Applied Soft Computing 47 (2016) 653–668

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Evolving flexible beta basis function neural tree using extended genetic programming & Hybrid Artificial Bee Colony

Souhir Bouaziz[a,*], Habib Dhahri[a], Adel M. Alimi[a], Ajith Abraham[b,c]

[a] REsearch Groups in Intelligent Machines (REGIM-Lab), University of Sfax, National School of Engineers (ENIS), BP 1173, Sfax 3038, Tunisia
[b] Faculty of Electrical Engineering and Computer Science, Technical University of Ostrava, Czech Republic
[c] Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, WA, USA,

ARTICLE INFO

Article history:
Received 13 December 2012
Received in revised form 19 March 2016
Accepted 21 March 2016
Available online 10 May 2016

Keywords:
Flexible beta basis function neural tree model
Extended Genetic Programming
Hybrid Artificial Bee Colony algorithm
Time-series forecasting

ABSTRACT

In this paper, a new hybrid learning algorithm is introduced to evolve the flexible beta basis function neural tree (FBBFNT). The structure is developed using the Extended Genetic Programming (EGP) and the Beta parameters and connected weights are optimized by the Hybrid Artificial Bee Colony algorithm. This hybridization is essentially based on replacing the random Artificial Bee Colony (ABC) position with the guided Opposite-based Particle Swarm Optimization (OPSO) position. Such modification can minimize the delay which might be lead by the random position, in reaching the global solution. The performance of the proposed model is evaluated for benchmark problems drawn from time series prediction area and is compared with those of related methods.

1. Introduction

The inefficiency of classical optimization algorithms to solve complex optimization problems such as highly non-linear problems encouraged researchers to employ natural or behavioral metaphors. This category of algorithms is called Evolutionary Computation (EC) [1]. Broadly speaking, evolutionary computing techniques include: evolution-inspired algorithms (such as: Genetic Algorithm [2], Differential Evolution [3]); swarm-inspired algorithms (such as: Ant Colony Optimization [4], Particle Swarm Optimization [5]); and other population-based algorithms such as Teaching–Learning-Based Optimization (TLBO) [6,7].

Recently Artificial Bee Colony (ABC) algorithm inspired of the bee behavior, was proposed by Karaboga in 2005 [8] for solving numerical optimization problems. Karaboga and Akay compared the ABC algorithm with GA, PSO, DE and Evolution Strategies (ES) optimization algorithms on large set of numerical test functions and results show that the performance of the ABC is competitive with that of other population-based algorithms [9]. The ABC algorithm is successfully used for many real-world problems. In fact, it can efficiently be applied on training feed-forward neural networks [10,11] for pattern classification. Nandy et al. proposed, in Ref. [12], the ABC based back-propagation algorithm to optimize the back-propagation neural network training and hence the analysis of this method is performed over standard data sets, showing its efficiency in terms of convergence speed and rate. The ABC algorithm is also effectively used in energy field like the work presented by Rashidi et al. in Ref. [13] where the parametric study and optimization of regenerative Clausius and organic Rankine cycles with two feed-water heaters were achieved basing on artificial neural network and ABC algorithm. This ABC algorithm was applied to medical pattern classification and clustering problems [14,15] and to solve TSP problems [16].

Although ABC algorithm is an efficient algorithm for many optimization problems, there still have several weaknesses, such as it easily gets stuck in the local optima for some complex multimodal problem and its random candidate food position may lead to delay in reaching the global solution [9]. These issues encourage some researchers to hybridize it with other algorithms such as the hybridization with DE algorithm proposed by Kong et al. [17]; and the hybridization with PSO algorithm introduced by Shi et al. [18] for global numerical optimization. In the present work, a hybridization between ABC algorithm and Opposite-based PSO [19,57,58] algorithm, called ABC-OPSOP, is proposed to optimize the Beta Basis Function Neural Network (BBFNN) parameters.

* Corresponding author.
E-mail addresses: souhir.bouaziz@ieee.org, souhir.bouaziz@gmail.com (S. Bouaziz), habib.dhahri@ieee.org (H. Dhahri), adel.alimi@ieee.org (A.M. Alimi), ajith.abraham@ieee.org (A. Abraham).

http://dx.doi.org/10.1016/j.asoc.2016.03.006
1568-4946/

In fact, a BBF neural network's performance depends mainly on two issues which are the design of the network structure and the architecture parameter's adjustment automatically. In order to learn connected weights and Beta parameters which include the center, width and form parameters of BBFNNs, many methods are applied, i.e., back-propagation algorithm [20], genetic algorithm [21], DE algorithm [22], PSO algorithm [23] and so on. In addition, many important attempts have been developed to optimize both structure and parameters of the BBFNN such as Hierarchical Genetic Algorithm (HGA) [24] and Hierarchical Multi-dimensional Differential Evolution (HMDDE) [25].

Although conventional representation of BBFNN has a number of advantages such as better approximation capabilities and simple network topologies, however adapting the matrix-representation suffers from slow premature convergence characteristics and makes the BBFNN's structure difficult to regulate. These reasons motivate us to apply the tree-based encoding method which was introduced by Chen et al. [26–28], to represent a BBFNN. The new model is called flexible beta basis function neural tree (FBBFNT) [29,42,53,54].

In this paper, the FBBFNT model is applied to benchmark problems drawn from time series prediction area. Based on the predefined Beta operator sets, a flexible beta basis function neural tree model can be created and evolved. The hierarchical structure is evolved using the Extended Genetic Programming (EGP). The fine tuning of the Beta parameters (centre, spread and the form parameters) and weights encoded in the structure is accomplished using an Artificial Bee Colony algorithm modified by Opposite-based PSO positions (ABC-OPSOP). Experimental results and comparisons demonstrate the effectiveness and efficiency of the proposed ABC-OPSOP algorithm.

The remainder of this paper is organized as follows: Section 2 describes the basic flexible beta basis function neural tree. A hybrid learning algorithm for evolving the Beta function neural tree model is the subject of Section 3. The set of some simulation results are provided in Section 4. Finally, some concluding remarks are presented in Section 5.

2. Flexible beta basis function neural tree representation

In 1997, Alimi [30] proposed the use of Beta basis function as transfer function for training Beta Basis Function Neural Network. The Beta basis function has many advantages such as its large flexibility, its universal approximation characteristics [31] and its ability to generate rich shapes (asymmetry, linearity, etc.) [32].

In this work, a tree-based encoding method is adopted to represent the Beta basis function neural network instead of the matrix-based encoding, as it is more flexible and gives a more adjustable and modifiable architecture. In fact, matrix representation has limited flexibility in expressing topologies of the network structure with variable layers. Thereafter, evolutionary computation operators need to be applied carefully to preserve the topological constraints of networks. This representation is called flexible beta basis function neural tree (FBBFNT).

The FBBFNT is formed of a node set NS representing the union of function node set F and terminal node set T:

$$NS = F \cup T = \{\beta_2, \beta_3, \ldots, \beta_N, /_N\} \cup \{x_1, \ldots, x_M\} \quad (1)$$

where

- β_n ($n = 2, 3, \ldots, N$) denote non-terminal nodes and represent flexible Beta basis neurons with n inputs and N is the maximum degree of the tree.
- $/_N$ is the root node and represent a linear transfer function.
- $x_1, x_2, \ldots, x_M$ are terminal nodes and define the input vector values.

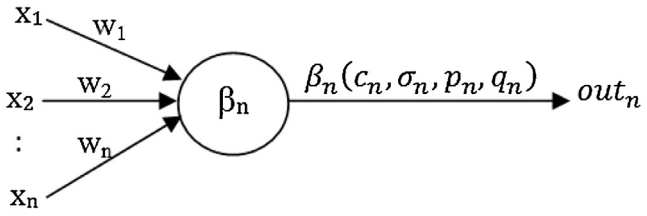

Fig. 1. A flexible neuron Beta operator.

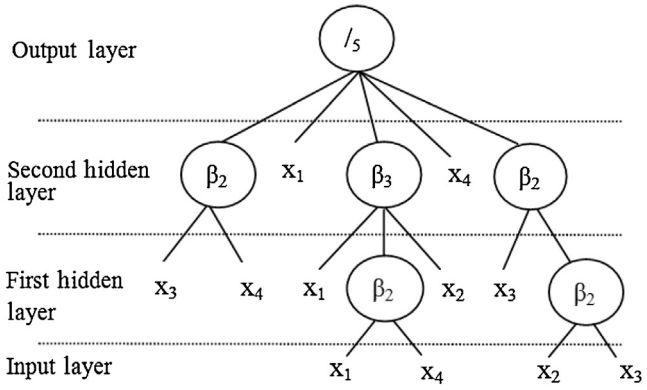

Fig. 2. A typical representation of FBBFNT: function node set $F = \{\beta_2, \beta_3, \beta_4, \beta_5, /_5\}$, and terminal node set $T = \{x_1, x_2, x_3, x_4\}$.

The output of a non-terminal node is calculated as a flexible neuron model (see Fig. 1).

If a function node, i.e., β_n is selected, n real values are randomly created to represent the connected weight between the selected node and its offspring. In addition, seen that the flexible activation function used for the hidden layer nodes is the Beta function, four adjustable parameters (the center c_n, width σ_n and the form parameters p_n, q_n) are randomly generated as flexible Beta operator parameters.

For each non-terminal node, its total excitation is calculated by:

$$y_n = \sum_{j=1}^{n} w_j \times x_j \quad (2)$$

where x_j ($j = 1, \ldots, n$) are the inputs of the selected node and w_j ($j = 1, \ldots, n$) are the connected weights.

The output of node β_n is then calculated by:

$$\text{out}_n = \beta_n(y_n; c_n, \sigma_n, p_n, q_n) = \begin{cases} [1 + \frac{(p_n + q_n)(y_n - c_n)}{\sigma_n p_n}]^{p_n} [1 + \frac{(p_n + q_n)(c_n - y_n)}{\sigma_n q_n}]^{q_n} \\ \text{if } y_n \in]c_n - \frac{\sigma_n p_n}{p_n + q_n}, c_n + \frac{\sigma_n q_n}{p_n + q_n}[\\ 0 \text{else} \end{cases} \quad (3)$$

The output layer yields a vector by linear combination of the node outputs of the last hidden layer to produce the final output.

