE Transactions on Industrial Electronics* 56, 3029–3039. doi:10.1109/TIE.2009.2024665.
- [25] Gupta, T.K., Raza, R., 2019. Optimization of ANN architecture: A review on nature-inspired techniques, in: Dey, N., Borra, S., Ashour, A.S., Shi, F. (Eds.), *Machine Learning in Bio-Signal Analysis and Diagnostic Imaging*. Academic Press, pp. 159–182.
- [26] Haykin, S.O., 2011. *Neural Networks and Learning Machines*, 3rd Edition. Pearson Education, London, UK.
- [27] Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359–366. doi:https://doi.org/10.1016/0893-6080(89)90020-8.
- [28] Kandel, E.R., Schwartz, J.H., Jessell, T.M., Siegelbaum, S.A., Hudspeth, A.J., 2012. *Principles of Neural Science*, 5th Edition. McGraw-Hill.
- [29] Kashiwao, T., Nakayama, K., Ando, S., Ikeda, K., Lee, M., Bahadori, A., 2017. A neural network-based local rainfall prediction system using meteorological data on the internet: A case study using data from the Japan Meteorological Agency. *Applied Soft Computing* 56, 317–330. doi:https://doi.org/10.1016/j.asoc.2017.03.015.
- [30] Kavzoglu, T., 1999. Determining optimum structure for artificial neural networks, in: *Proceedings of the 25th Annual Technical Conference and Exhibition of the Remote Sensing Society*, pp. 675–682.
- [31] Kerlin, T.W., 1978. Dynamic analysis and control of pressurized water reactors. *Control and Dynamic Systems* 14, 103–212. doi:https://doi.org/10.1016/B978-0-12-012714-6.50008-8.
- [32] Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. *Science* 220, 671–680. doi:10.1126/science.220.4598.671.
- [33] Koza, J.R., 1998. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, 6th Edition. Massachusetts Institute of Technology, Cambridge, Massachusetts, US.
- [34] Krizhevsky, A., Sutskever, I., Hinton, G.E., 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* 60, 84–90. doi:10.1145/3065386.
- [35] Kurkova, V., 1992. Kolmogorov’s theorem and multilayer neural networks. *Neural Networks* 5, 501–506. doi:https://doi.org/10.1016/0893-6080(92)90012-8.
- [36] Llobet, E., Hines, E.L., Gardner, J.W., Franco, S., 1999. Non-destructive banana ripeness determination using a neural network-based elec-