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

3. The hybrid FBBFNT evolving algorithm

The optimization of FBBFNT includes the tree-structure and parameter optimization. In this study, finding an optimal or a near optimal Beta basis function neural tree structure is achieved by using Extended Genetic Programming algorithm (EGP) and the

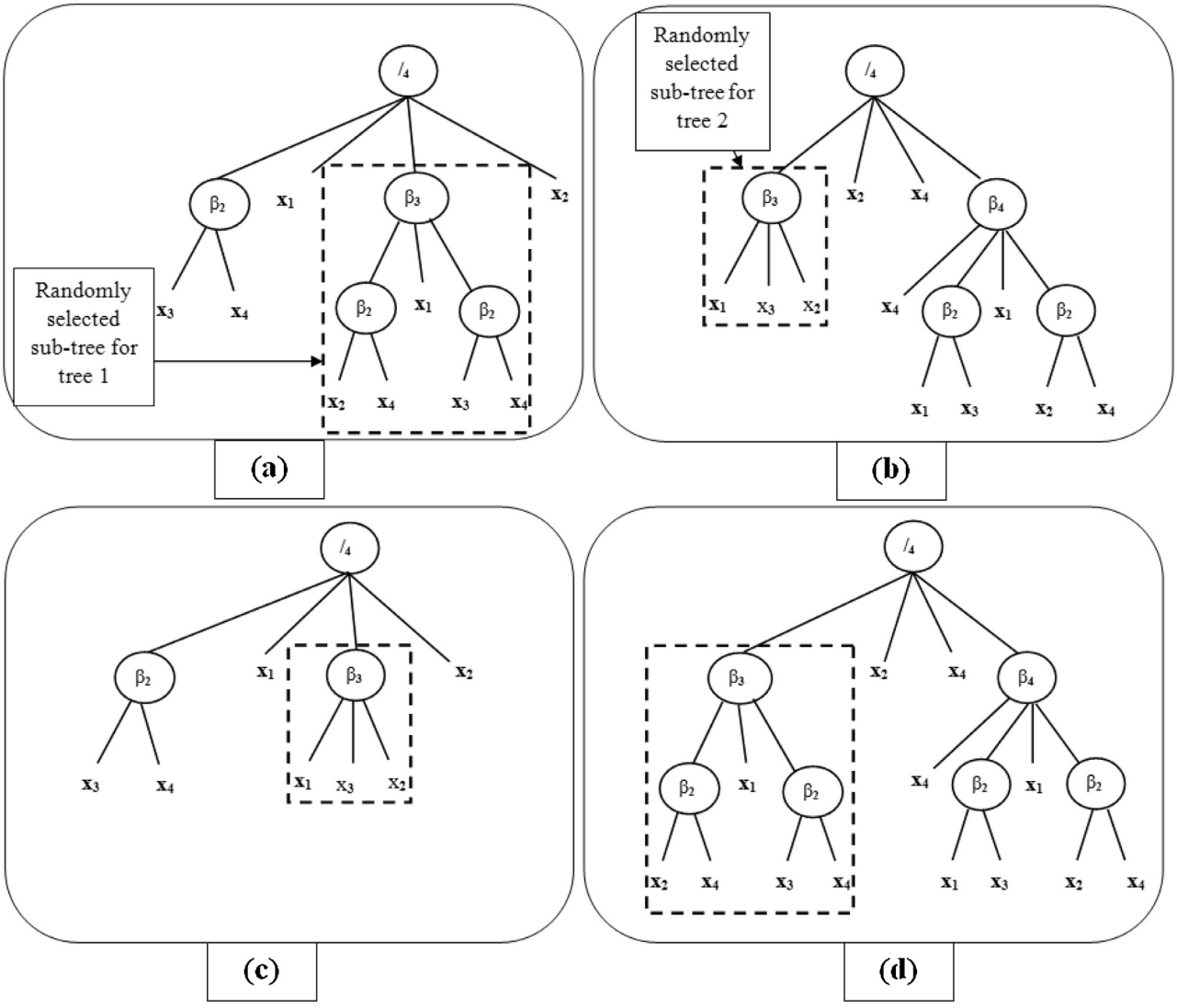

Fig. 3. Examples of the EGP crossover operator: (a) selected tree 1. (b) Selected tree 2. (c) Tree 1 after crossover operator with tree 2. (d) Tree 2 after crossover operator with tree 2.

parameters implanted in a FBBFNT are optimized by Artificial Bee Colony algorithm modified by Opposite-based PSO positions (ABC-OPSOP).

3.1. Structure optimization

The first step of the structure optimization is to create an initial population of flexible Bata basis function trees with random structures; i.e., uniformly distributed random number of layers in [3,*NL_Max*] and uniformly distributed random number of nodes for each layer in [2,*NN_Max*], where *NL_Max* (the maximum layer number) and *NN_Max* (the maximum node number) are chosen depending on the studied problem. The node parameters (Beta parameters and connected weights) of each tree are also randomly generated in its search spaces. Indeed, the search space of each node parameter is its interval limits: [*Minc*, *Maxc*] for *c*, [*Min*σ, *Max*σ] for σ, [*Minp*, *Maxp*] for *p*, [*Minq*, *Maxq*] for *q* and [0,1] for *w*. Each individual is then evaluated according to the structure fitness function.

In the second time, a number of flexible neural tree variation operators, which are an extension of standard Genetic Programming (GP).

3.1.1. Structure fitness function

According to Zhang and Mühlenbein [35] and many other researches [24,25], the fitness function adopted to find an optimal or a near-optimal Artificial Neural Network structure, depends mainly on both the ANN performance on the training data (such as the error and the training time or the number of functions evaluation) and the ANN complexity measurement (such as the number of nodes and the number of layers). The structure objective function to minimize which we used in our study is composed so of two measurements f_1 and f_2. The function f_1 measures the Root Mean Squared Error (RMSE) between the target and output of the proposed model. The function f_2 measures the complexity of the FBBFNT model.

$$\mathrm{Fit}_{\mathrm{stru}}(i) = \alpha f_1(i) + \delta f_2(i) \tag{4}$$

where α, δ are user specified fitness coefficients that allow a trade-off between the objectives, $\alpha, \delta \in [0,1]$.

The function $f_1(i)$ denoted the RMSE value of *i*-th individual, is as follows.

$$f_1(i) = \sqrt{\frac{1}{P}\sum_{j=1}^{P}\left(y_t^j - y_{\mathrm{out}}^j\right)^2} \tag{5}$$

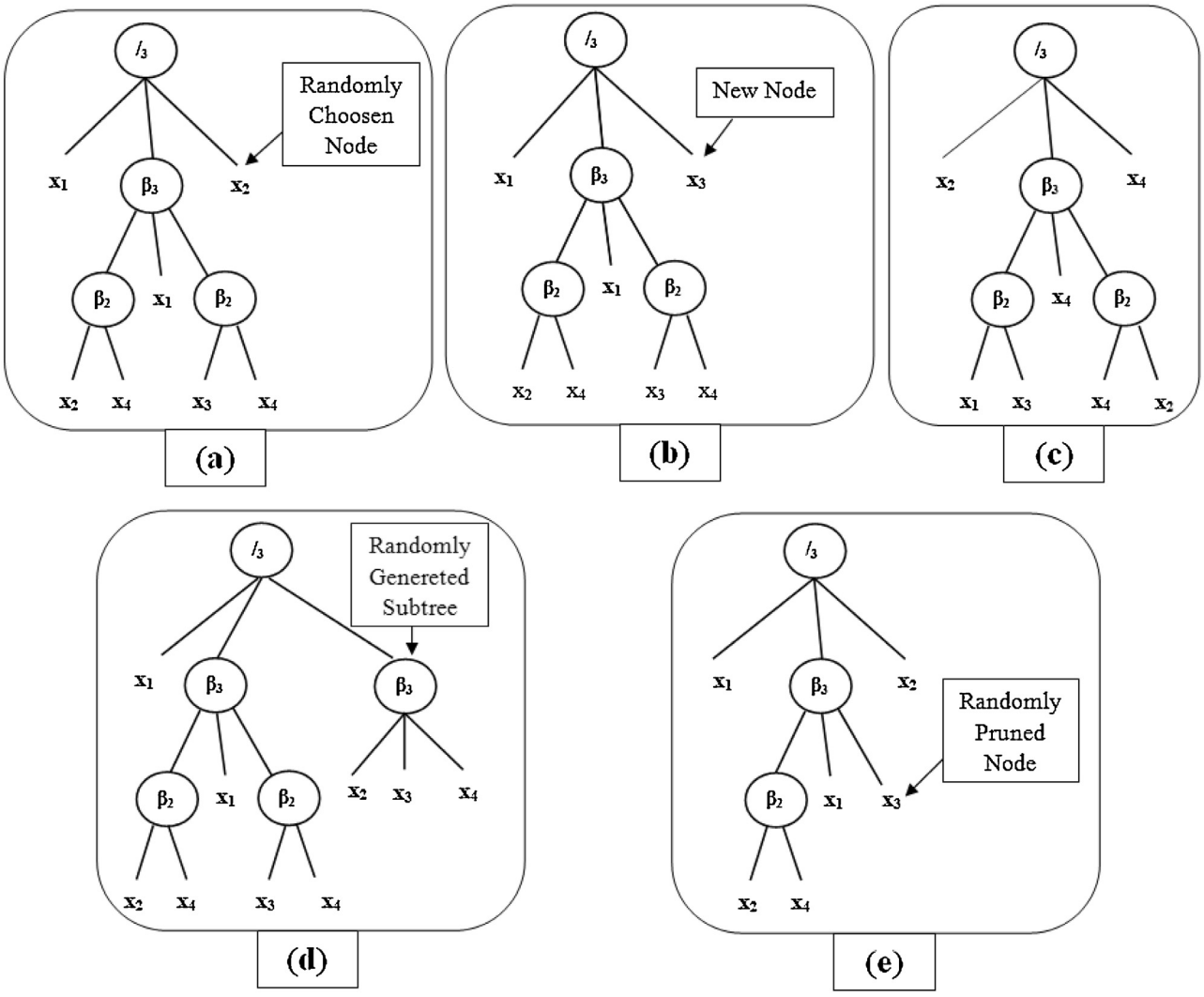

Fig. 4. Examples of the four EGP mutation operators: (a) original tree. (b) Changing one leaf node. (c) Changing all the leaf nodes. (d) Growing. (e) Pruning.

where P is the total number of samples, y_t^j and y_{out}^j are the desired output and the FBBFNT model output of j-th sample.

After a series of tuning experiments of the expression of $f_2(i)$ based on the number of structure nodes, the used expression is as follows.

$$f_2(i) = \frac{Size_i - SizeMin + 1}{SizeMax - Size_i + 1} \tag{6}$$

where $Size_i$ is the node number of the i-th individual. $SizeMin$ is equal to 5 and $SizeMax$ can be calculated as following:

$$SizeMax = \frac{NNMax^{NLMax} - 1}{NNMax - 1} \tag{7}$$

3.1.2. Extended Genetic Programming: EGP

The Extended Genetic Programming which is an extended version of the standard genetic programming is formed by three mainly operators.

3.1.2.1. Selection operator. Its purpose mainly is to select two parents from the population in order to procreate a new child by crossover or mutation operator. In this study firstly a truncation selection is used by ranking all individuals according to their fitness. After that, a threshold T ($T \in [0, 1]$) is applied such that the $(1-T)$% best individuals are selected to survive to the next generation and the remaining individuals are removed and replaced with new ones. Secondly, the binary tournament selection is also used to select individuals for the crossover or mutation operator. For each individual, two opponents are randomly chosen from all the parents and offspring. For each comparison, if the fitness of individual exceeds that of the opponent, it receives a selection.

3.1.2.2. Crossover operator. Its principle is to take randomly selected sub-trees in the individuals and to select randomly one non-leaf node in the hidden layer for each chromosome, and then swapping the selected sub-trees. An example of crossover operator is shown in Fig. 3.

3.1.2.3. Mutation operators. Four different mutation operators were used to generate offspring from the parents:

1. *Changing one leaf node*: select one leaf node randomly in the FBBFNT and replace it with another leaf node (Fig. 4(b));
2. *Changing all the leaf nodes:* select each leaf nodes in the FBBFNT and replace it with another leaf node (Fig. 4(c));
3. *Growing:* select a random leaf node in hidden layer of the FBBFNT and replace it with a randomly generated sub-tree (Fig. 4(d));
4. *Pruning:* randomly select a Beta operator node in the FBBFNT and replace it with a random leaf node (Fig. 4(e)).

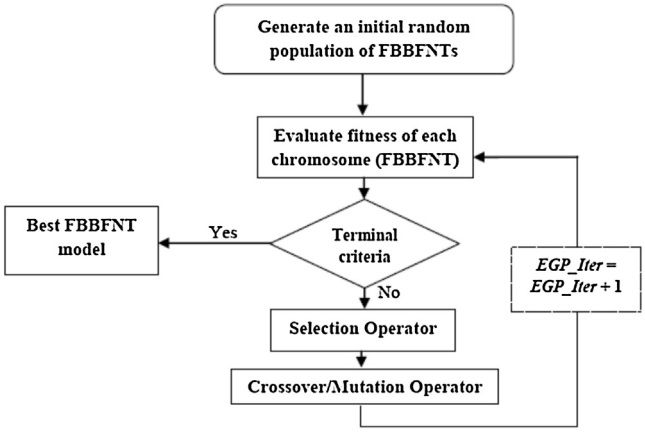

Fig. 5. Flowchart of the Extended Genetic Programming algorithm.

The EGP mutation operators were applied to each parent according to the method of Chellapilla [34] as following:

(a) Define a number M which represents a sample from a Poisson random variable.
(b) Select randomly M mutation operators from above four mutation operator set.
(c) Apply these M mutation operators in sequence one after the other to the parent to create the offspring.

The new generation which modify some individuals of the population, is evaluated using the structure fitness function. If this generation is better than the previous one, the population is updated. This process can be repeated until the terminal criteria are reached; i.e., a better structure is found or a maximum number of EGP iterations is attained. A schematical overview of EGP is given in Fig. 5.

3.2. Parameter optimization

After obtaining the optimal or near-optimal structure of the FBBFNT model using the Extended Genetic Programming (EGP), the hybrid system move to improve the performance of the parameters (weights and Beta parameters) encoded in this best found structure.

To find optimal or near-optimal FBBFNT parameters, the first step is to decide the corresponding representation. In our case, we used the real-number representation for mapping the FBBFNT parameters into a matrix-representation. Indeed, the real-number representation is preferred compared to the binary-string representation especially when the number of connection weights is very large and the desired error is very small.

The second step is the choice of the learning method. In this context several researches have been achieved and most of all proved that Evolutionary Computation (EC) gives better results compared to classical methods. For these reasons, we adopted, as learning algorithms, existing or hybrid evolutionary computation algorithms. The parameter fitness function used to evaluate individuals will be presented in the next section.

3.2.1. Parameter fitness function

To find an optimal FBBFNT, the Root Mean Squared Error (RMSE) is employed as a fitness function:

$$\mathrm{Fit}_{\mathrm{par}}(i) = \sqrt{\frac{1}{P}\sum_{j=1}^{P}\left(y_t^j - y_{\mathrm{out}}^j\right)^2} \tag{8}$$

where P is the total number of samples, y_t^j and y_{out}^j are the desired output and the FBBFNT model output of jth sample. $\mathrm{Fit}_{\mathrm{par}}(i)$ denotes the fitness value of ith individual.

3.2.2. Artificial Bee Colony algorithm

Artificial Bee Colony algorithm (ABC) is a swarm intelligent optimization algorithm based on the metaphor foraging behavior of honey bee swarm, proposed by Karaboga in 2005 [8]. This model that leads to the emergence of collective intelligence of honey-bee swarms consists of three essential bee kinds: employed bees, onlooker bees and scout bees. Employed bees are responsible for exploiting a specific food source and sharing the information of the nectar amount with onlooker bees waiting in the hive.

Onlooker bees watch the dances of employed bees and choose food sources depending on the quantity information. Scout bees search the space randomly to find a new nectar source. The number of employed bees is equal to the number of nectar sources. The position of a nectar source represents a possible solution to the optimization problem, and the nectar amount of a food source corresponds to the fitness of the associated solution.

Denote the food sources number as NP. In our case, the position of the ith food source as a chromosome x_i ($i = 1,\ldots, NP$) which is $NParam \times NN$ matrix; with $NParam$ is the number of parameters (Beta parameters and connected weights) and NN is the number of FBBFNT nodes. A candidate food position from the old one in memory can be generated as:

$$v_i = x_i + \varphi_1 (x_i - x_k) \tag{9}$$

where $k \in \{1, \ldots, NP\}$ with $k \neq i$ and φ_1 are randomly distributed number in [0,1]. Then, v_i is compared with x_i, and the better one should be remained.

An onlooker bee chooses a food source depending on the probability value associated with that food source, which is given by:

$$p_i = \frac{\mathrm{Fit}_{\mathrm{par}}(i)}{\sum_{j=1}^{\mathrm{NP}} \mathrm{Fit}_{\mathrm{par}}(j)} \tag{10}$$

where $\mathrm{Fit}_{\mathrm{par}}(i)$ is the fitness value of the ith food source.

If the abandoned food source is x_i, the scout bee discovers a new food source according to:

$$x_i = a_j + rand_i \left(b_j - a_j\right) \tag{11}$$

where $[a_j, b_j]$ is the search space of each parameter, $j \in \{1, \ldots, NParam\}$.

3.2.3. Opposite-based Particle Swarm Optimization (OPSO)

PSO was proposed by Kennedy and Eberhart [36] and is inspired by the swarming behavior of animals. The initial population of particles is randomly generated. Each particle has a position vector denoted by x_i. A swarm of particles 'flies' through the search space; with the velocity vector v_i of each particle. At each time step, a fitness function is calculated by using x_i.

Each particle records its best position corresponding to the best fitness, which has done so far, in a vector p_i. Moreover, the best position among all the particles obtained in a certain neighborhood of a particle is recorded in a vector p_g.

The use of heuristic operators or the update of the position in the PSO algorithm can mislead the finding of best particle by heading it towards a bad solution. Consequently, the convergence to the desired value becomes very expensive. To avoid these drawbacks, research dichotomy is adapted to improve the generalization performance and accelerate the convergence rate. Thus the reduction of the convergence time of the beta neural system is done by dividing the search space in two subspaces and a concept of the opposite number can be used to look for the guess solution between the two search subspaces. This concept can be integrated in the basic PSO

algorithm and form a new algorithm called Opposite-based Particle Swarm Optimization (OPSO) [19,57,58].

The OPSO steps are described as follows:

Step 0 (Initialization): At iteration $t=0$, the initial positions $x_i(0)$ ($i=1, \ldots, NP$) which are $NParam \times NN$ arrays, are generated uniformly distributed randomly as Eq. (11);

Generate the opposite population as follows:

$$\bar{x}_i(0) = \begin{cases} \alpha_i(x_i(0) + \dfrac{a_j+b_j}{2}), \text{if} x_i(0) < \dfrac{a_j+b_j}{2} \\ \alpha_i(x_i(0) - \dfrac{a_j+b_j}{2}), \text{if} x_i(0) > \dfrac{a_j+b_j}{2} \end{cases}, \alpha_i \in]0,\ 1[\quad (12)$$

The initial velocities, $v_i(0)$, $i=1,\ldots, NP$, of all particles are randomly generated.

Step 1 (Particle evaluation): Evaluate the performance of each particle in the population using a fitness function (Section 3.2.1).

Step 2 (Velocity update): At iteration t, the velocity v_i of each particle i is updated using $p_i(t)$ and $p_g(t)$.

Here, the mutation operator is adopted according to:

$$v_i(t+1) = \Psi(t)v_i(t) + c_1\varphi_1(p_i(t) - x_i(t)) + c_2\varphi_2\left(p_g(t) - x_i(t)\right) \quad (13)$$

where c_1, c_2 (acceleration) and Ψ (inertia) are positive constant and φ_1 and φ_2 are randomly distributed number in [0,1]. The velocity v_i is limited in $[-v_{max}, +v_{max}]$.

Step 3 (Position update): Depending on their velocities, each particle changes its position and its opposite- position according to:

$$x_i(t+1) = x_i(t) + (1-\Psi(t))v_i(t+1) \quad (14)$$

$$\bar{x}_i(t+1) = \bar{x}_i(t) + (1-\Psi(t))v_i(t+1) \quad (15)$$

Step 4 (p_i and p_g update): After traveling the whole population and changing the individual positions, the values of $p_i(t)$ and $p_g(t)$ obtained so far are updated.

3.2.4. Artificial Bee Colony algorithm modified by Opposite-based Particle Swarm Optimization positions (ABC-OPSOP)

In ABC, the best solution found so far is not always held in the population unlike PSO and OPSO. Since, it might be replaced, at some limit, with a randomly produced solution by a scout. Therefore, it might not contribute to the creation of trial solutions. Moreover, ABC algorithm suffers from slower convergence speed for some unimodal problems and easily gets trapped in the local optima for some complex multimodal problems.

For these reasons, we proposed to hybridize the classical ABC with the Opposite-based Particle Swarm Optimization (OPSO). This hybridization is essentially based on replacing the solution search equation of ABC algorithm (Eq. (9)) by the position equations of the OPSO algorithm (Eqs. (14) and (15)). The solution procedure of ABC-OPSOP is as follows:

ABC-OPSOP algorithm

1: Initialize the food sources and evaluate the population and the opposite population (step 0 and step 1 of OPSO algorithm), $trail_i = 0$, ($i = 1, \ldots, NP$), $cycles$=1, $NFEs$= number of Function Evaluations of EGP.
2: Repeat
3: Produce the solution x_i according to (14) and the opposite solution $\bar{x}_i$ according to (15) for employed bees and evaluate them.
4: $NFEs = NFEs + 2$
5: Select the better solution between these two solutions using the fitness value.
6: If solution does not improve $trail_i = trail_i + 1$, otherwise $trail_i = 0$.
7: Calculate the probability according to (10) and apply roulette wheel selection scheme to choose a food source for onlooker bees.
8: Produce the solution x_i according to (14) and the opposite solution $\bar{x}_i$ according to (15) for onlooker bees and evaluate them.
9: $NFEs = NFEs + 2$
10: Select the better solution between these two solutions using the fitness value.
11: If solution does not improve $trail_i = trail_i + 1$, otherwise $trail_i = 0$.
12: Update $p_i(cycles)$ and $p_g(cycles)$ values.
13: If ($max(trail_i) > limit$)
replace this food source with a new randomly produced food source by (11) and evaluate it
$NFEs = NFEs + 1$
14: Memorize the best solution achieved so far.
15: $cycles = cycles + 1$.
16: Until (*Terminal criteria satisfied*)

3.3. Learning algorithm for FBBFNT model

To find an optimal or near-optimal FBBFNT model, structure and parameters optimization are used alternately. Combining the EGP and ABC-OPSOP algorithms, a hybrid algorithm for evolving FBBFNT model is described as follows and is depicted in Fig. 6. The FBBFNT model is also freely available as Matlab code [62].

(a) Create randomly an initial population (FBBFNT trees and its corresponding parameters);

$G=0$, where G is the generation number of the learning algorithm;

global iterations = 0;

NFEs = 0;

(b) Structure optimization is accomplished by the Extended Genetic Programming (EGP) as described in Section 3.1;

(c) If the terminal criteria are reached, then go to step (d),

global iterations = *global iterations* + *EGP iterations*;

NFEs = *NFEs* + *number of Function Evaluations of EGP*;

otherwise go to step (b);

(d) Parameter optimization is achieved by the ABC-OPSOP algorithm. The architecture of FBBFNT model is fixed, and it is the best tree found by the structure search. The parameters (weights and flexible Beta function parameters) encoded in the best tree formulate a food source;

(e) If the terminal criteria is satisfied, or no better parameter vector is found for a fixed time then go to step (f),

global iterations = *global iterations* + *cycles*;

NFEs = *NFEs* + *number of Function Evaluations of ABC-OPSOP*;

otherwise go to step (d);

(f) If satisfactory solution is found or a maximum global iteration number is reached, then the algorithm is stopped; otherwise let ($G = G + 1$) and go to step (b);

4. Experimental results

Nonlinear problems are of interest to scientists and engineers because most physical systems are inherently nonlinear in nature.