- tronic nose. *Measurement Science and Technology* 10, 538–548. doi:10.1088/0957-0233/10/6/320.
- [37] Michalewicz, Z., 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.
- [38] Naghedolfeizi, M., 1990. *Dynamic Modeling of a Pressurized Water Reactor Plant for Diagnostics and Control*. Master's thesis. University of Tennessee. Knoxville, US.
- [39] Nolfi, S., Parisi, D., 1997. Evolution of artificial neural networks. *Neural Networks* 2, 1–8. doi:10.1016/j.microc.2008.10.006.
- [40] Norgaard, M., Ravn, O., Poulsen, N.K., Hansen, L.K., 2000. *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook*. Springer Verlag, London, UK.
- [41] Papavasileiou, E., Cornelis, J., Jansen, B., 2021. A systematic literature review of the successors of 'neuroevolution of augmenting topologies'. *Evolutionary Computation* 29, 1–73. doi:https://doi.org/10.1162/evco\_a\_00282.
- [42] Perrusquía, A., Yu, W., 2021. Identification and optimal control of nonlinear systems using recurrent neural networks and reinforcement learning: An overview. *Neurocomputing* 438, 145–154. doi:https://doi.org/10.1016/j.neucom.2021.01.096.
- [43] Ponulak, F., Kasiński, A., 2011. *Introduction to spiking neural networks: Information processing, learning and applications*. *Acta Neurobiologiae Experimentalis* 71, 409–433.
- [44] Puchalski, B., 2018. *Sterowanie z wykorzystaniem rachunku niecałkowitego rzędu reaktorem wodnym ciśnieniowym elektrowni jądrowej* (in Polish). Ph.D. thesis. Gdańsk University of Technology. Gdańsk, Poland.
- [45] Puchalski, B., Rutkowski, T.A., 2020. Approximation of fractional order dynamic systems using elman, GRU and LSTM neural networks, in: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (Eds.), *Artificial Intelligence and Soft Computing. ICAISC 2020. Lecture Notes in Computer Science*, Springer, Cham. pp. 215–230. doi:https://doi.org/10.1007/978-3-030-61401-0\_21.
- [46] Puchalski, B., Rutkowski, T.A., Duzinkiewicz, K., 2017. Nodal models of pressurized water reactor core for control purposes – a comparison study. *Nuclear Engineering and Design* 322, 444–463. doi:https://doi.org/10.1016/j.nucengdes.2017.07.005.
- [47] Puchalski, B., Rutkowski, T.A., Duzinkiewicz, K., 2020. Fuzzy multi-regional fractional PID controller for pressurized water nuclear reactor. *ISA Transactions* 103, 86–102. doi:https://doi.org/10.1016/j.isatra.2020.04.003.
- [48] Rios, J.D., Alanis, A.Y., Arana-Daniel, N., Lopez-Franco, C., 2020. Neural networks modeling and control: Applications for unknown nonlinear delayed systems in discrete time, in: Sanchez, E.N. (Ed.), *Neural Networks Modeling and Control: Applications for Unknown Nonlinear Delayed Systems in Discrete Time*. Academic Press.
- [49] Roffel, B., Betlem, B., 2006. *Process Dynamic and Control. Modelling for Control and Prediction*. John Wiley & Sons, Inc., Chichester, West Sussex, UK.
- [50] SaiSindhuTheja, R., Shyam, G.K., 2021. An efficient metaheuristic algorithm based feature selection and recurrent neural network for DoS attack detection in cloud computing environment. *Applied Soft Computing* 100. doi:https://doi.org/10.1016/j.asoc.2020.106997.
- [51] Saxena, A., Saad, A., 2007. Evolving an artificial neural network classifier for condition monitoring of rotating mechanical systems. *Applied Soft Computing Journal* 7, 441–454. doi:10.1016/j.asoc.2005.10.001.
- [52] Siebel, N.T., Sommer, G., 2007. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems* 4, 171–183. doi:10.3233/HIS-2007-4304.
- [53] Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R., 2019. Designing neural networks through neuroevolution. *Nature Machine Intelligence* 1, 24–35. doi:10.1038/s42256-018-0006-z.
- [54] Stanley, K.O., D'Ambrosio, D.B., Gauci, J., 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life* 15, 185–212. doi:10.1162/artl.2009.15.2.15202.
- [55] Stanley, K.O., Miikkulainen, R., 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10, 99–127.
- [56] Stubberud, S.C., Lobbia, R.N., Owen, M., 1995. An adaptive extended kalman filter using artificial neural networks, in: *Proceedings of the 34th IEEE Conference on Decision and Control*, pp. 1852–1856.
- [57] Sun, X., Chen, L., Yang, Z., Zhu, H., 2013. Speed-sensorless vector control of a bearingless induction motor with artificial neural network inverse speed observer. *IEEE/ASME Transactions on Mechatronics* 18, 1357–1366. doi:10.1109/TMECH.2012.2202123.
- [58] Sutskever, I., Martens, J., Hinton, G.E., 2011. Generating text with recurrent neural networks, in: *Proceedings of the 28th International Conference on Machine Learning*, pp. 528–535.
- [59] Tam, K.Y., Kiang, M.Y., 1992. Managerial applications of neural networks: The case of bank failure predictions. *Management Science* 38, 926–947. doi:10.1287/mnsc.38.7.926.
- [60] Tatjewski, P., 2007. *Advanced Control of Industrial Processes. Structures and Algorithms*. Springer-Verlag, London, UK.
- [61] Tetko, I.V., Livingstone, D.J., Luik, A.I., 1995. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Modeling* 35, 826–833. doi:10.1021/ci00027a006.
- [62] The MathWorks, I., Accessed on: Apr. 08, 2021. Design time series NARX feedback neural networks. URL: <https://ch.mathworks.com/help/deeplearning/ug/design-time-series-narx-feedback-neural-networks.html>.
- [63] Turner, A.J., Miller, J.F., 2014. Neuroevolution: Evolving heterogeneous artificial neural networks. *Evolutionary Intelligence* 7, 135–154. doi:10.1007/s12065-014-0115-5.
- [64] Vincent, J.F.V., Bogatyreva, O.A., Bogatyrev, N.R., Bowyer, A., Pahl, A.K., 2006. Biomimetics: Its practice and theory. *Journal of The Royal Society Interface* 3, 471–482. doi:10.1098/rsif.2006.0127.
- [65] Yaot, X., 1993. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems* 8, 539–567. doi:https://doi.org/10.1002/int.4550080406.
- [66] Yu, W., Gonzalez, J., Li, X., 2021. Fast training of deep LSTM networks with guaranteed stability for nonlinear system modeling. *Neurocomputing* 422, 85–94. doi:https://doi.org/10.1016/j.neucom.2020.09.030.