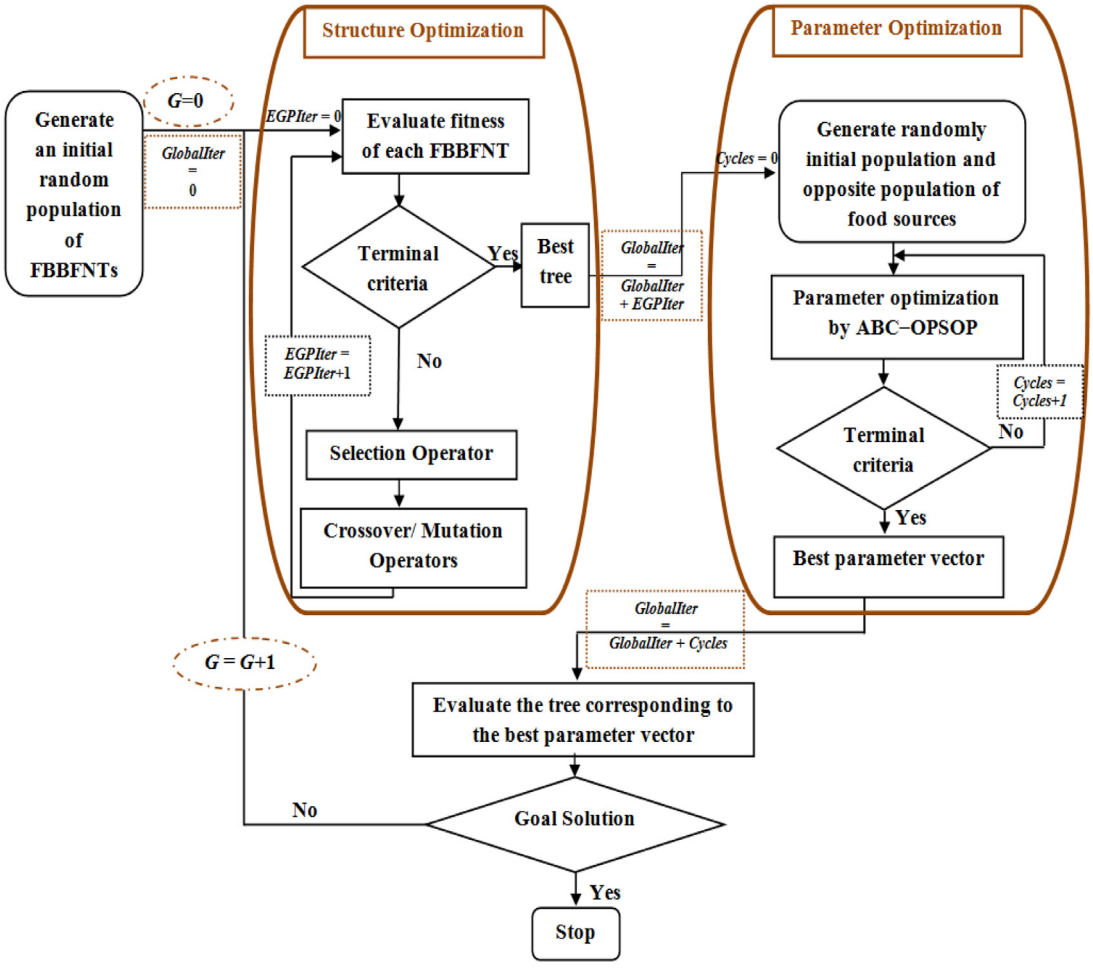

Fig. 6. The Hybrid Learning Algorithm for FBBFNT model.

In addition, nonlinear equations are difficult to solve and give rise to important subsystems such as chaotic, identification and control systems.

Chaotic systems which have held researchers' interest in the past decades, is one of the most used nonlinear systems for the artificial neural network test. As well-known benchmarks of chaotic systems, we can note the time-series problems. These problems are based on the following principle: knowing the solution at recent past values of the observed series, we can predict the solution at a point in the future. According to the literature, the widely used examples are the Mackey-Glass, Jenkins-Box, sunspot number, and Lorenz chaotic time series.

Another class of nonlinear systems that is broadly used in the literature consisted of the nonlinear plant identification problems. In fact, the identification system is model of a system which can be represented by a mathematical model based in the physical laws that govern the problem or by using the experimental data measured in the own system [59].

In the identification context, the methods do not need previous knowledge of the system, they are so known as black-box identification process [60].

Therefore, we apply in this section, the proposed FBBFNT model to predict and identify these nonlinear systems.

After a series of tuning experiments of the system settings, the chosen setting values for the structure optimization algorithm and the parameter optimization algorithm are as listed in Table 1. These FBBFNT settings are the same for all problems. For all examples the illustrated results are obtained by averaging the results in 30 runs. To measure the time complexity of the hybrid algorithm, we adopted the number of Function Evaluations: NFEs.

We used Matlab environment using a PC with Intel(R) Core(TM) i3-2370 M CPU 2.4 GHz and 4GB RAM.

Several tests will be used to compare the effectiveness of all competitor algorithms with respect to three measures of performance which are:

Table 1
FBBFNT parameters.

	Parameter	Initial value
EGP	Population size	50
	Crossover probability	0.3
	Mutation probability	0.6
	Generation gap	0.9
	Maximum generation number	4000
PSO	c_1	0.2
	c_2	0.2
OPSO	c_1	0.8
	c_2	0.8
ABC	ABC_LimitTrial	50
ABC-OPSOP	Population size	50
	Maximum iteration number	4000
	c_1	0.8
	c_2	0.8
	ABC_LimitTrial	50
Common settings	Population size	50
	Maximum iteration number	4000

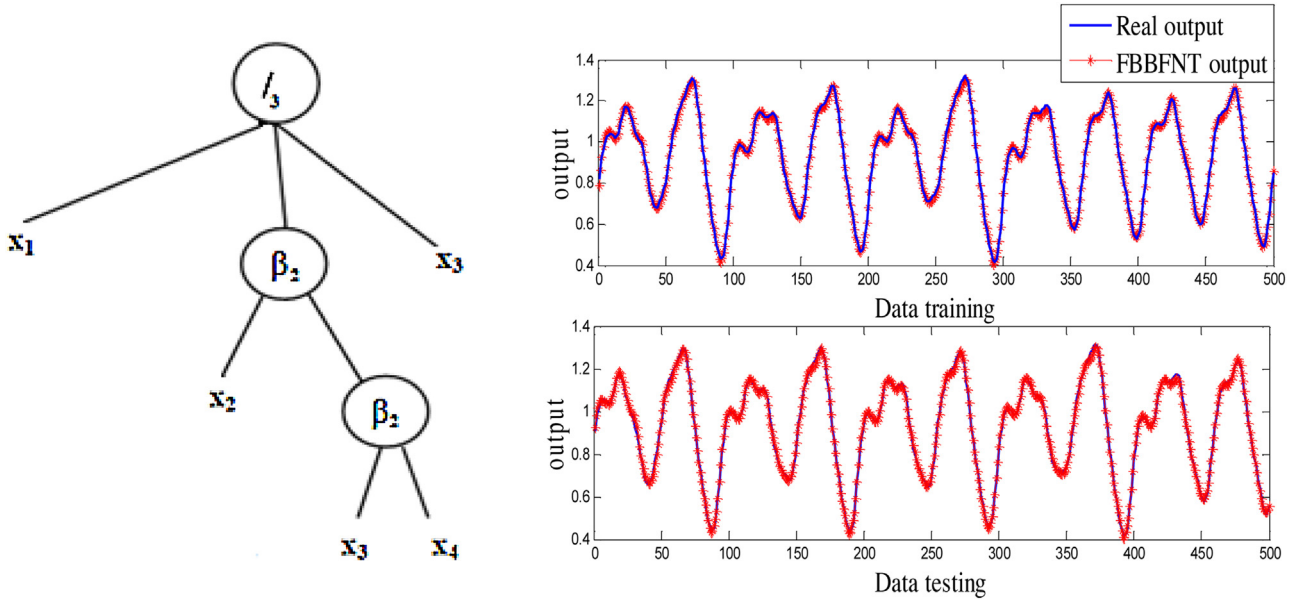

Fig. 7. The evolved FBBFNT (left), and the actual time series data and the output of the FBBFNT model for training and test samples (right) to forecast Mackey–Glass data.

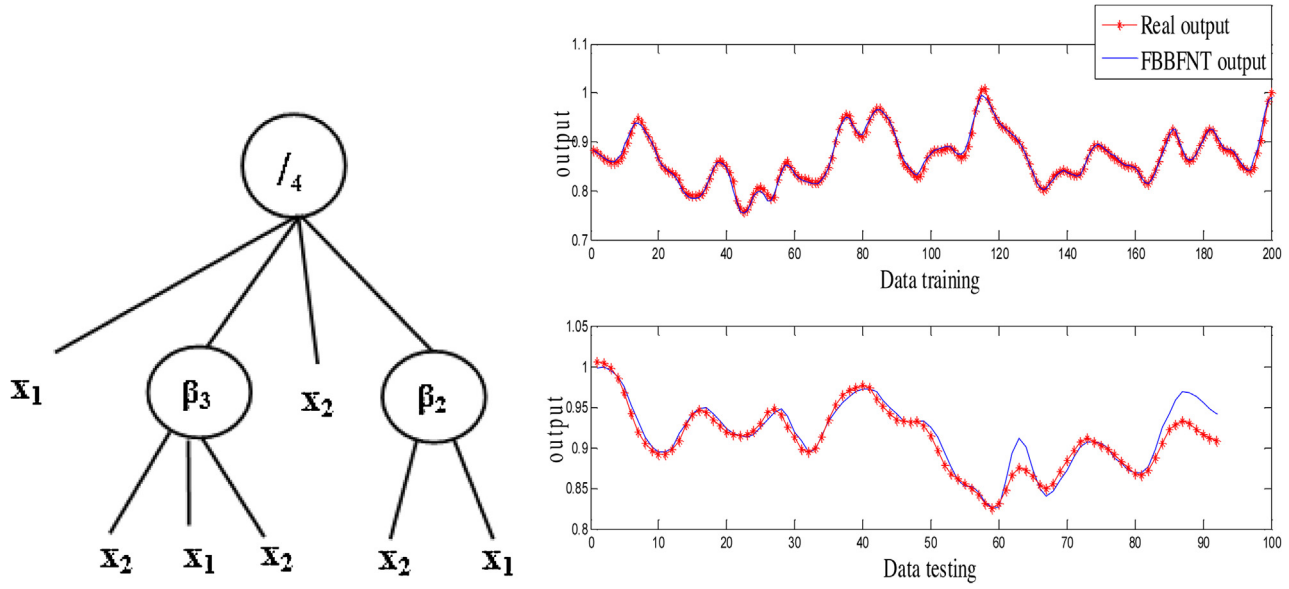

Fig. 8. The evolved FBBFNT (left), and the actual time series data and the output of the FBBFNT model for training and test samples (right) to predict the Jenkins–Box time-series ($y(t-1), u(t-4)$).

- Solution quality or accuracy: by calculating the RMSE values,
- Convergence speed: through the Global Iteration Number (GIN),
- Complexity: FBBFNT complexity (number of function hidden nodes) and hybrid algorithm complexity (number of Function Evaluations: NFEs).

In addition, we will consider other performance characteristics of the model, namely, the memory footprint of the source code and the input data in *kilo-bytes* as well as the execution time in *seconds*.

The developed software was also tested on other datasets and applications but due to space limitations, we are not able to include all the results in this manuscript.

4.1. Example 1: Mackey–Glass time series prediction

A time-series prediction problem can be constructed based on the Mackey–Glass [37] differential equation:

$$\frac{d(x(t))}{dt} = \frac{ax(t-\tau)}{1+x^c(t-\tau)} - bx(t) \tag{16}$$

The setting of the experiment varies from one work to another. In this work, the same parameters of Refs. [18,19], and [22] namely $a=0.2, b=0.1, c=10$ and $\tau \geq 17$, were adopted, since the results from these works will be used for comparison. As in the studies mentioned above, the task of the neural network is to predict the value of the time series at point $x(t+6)$, with using the inputs variables $x(t), x(t-6), x(t-12)$ and $(t-18)$. 1000 sample points are used in our study. The first 500 data pairs of the series are used as train-

Table 2
Comparison (RMSE and GIN) between EGP & parameter optimization algorithms for Mackey–Glass problem (NFEs = 6,000,000).

Hybrid algorithm	RMSE training	RMSE testing	GIN
FBBFNT_EGP&PSO	5.3000e-03	5.4000e-03	123,336
FBBFNT_EGP&OPSO	5.5985e-04	5.8488e-04	63,608
FBBFNT_EGP&ABC	8.2018e-04	8.8512e-04	70,911
FBBFNT_EGP&ABC-OPSOP	1.3529e-10	1.3534e-10	40,140

Table 3
Comparison (NFEs and GIN) between EGP & parameter optimization algorithms for Mackey–Glass problem (RMSE of training = 1.00e-06, maximum GIN = 40,000).

Hybrid algorithm	NFEs	GIN	RMSE testing
FBBFNT_EGP&PSO	1,962,284	40,245	5.2000e-03
FBBFNT_EGP&OPSO	3,954,397	40,001	2.9000e-03
FBBFNT_EGP&ABC	4,101,431	40,785	3.6381e-03
FBBFNT_EGP&ABC-OPSOP	621,556	6231	6.8205e-06

Table 5
Comparison (RMSE and GIN) between EGP & parameter optimization algorithms for Jenkins–Box problem (NFEs = 6,000,000).

Hybrid algorithm	RMSE training	RMSE testing	GIN
FBBFNT_EGP&PSO	0.01735	0.01814	42,163
FBBFNT_EGP&OPSO	0.01245	0.01216	33,840
FBBFNT_EGP&ABC	0.01471	0.01669	39,012
FBBFNT_EGP&ABC-OPSOP	0.01102	0.01202	20,840

Table 6
Comparison (NFEs and GIN) between EGP & parameter optimization algorithms for Jenkins–Box problem (RMSE of training = 0.0125, maximum GIN = 40,000).

Hybrid algorithm	NFEs	GIN	RMSE testing
FBBFNT_EGP&PSO	2,101,145	42,754	0.02191
FBBFNT_EGP&OPSO	1,381,078	16,200	0.01216
FBBFNT_EGP&ABC	1,901,310	28,001	0.01962
FBBFNT_EGP&ABC-OPSOP	1,117,608	8754	0.01116

ing data, while the remaining 500 are used to validate the model identified.

As we have seen in previous section that evolving FBBFNT depends essentially on two issues which are the FBBFNT structure's creation and optimization; and FBBFNT parameter's adjustment. For the structure optimization we apply the EGP and for the FBBFNT parameter's optimization, we essentially apply, in this study, three evolutionary computation algorithms: PSO [29], OPSO [42], ABC and the proposed ABC-OPSOP. We thus study the response of each algorithm for evolving FBBFNT.

In this case, we fix firstly the number of Function Evaluations (NFEs) to 6,000,000 and then we extract the RMSE values and the Global Iteration Number (GIN) for all parameter optimization algorithms.

Based on the initial parameter optimization setting values (Table 1), Table 2 makes a comparison between the different parameter optimization algorithms combined with the EGP (for the structure optimization).

We can observe from Table 2, that the combination between EGP and ABC-OPSOP for evolving FBBFNT performs better than the other combinations after 6,000,000 NFEs and provides good results by minimizing the prediction error.

Moreover, in order to make the comparison fair enough, we fix the RMSE of training equal to 1.00e-06. We then evaluate the number of Function Evaluations (NFEs) required by each of the FBBFNT parameter's competitor algorithms to reach the prescribed given error.

Table 3 shows, for all algorithms, the NFEs necessary to find the optimum solution previously fixed within a given tolerance or training RMSE. From simulations and results, it is remarkable that FBBFNT_EGP&ABC-OPSOP algorithm reduces generally the NFEs to reach a given training RMSE value as compared with the other algorithms. We note, moreover, that FBBFNT_EGP&PSO, FBBFNT_EGP&OPSO and FBBFNT_EGP&ABC cannot even reach the error fixed in advance with the allowed number of iterations (GIN = 40,000).

To fully evaluate the performance of our new technique and the other models of the literature, a list of experimental tests were performed to predict Mackey–Glass time series.

The used Beta operator sets to create an optimal FBBFNT model is $NS = F \cup T = \{\beta_2, \beta_3, /_3\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i $(i = 1, 2, 3, 4)$ denotes $x(t), x(t-6), x(t-12)$ and $x(t-18)$, respectively. After 20 global generations ($G = 20$), 865,199 global number of function evaluations ($NFEs = 865{,}199$), an optimal FBBFNT model was obtained with RMSE 9.9620e-07. The RMSE value for validation data set is 2.0189e-06. The evolved FBBFNT, and the actual time-series data and the output of FBBFNT model for training ant testing samples are shown in Fig. 7.

In addition, the memory footprint of the source code is of the order of 10,720 Kb, knowing that the memory footprint of the input data is 40 Kb. For the execution time is of the order of 2924 s.

The proposed system is essentially compared with Hierarchical multi-dimensional differential evolution for the design of Beta basis function neural network (HMDDE-BBFNN) [25], the FNT model with Gaussian function as flexible neuron operator [26], the local least-squares support vector machines-based neuro-fuzzy model (LNF) [52] and also with other systems.

The HMDDE-BBFNN approach adopts for parameters: 50 for the population size, 10,000 for a total number of iterations, and 4 for the number of the hidden nodes. Moreover, the parameter settings of the FNT system [26] are 30 to the population size, 135 as generation number, and 4 as hidden function unit number (with two hidden layers). For LNF network, the authors use 6 neurons to generate their model.

A comparison result of different methods for forecasting Mackey–Glass data is shown in Table 4.

Table 4
Comparison between the FBBFNT model and other systems for the Mackey–Glass time series.

Method	Solution quality		Convergence speed (GIN)	Complexity	
	Training error (RMSE)	Testing error (RMSE)		Number of function hidden nodes	Number of function evaluations (NFEs)
PSO-BBFN [23]	–	2.7000e-02	20,000	6 (1 hidden layer)	–
HMDDE-BBFNN [25]	9.4000e-03	1.7000e-02	10,000	4 (1 hidden layer)	–
GA-BBFNN [24]	–	1.3000e-02	–	9 (1 hidden layer)	–
Classical RBF [38]	9.6000e-03	1.1400e-02	–	23 (1 hidden layer)	–
CPSO [39]	1.9900e-02	3.2200e-02	–	4 (1 hidden layer)	–
HCMSPSO [40]	9.5000e-03	2.0800e-02	–	4 (1 hidden layer)	15,000,000
FNT [26]	6.9000e-03	7.1000e-03	–	4 (2 hidden layers)	–
LNF [52]	7.0000e-04	7.9000e-03	–	6 (1 hidden layer)	–
FBBFNT_EGP&ABC-OPSOP	9.9620e-07	2.0189e-06	7421	2 (2 hidden layers)	865,199

Table 7
Comparison of training and testing errors of Box and Jenkins.

Inputs	Training error (RMSE)				Testing error (RMSE)			
	HMDDE [25]	FBBFNT_EGP &OPSO [42]	FBBFNT_ EGP & ABC	FBBFNT_EGP &ABC-OPSOP	HMDDE [25]	FBBFNT_EGP &OPSO [42]	FBBFNT_ EGP& ABC	FBBFNT_EGP &ABC-OPSOP
$y(t-1), u(t-3)$	0.1328	0.0034	0.0299	**0.0023**	0.2276	0.0114	0.0269	**0.0103**
$y(t-3), u(t-4)$	0.0210	0.0060	0.0288	**0.0040**	0.4224	0.0209	0.0267	**0.0116**
$y(t-2), u(t-4)$	0.1365	0.0057	0.0240	**0.0045**	0.3200	0.0165	0.0223	**0.0105**
$y(t-1), u(t-2)$	0.1735	0.0057	0.0307	**0.0047**	0.2334	0.0114	0.0295	**0.0098**
$y(t-1), u(t-4)$	0.2411	0.0047	0.0258	**0.0040**	0.3745	0.0116	0.0245	**0.0109**
$y(t-4), u(t-4)$	0.1594	0.0071	0.0263	**0.0066**	0.4549	0.0237	0.0268	**0.0178**
$y(t-2), u(t-3)$	0.1702	0.0066	0.0251	**0.0056**	0.2700	0.0185	0.0271	**0.0163**
$y(t-1), u(t-1)$	0.1598	0.0124	0.0343	**0.0095**	0.2577	0.0121	0.0427	**0.0120**
$y(t-4), u(t-3)$	0.1921	0.0142	0.0284	**0.0102**	0.6148	0.0289	0.0334	**0.0210**
$y(t-1), u(t-6)$	0.6619	0.0124	0.0315	**0.0113**	0.6638	0.0121	0.0248	**0.0094**
$y(t-3), u(t-3)$	0.1600	0.0106	0.0269	**0.0097**	0.2521	0.0250	0.0321	**0.0198**
$y(t-2), u(t-2)$	0.2773	0.0157	0.0340	**0.0118**	0.1615	0.0256	0.0324	**0.0206**
$y(t-1), u(t-5)$	0.3333	0.0124	0.0288	**0.0091**	0.5595	0.0122	0.0259	**0.0118**
$y(t-4), u(t-5)$	0.0178	0.0116	0.0293	**0.0090**	0.0203	0.0108	0.0256	**0.0097**
$y(t-2), u(t-1)$	0.1960	0.0236	0.0365	**0.0183**	0.2759	0.0226	0.0377	**0.0187**
$y(t-2), u(t-5)$	0.2165	0.0236	0.0286	**0.0117**	0.4021	0.0223	0.0226	**0.0199**
$y(t-3), u(t-5)$	0.1346	0.0137	0.0288	**0.0118**	0.2307	0.0180	0.0256	**0.0112**
$y(t-3), u(t-2)$	0.2128	0.0326	0.0325	**0.0177**	0.2760	0.0307	0.0350	**0.0281**
$y(t-4), u(t-6)$	0.1379	0.0209	0.0305	**0.0169**	0.2635	0.0249	0.0261	**0.0207**
$y(t-2), u(t-6)$	0.2128	0.0236	0.0328	**0.0209**	0.5590	0.0225	0.0323	**0.0206**
$y(t-4), u(t-2)$	0.2152	0.0229	0.0341	**0.0165**	0.2737	0.0335	0.0383	**0.0284**
$y(t-3), u(t-6)$	0.2128	0.0330	0.0367	**0.0208**	0.4027	0.0301	0.0256	**0.0255**
$y(t-3), u(t-1)$	0.2135	0.0326	0.0386	**0.0233**	0.2803	0.0307	0.0378	**0.0268**
$y(t-4), u(t-1)$	0.2695	0.0398	0.0430	**0.0332**	0.2695	0.0363	0.0409	**0.0324**

Table 8
Comparison (RMSE and GIN) between EGP & parameter optimization algorithms for sunspot number problem (NFEs = 600,000).