## An automatic selection of optimal recurrent neural network architecture



Krzysztof Laddach received the M.Sc. degree (Hons.) in control engineering from the Faculty of Electrical and Control Engineering, the Gdańsk University of Technology in 2019. His master's dissertation was focused on algorithms of artificial neural network structure search for processes dynamics modelling purposes. Since October 2019 he has been a Ph.D. student in the Doctoral School at the Gdańsk University of Technology. His research interests involve mathematical modelling, especially in connection with the use of computational intelligence (artificial intelligence), optimal selection of the architecture of artificial neural networks, and the use of artificial neural networks in estimation.



Rafał Łangowski received the M.Sc. and the Ph.D. degrees (Hons.) in control engineering from the Faculty of Electrical and Control Engineering at the Gdańsk University of Technology in 2003 and 2015, respectively. From 2007 to 2014, he held the specialist as well as manager positions at ENERGA, one of the biggest energy enterprises in Poland. Since February 2014, he has been an owner of VIDEN a business in energy and control areas. He provides theoretical and practical experience, especially in front and back office at energy company and operation of the energy market in Poland. From 2016 to 2017, he was a Senior Lecturer with the Department of Control Systems Engineering at the Gdańsk University of Technology. He is currently an Assistant Professor with the Department of Electrical Engineering, Control Systems and Informatics. His research interests involve mathematical modelling and identification, estimation methods, especially state observers, and monitoring of large scale complex systems.



Tomasz A. Rutkowski received the M.Sc. degree in automatics and robotics and the Ph.D. (Hons.) degree in automatic control from the Gdańsk University of Technology in 2000 and 2004, respectively. He has experience in control systems of critical infrastructure, such as environmental systems (drinking water distribution systems and wastewater treatment plant systems) and power systems, including nuclear power plants. He is currently an Assistant Professor with the Department of Electrical Engineering, Control Systems and Informatics, Gdańsk University of Technology. His current research interests include mathematical modelling, advanced control algorithms, estimation algorithms, computational intelligence techniques, and industrial control systems.



Bartosz Puchalski received a MSc and PhD (Hons.) degree in control engineering from the Faculty of Electrical and Control Engineering at the Gdańsk University of Technology in 2011 and 2018 respectively. In 2012 he began to work at the Gdańsk University of Technology as a lecturer. In 2013, he received the engineer degree in the discipline of Electrical Engineering at the same Faculty. From 2018, he is employed at his alma mater as an assistant professor. Currently, his main scientific interests are focused on the control, identification and modelling of dynamic systems, fractional order calculus and recursive neural networks.