Hybrid algorithm	RMSE training	RMSE testing 1	RMSE testing 2	GIN
FBBFNT_EGP&PSO	1.0138e-07	2.018e-07	2.113e-07	10,163
FBBFNT_EGP&OPSO	1.2021e-09	2.077e-09	3.102e-09	15,840
FBBFNT_EGP&ABC	1.0010e-08	1.067e-08	1.099e-08	18,481
FBBFNT_EGP&ABC-OPSOP	1.1981e-11	2.018e-11	2.113e-11	9432

Table 9
Comparison (NFEs and GIN) between EGP & parameter optimization algorithms for sunspot number problem (RMSE of training = 1.00e-07, maximum GIN = 40,000).

Hybrid algorithm	NFEs	GIN	RMSE testing 1	RMSE testing 2
FBBFNT_EGP&PSO	617,108	18,918	1.6078e-07	1.3128e-07
FBBFNT_EGP&OPSO	784,763	17,281	7.3589e-08	6.0259e-08
FBBFNT_EGP&ABC	801,904	18,002	1.2157e-07	1.5526e-03
FBBFNT_EGP&ABC-OPSOP	271,521	4502	3.0733e-08	2.9634e-08

It is clear from the results of our model and the errors provided by the other methods (Table 4), that our system gives the best prediction accuracy for the Mackey–Glass time-series. Our method can also minimize the number of hidden function units to two neurons in two hidden layers for EGP algorithm.

This result justifies more that we have seen in Section 3.2.4. In fact, the hybridization between OPSO and ABC improves the behavior of both algorithms.

For better explain these results theoretically, we are based, in the choice of metaheuristic algorithms for FBBFNT parameter optimization, on two essential phenomena: exploration and exploitation [55,56]. The exploration means the search for new solutions in the search space and the exploitation means the use and the improvement of existing solutions.

Indeed, PSO is known in the literature as a strong algorithm in the exploitation seen that it is based on a guided solution through the local and global optima. But it would be sometimes weak in exploration in the search space by getting stuck in a local optimum. For OPSO, it improves slightly the exploration compared to PSO but it can also get stuck in a local optimum because it doesn't use new solutions of the search space.

On the other hand, ABC rather promotes exploration as exploitation compared to other algorithms in the literature by seeking new solutions of the search space after a maximum number of trials.

For these reasons we hybridized both algorithms into a new algorithm (ABC-OPSOP) to benefit from their advantages. This strategy is also justified by the experimental results found in this section and the following sections.

4.2. Example 2: Box and Jenkins' gas furnace problem

The gas furnace data of Box and Jenkins [41] was saved from a combustion process of a methane–air mixture. It is frequently used as a benchmark example for testing prediction algorithms. The data set forms of 296 pairs of input–output measurements. The input $u(t)$ is the gas flow into the furnace and the output $y(t)$ is the CO_2 concentration in outlet gas.

The inputs for constructing FBBFNT model are $y(t-1)$, $u(t-4)$, and the output is $y(t)$. In this work, 200 data samples are used for training and the remaining data samples are used for testing the performance of the evolved model.

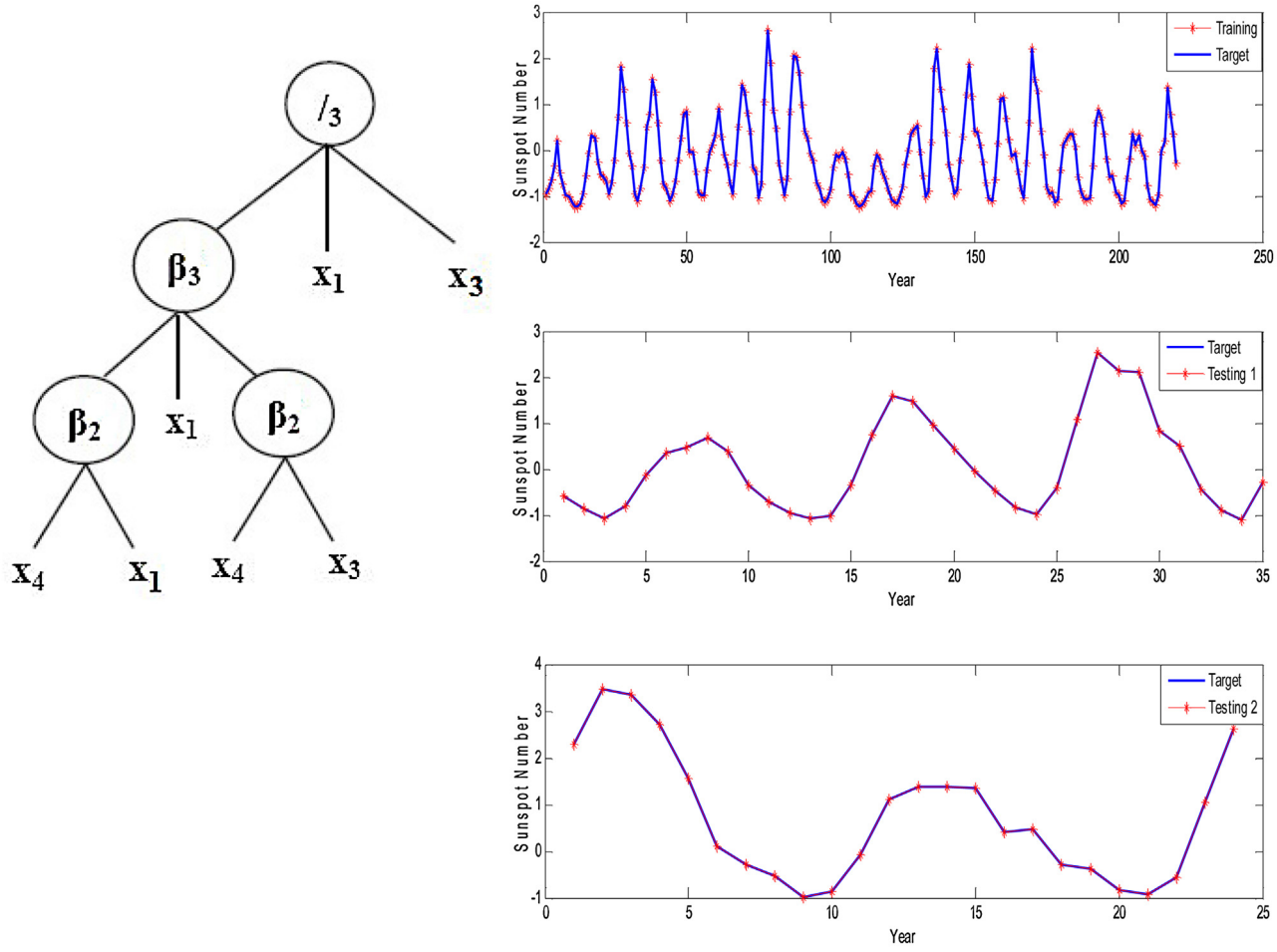

Fig. 9. The evolved FBBFNT (left), and the actual time series data and the output of the FBBFNT model for training, test 1 and test 2 to predict the sunspot number time-series.

Table 10
Comparison between the FBBFNT model and other systems for the sunspot number time-series.

Method	Solution quality			GIN	Complexity	
	RMSE training	RMSE testing 1	RMSE testing 2		Number of function hidden nodes	NFEs
Recurrent net [44]	2.6790e-01	3.1510e-01	9.0050e-01	–	2 (1 hidden layer)	–
RFNN [45]	–	2.7490e-01	6.2490e-01	–	7 (1 hidden layer)	–
FENN [47]	–	–	5.7850e-01	–	–	–
FWNN-S [48]	2.5270e-01	3.3410e-01	5.2990e-01	200	16 rules	–
FWNN-R [48]	2.3830e-01	3.3500e-01	6.8850e-01	200	16 rules	–
FWNN-M [48]	2.4300e-01	3.1520e-01	6.0800e-01	200	16 rules	–
ABC_BBFNN [46]	1.2000e-03	1.8000e-03	4.4000e-03	–	–	–
LNF[52]	1.8880e-03	2.5370e-03	3.8080e-03	–	7 (1 hidden layer)	–
FBBFNT_EGP&ABC-OPSOP	3.1035e-08	7.2848e-07	8.0029e-07	3821	3 (2 hidden layers)	631,075

Table 11
Comparison (RMSE and GIN) between EGP & parameter optimization algorithms for Lorenz problem (NFEs = 200,000).

Hybrid algorithm	RMSE training	RMSE testing	GIN
FBBFNT_EGP&PSO	3.3552e-03	8.3352e-03	3386
FBBFNT_EGP&OPSO	6.0945e-04	8.6922e-04	1798
FBBFNT_EGP&ABC	3.7193e-04	5.8124e-04	2931
FBBFNT_EGP&ABC-OPSOP	4.6529e-09	1.3534e-08	885

Table 12
Comparison (NFEs and GIN) between EGP & parameter optimization algorithms for Lorenz problem (RMSE of training = 1.00e-09, maximum GIN = 40,000).

Hybrid algorithm	NFEs	GIN	RMSE testing
FBBFNT_EGP&PSO	607,883	12,734	8.3152e-09
FBBFNT_EGP&OPSO	642,542	8772	5.5060e-09
FBBFNT_EGP&ABC	700,092	10,041	7.9681e-09
FBBFNT_EGP&ABC-OPSOP	266,106	4477	5.0145e-10

In the first case, we fix the number of Function Evaluations (NFEs) to 6,000,000 and we extract the RMSE values and the Global Iteration Number (GIN) for all parameter optimization algorithms. So Table 5 makes a comparison between the different parameter optimization algorithms combined with EGP (for the architecture optimization).

We can notice, from Table 5, that there is not a significant difference between the different algorithm responses. However, the combination between EGP and ABC-OPSOP for evolving FBBFNT rel-

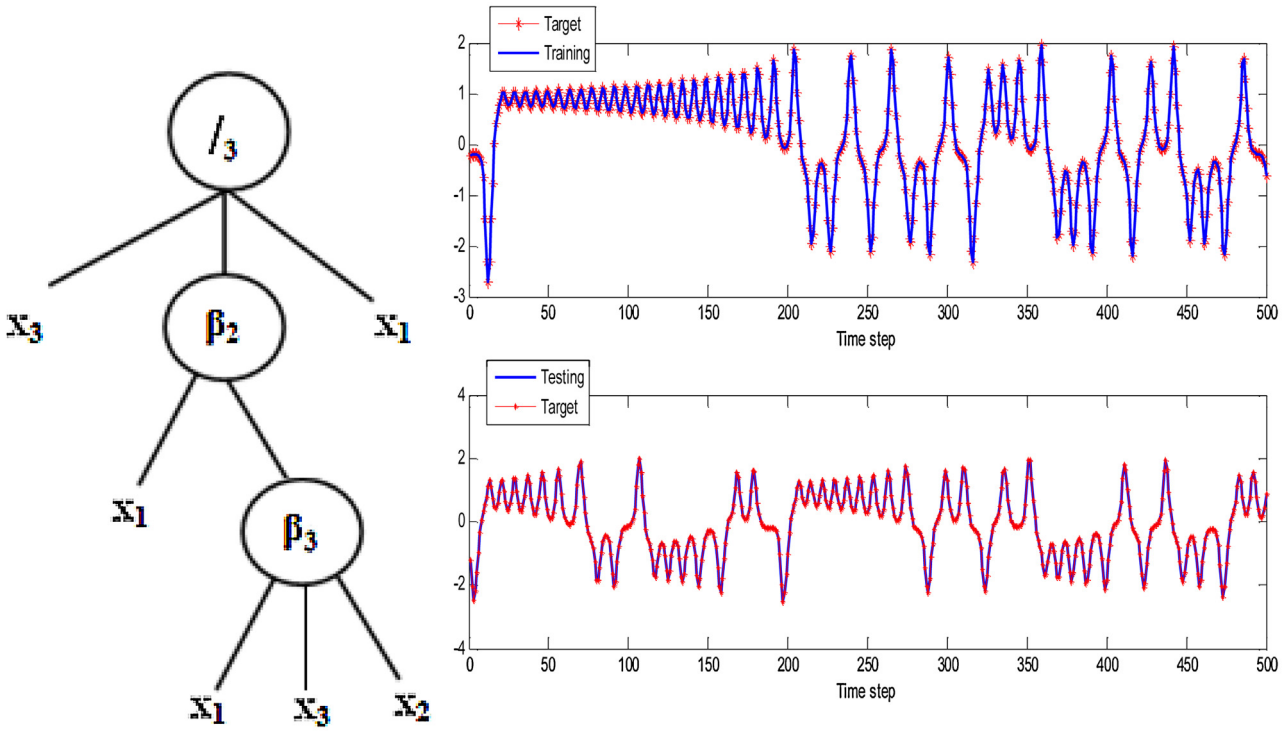

Fig. 10. The evolved FBBFNT, and the actual time series data and the output of the FBBFNT model for training and test examples to predict the Lorenz chaotic time-series.

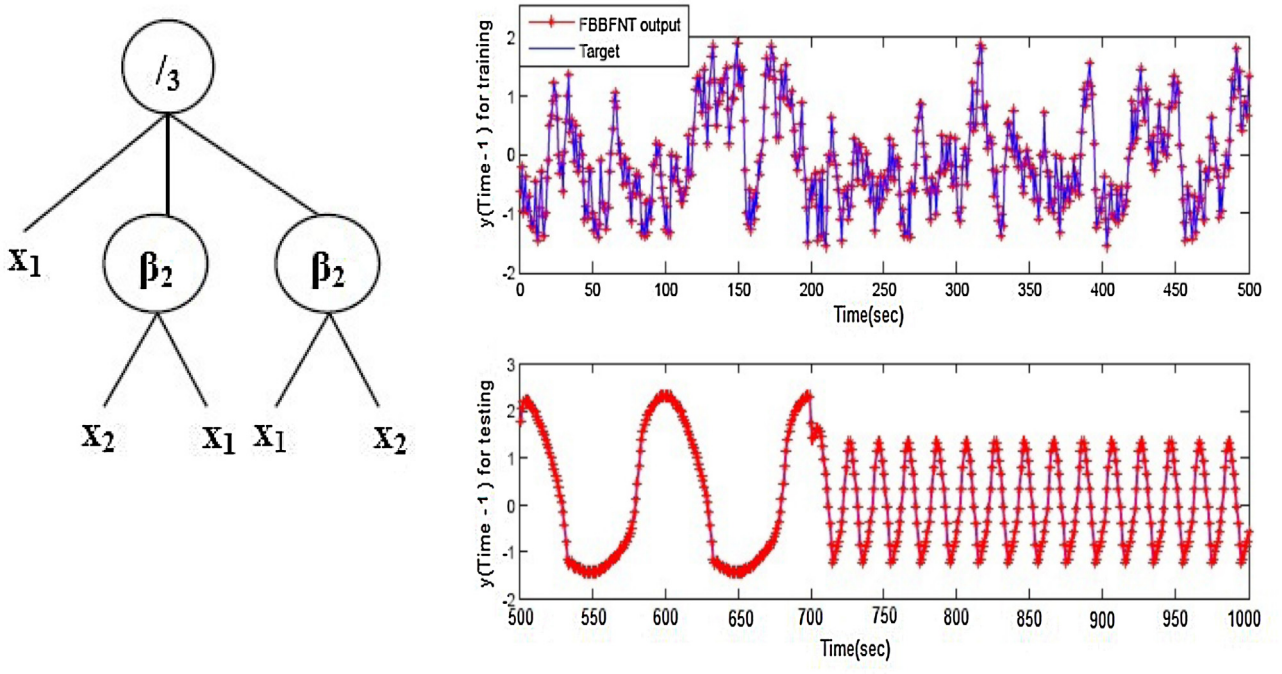

Fig. 11. The evolved FBBFNT, and the actual time series data and the output of the FBBFNT model for training and test examples to identify the nonlinear plant identification.

atively performs better than the other combinations after 6,000,000 NFEs, in terms of accuracy and also in term of global iteration number.

In the second case, we fix the RMSE of training equal to 0.0125. We then extract the number of Function Evaluations NFEs required by each of the FBBFNT parameter's competitor algorithms to reach the prescribed given error.

Table 6 shows, for all algorithms, the NFEs necessary to find the optimum solution previously fixed within a given tolerance or RMSE training. From simulations and results, it is remarkable that FBBFNT_EGP&ABC-OPSOP algorithm reduces significantly the NFEs to reach a given training RMSE value as compared with the other algorithms. We note, moreover, that FBBFNT_EGP&PSO cannot even reach the error fixed in advance at the allowed number of iterations (GIN = 40,000).

To observe the behavior of FBBFNT_EGP&ABC-OPSOP, the used instruction set for creating a FBBFNT model is $S = F \cup T = \{\beta_2, \beta_3, \beta_4, /_4\} \cup \{x_1, x_2\}$, where x_i $(i = 1, 2)$ denotes

Table 13
Comparison between the FBBFNT model and other systems for the Lorenz time-series.

Method	Solution quality		GIN	Complexity	
	Training error (RMSE)	Testing error (RMSE)		Number of function hidden nodes	NFEs
UD-SVM [50]	–	2.9000e-01	–	–	–
MLP-EKF [51]	1.4500e-02	4.0800e-02	1000	37 (3 hidden layers)	–
MLP-BLM [51]	1.7400e-02	3.1400e-02	1000	33 (3 hidden layers)	–
RNN-BPTT [51]	2.2700e-02	4.3600e-02	1000	19 (3 hidden layers)	–
RNN-RTRL [51]	2.2900e-02	4.2000e-02	1000	19 (3 hidden layers)	–
RNN-EKF [51]	1.8200e-02	3.5200e-02	1000	19 (3 hidden layers)	–
RBLM-RNN [51]	1.8200e-02	3.0400e-02	1000	16 (3 hidden layers)	–
LNF [52]	3.9000e-03	8.1000e-03	–	–	–
FBBFNT_EGP&ABC-OPSOP	7.4320e-08	1.0218e-07	3872	2 (2 hidden layers)	204,911

Table 14
Comparison (RMSE and GIN) between EGP & parameter optimization algorithms for nonlinear plan identification (NFEs = 200,000).

Hybrid algorithm	RMSE training	RMSE testing	GIN
FBBFNT_EGP&PSO	2.3452e-04	6.3362e-04	7386
FBBFNT_EGP&OPSO	3.1645e-06	5.6822e-06	10,098
FBBFNT_EGP&ABC	5.0319e-06	8.0077e-06	11,123
FBBFNT_EGP&ABC-OPSOP	4.7509e-08	6.0531e-08	9485

Table 15
Comparison (NFEs and GIN) between EGP & parameter optimization algorithms for nonlinear plan identification (RMSE of training = 1.00e-10, maximum GIN = 40,000).

Hybrid algorithm	NFEs	GIN	RMSE testing
FBBFNT_EGP&PSO	6,073,307	40,230	4.3392e-09
FBBFNT_EGP&OPSO	1,235,763	14,662	3.9081e-10
FBBFNT_EGP&ABC	4,651,800	17,002	5.8644e-10
FBBFNT_EGP&ABC-OPSOP	732,933	31,752	1.4082e-10

$y(t-1)$, $u(t-4)$, respectively. After 12 generations of the learning algorithm ($G = 12$), 450,978 global number of function evaluations ($NFEs = 450{,}978$), the optimal Beta basis function neural tree model was obtained with the RMSE 0.004096. The RMSE value for validation data set is 0.010918. The evolved FBBFNT, and the actual time-series data and the output of FBBFNT model for training and testing samples are shown in Fig. 8. In addition, the memory footprint of the source code is of the order of 2508 Kb, knowing that the memory footprint of the input data is 7.008 Kb. The execution time is of the order of 1021 s.

Furthermore, we have taken two inputs the first one is from furnace output and other is from furnace input. Therefore we have construct 24 models of different input–output and the training and testing performances of these models are given in Table 7. From the simulation results of Table 7, it can be seen that the proposed FBBFNT_EGP&ABC-OPSOP system works well for generating prediction models of Box and Jenkins' gas furnace problem.

4.3. Example 3: prediction of sunspot number time series

This example presents the series of the sunspot annual average numbers which show the yearly average relative number of sunspot observed [43]. The sunspot number time series is considered as a real-world highly-complex and non-stationary time series. It is recorded for the years 1700–1979. The dataset is available at the National Geophysical Data Center website (http://www.ngdc.noaa.gov/stp/solar/ssndata.html).

To make our comparisons meaningful with related works [43–45], the dataset is divided into three parts. The data points between 1700 and 1920 are used for training FBBFNT model. For the test, two sets are used; the first one is from 1921 to 1955 and the second is from 1956 to 1979. The $y(t-4), y(t-3), y(t-2)$ and $y(t-1)$ are used as inputs to the FBBFNT model in order to predict the output $y(t)$.

For FBBFNT's structure optimization, we apply the Extended Genetic Programming and for the parameter's adjustment we test PSO, OPSO, and ABC-OPSOP algorithms.

Firstly, we fix the number of Function Evaluations (NFEs) to 600,000 and we illustrate the RMSE values and the Global Iteration Number (GIN) for all parameter optimization algorithms.

As presented in Table 8, a comparison is performed over a diverse collection of the different combinations between parameter optimization algorithms and EGP.

From Table 8, it is clear that the combination between EGP and ABC-OPSOP for evolving FBBFNT is having less training and testing errors in comparison to the other combinations after 600,000 NFEs.

In order to compare the convergence speed and the evolving FBBFNT complexity of the different algorithms, several numbers of GIN and NFEs were noted. In Table 9, we fixed the error value equal to 1:00e-07, then we evaluate both GIN and NFEs. Table 9 makes a comparison between the different parameter optimization algorithms.

It is clear from Table 9 that ABC-OPSOP algorithm reduces generally the NFEs to reach a given training RMSE value as compared with the other algorithms.

After that, the accuracy and the efficiency of the proposed methodology were demonstrated through several comparisons with other available learning approaches developed in recent years.

The used Beta operator sets to create an optimal FBBFNT_EGP&ABC-OPSOP model is $S = F \cup T = \{\beta_2, \beta_3, /_3\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i ($i = 1, 2, 3, 4$) denotes $y(t-4), y(t-3), y(t-2)$ and $y(t-1)$, respectively.

After 22 generations of the evolution ($G = 22$), 631,075 global number of function evaluations ($NFEs = 631{,}075$), an optimal FBBFNT model was obtained with RMSE 3.1035e-08. The RMSE value for the first data set validation is 7.2848e-07 and for the sec-

Table 16
Comparison between the FBBFNT model and other systems for the nonlinear plant identification.

Method	Solution quality		GIN	Complexity	
	Training error (RMSE)	Testing error (RMSE)		Number of function hidden nodes	NFEs
ODE [61]	1.9000e-02	1.1370e-01	1000	11 (1 hidden layer)	–
HMDDE [25]	1.9000e-02	1.1000e-01	10000	3 (1 hidden layer)	–
FBBFNT_EGP&ABC-OPSOP	1.8961e-10	2.1203e-10	15067	2 (1 hidden layer)	270,311

ond data set validation is 8.0029e-07. The evolved FBBFNT, and the actual time-series data and the output of FBBFNT model for training and the two test cases are shown in Fig. 9.

Table 10 illustrates the comparison of the proposed algorithm with other models according to the training and testing errors. As evident from Table 10, FBBFNT_EGP&ABC-OPSOP shows again the efficiencies considering the compromise between solution quality and complexity for the sunspot number time series.

4.4. Example 4: Lorenz chaotic time series prediction

The Lorenz system [49] is a model of fluid motion between a hot surface and a cool surface. It is described by the following nonlinear ordinary differential equations:

$$\begin{pmatrix} \dot{x} & = \sigma(y-x) \\ \dot{y} & = -y - xz + rx \\ \dot{z} & = xy - bz \end{pmatrix} \tag{17}$$

In this example, the x-component in the Lorenz equations is used as the time series. The data that describe the Lorenz attractor, were generated by solving the system of differential equations, with $\sigma = 10$, $r = 28$ and $b = 8/3$. The data were used as inputs to the neural networks. The prediction is based on four past values $(x(t-4), x(t-3), x(t-2), x(t-1))$ and thus the output pattern is $x(t) = f(x(t-4), x(t-3), x(t-2), x(t-1))$.

To study the Lorenz chaotic problem, we choose EGP for the FBBFNT's structure optimization and we vary the algorithms of FBBFNT's parameter optimization.

In the first case, we fix the number of Function Evaluations (NFEs) to 200,000 and we extract the RMSE values and the Global Iteration Number (GIN) for all parameter optimization algorithms.

Based on the initial parameter optimization setting values (Table 1), Table 11 makes a comparison between the different proposed parameter optimization algorithms combined with EGP.

We can observe from Table 11, that the combination between EGP and ABC-OPSOP for evolving FBBFNT performs better than the other combinations after 200,000 NFEs and provides good results by generating high accuracy system.

In the second case, we fix the RMSE of training to 1.00e-09. We then extract the number of Function Evaluations NFEs required by each of the FBBFNT parameter's competitor algorithms to reach the prescribed given error. Table 12 illustrates a comparison between the different parameter optimization algorithms combined with EGP.

It is remarkable from Table 12 that FBBFNT_EGP&BC-OPSOP algorithm reduces generally the NFEs to reach a given training RMSE value as compared with the other algorithms.

To fully evaluate the performance of our new technique and the other models of the literature, a list of experimental tests were performed for predicting Lorenz time series.

The used Beta operator sets to create an optimal FBBFNT model is $S = F \cup T = \{\beta_2, \beta_3, /_3\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i $(i = 1, 2, 3, 4)$ denotes $x(t-4), x(t-3), x(t-2)$ and $x(t-1)$, respectively.

After 19 generations of the evolution ($G = 19$), 204,911 global number of function evaluations (*NFEs* = 204,911, an optimal FBBFNT model was obtained with RMSE 7.4320e-08. The RMSE value for validation data set is 1.0218e-07. In addition, the memory footprint of the source code is of the order of 2480 Kb, knowing that the memory footprint of the input data is 40 Kb. The execution time is of the order of 979 s. The evolved FBBFNT, and the actual time-series data and the output of FBBFNT model for training and testing examples are shown in Fig. 10.

Table 13 illustrates the comparison of the proposed algorithm with other models according to the training and testing errors. This table shows the performance of our approach for Lorenz chaotic time series comparing with the other model.

4.5. Example 5: nonlinear plant identification

In this example, the nonlinear system [41] to be identified is expressed by:

$$y_P(t+1) = \frac{y_P(t)[y_P(t-1)+2][y_P(t)+2.5]}{8.5 + [y_P(t)]^2 + [y_P(t-1)]^2} + u(t) \tag{18}$$

where $yp(t)$ is the output of the system at the tth time step and $u(t)$ is the plant input which is uniformly bounded in the region $[-2,2]$. The adopted identification model is as the following form:

$$y_{pi}(t+1) = f\left(y_p(t), y_p(t-1)\right) + u(t) \tag{19}$$

where $f\left(y_p(t), y_p(t-1)\right)$ is the nonlinear function of $y_p(t)$ and $y_p(t-1)$ that will be input of our model and related works; and $y_{pi}(t+1)$ will be the output from the neural models.

The objective is to train the artificial neural networks used to identify the above system such that when an input $u(t)$ is presented to the network and to the nonlinear identification system, the network outputs $y_{pi}(t+1)$ and the actual system output $y_p(t+1)$ should be as close as possible i.e., the identified system output should follow the actual system output.

In this example, 500 data samples are used for training and 500 data samples are used for testing the performance of the evolved model. After the training is over, the identifier's prediction ability has been tested for the input calculated as following:

$$u(t) = \begin{cases} 2\cos\left(\dfrac{2\pi t}{100}\right) & \text{if } t \le 200 \\ 1.2\sin\left(\dfrac{2\pi t}{20}\right) & \text{if } 200 < t \le 500 \end{cases} \tag{20}$$

In the first case, we fix the number of Function Evaluations (NFEs) to 200,000 and we extract the RMSE values and the Global Iteration Number (GIN) for all parameter optimization algorithms.

Table 14 illustrates a comparison between the different parameter optimization algorithms combined with EGP.

We can observe from Table 14, that the combination between EGP and ABC-OPSOP for evolving FBBFNT performs better than the other combinations after 200,000 NFEs and provides good results by minimizing the prediction error.

In order to compare the convergence speed and the evolving FBBFNT complexity of the different algorithms, several numbers of GIN and NFEs were noted. In Table 15, we fixed the error value equal to 1.00e-10, and then we evaluate both GIN and NFEs. This table makes a comparison between the different adopted parameter optimization algorithms combined with EGP (for the structure optimization).

The FBBFNT_EGP&ABC-OPSOP hybrid system gives relatively the best results for the number of function evaluations as well as for the testing error. This comparison is illustrated in Table 16.

To fully evaluate the performance of our new technique and the other models of the literature, a list of experimental tests were performed for identifying nonlinear plan identification.

From the previous cited combinations, we choose the FBBFNT_EGP&ABC-OPSOP as an example to detail it. In fact, the used Beta basis function sets to create an optimal FBBFNT model with EGP& ABC-OPSOP system is $NS = F \cup T = \{\beta_2, \beta_3, /_3\} \cup \{x_1, x_2\}$, where x_i $(i = 1, 2)$ denotes $y_p(t)$ and$y_p(t-1)$, respectively.

After 10 global generations ($G = 10$), 270,311 global number of function evaluations, and 8009 global iterations of the hybrid learning algorithm, an optimal FBBFNT model was obtained with RMSE 1.8961e-10. The RMSE value for identification data set is 2.1203e-10.

In addition, the memory footprint of the source code is of the order of 97921 Kb, knowing that the memory footprint of the input data is 30 Kb. The execution time is of the order of 3016 s. The evolved FBBFNT, and the nonlinear plant identification and the output of FBBFNT model for training and testing examples are shown in Fig. 11.

According to the available measures of other methods, we can conclude from the comparison table, that the FBBFNT_EGP&ABC-OPSOP model is relatively more efficient considering the compromise between solution quality and complexity, for the four tested time series as well as for the nonlinear plant identification.

5. Conclusion

In this paper, a hybrid learning algorithm is proposed to create and evolve a flexible beta basis function neural tree (FBBFNT) model for the prediction of benchmark problems. As evident from the experiments, the new learning algorithm can successfully optimize the structure and parameters of Beta basis function neural network simultaneously based on the tree representation. In fact, the FBBFNT structure is developed using Extended Genetic Programming (EGP) and the Beta parameters and connected weights are optimized by the Artificial Bee Colony algorithm modified by Opposite-based Particle Swarm Optimization positions (ABC-OPSOP).

The results show that the FBBFNT_EGP&ABC-OPSOP model can effectively predict time-series problems (Mackey–Glass chaotic, the Jenkins–Box, sunspot number and Lorenz chaotic time series) and identify an example of nonlinear plant identification.

Acknowledgments

The authors would like to acknowledge the financial support of this work by grants from the General Direction of Scientific Research and Technological Renovation (DGRSRT), Tunisia, under the ARUB program 01/UR/11/02. Ajith Abraham acknowledges the support from the framework of the IT4Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 supported by Operational Programme 'Research and Development for Innovations' funded by Structural Funds of the European Union and state budget of the Czech Republic.

References

[1] M. Črepinšek, S.-H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them, Appl. Soft Comput. 19 (2014) 161–170.
[2] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, USA, 1975.
[3] R. Storn, K. Price, Differential Evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces, rapport technique (1995).
[4] A. Colorni, M. Dorigo, V. Maniezzo, Distributed optimization by ant colonies, Proceedings of European Conference on Artificial Life. Elsevier Publishing (1991).
[5] J. Kennedy, R. Eberhart, Particle swarm optimization, Western Australia, 27 November–1 December, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[6] M. Črepinšek, S.-H. Liu, L. Mernik, A note on teaching–learning-based optimization algorithm, Inf. Sci. 212 (2012) 79–93.
[7] M. Črepinšek, S.-H. Liu, L. Mernik, M. Mernik, Is a comparison of results meaningful from the inexact replications of computational experiments? Soft Comput. 20 (1) (2016) 223–235.
[8] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report-tr06, Erciyes University, Kayseri, Turkey, 2005.
[9] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Appl. Math. Comput. 214 (1) (2009) 108–132.
[10] P.Y. Kumbhar, S. Krishnan, Use of artificial bee colony (ABC) algorithm in artificial neural network synthesis, Int. J. Adv. Eng. Sci. Technol. 11 (1) (2011) 162–171 (ISSN: 2230-7818).
[11] D. Karaboga, B. Akay, C. Ozturk, Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks, International Conference on Modeling Decisions for Artificial Intelligence vol. 4617 (2007) 318–329.
[12] S. Nandy, P.P. Sarkar, A. Das, Training a feed-forward neural network with artificial bee colony based back-propagation, Int. J. Comput. Sci. Inf. Technol. 4 (4) (2012) 33–46.
[13] M.M. Rashidi, N. Galanis, F. Nazari, A. Basiri Parsa, L. Shamekhi, Parametric analysis and optimization of regenerative Clausius and organic Rankine cycles with two feedwater heaters using artificial bees colony and artificial neural network, Energy 36 (9) (2011) 5728–5740.
[14] A. Babalik, I. Babaoglu, A. Özkis, A pre-processing approach based on artificial bee colony for classification by support vector machine, Int. J. Comput. Commun. Eng. 2 (January (1)) (2013) 68–70.
[15] D. Karaboga, C. Ozturk, Fuzzy clustering with artificial bee colony algorithm, Sci. Res. Essays 5 (14) (2010) 1899–1902.
[16] N. Pathak, S.P. Tiwari, Travelling salesman problem using bee colony with SPV, Int. J. Soft Comput. Eng. 2 (3) (2012) (ISSN: 2231-2307).
[17] X. Kong, S. Liu, Z. Wang, L. Yong, Hybrid artificial bee colony algorithm for global numerical optimization, J. Comput. Inf. Syst. (2012) 2367–2374.
[18] X. Shi, Y. Li, H. Li, R. Guan, L. Wang, Y. Liang, An integrated algorithm based on artificial bee colony and particle swarm optimization, International Conference on Natural Computation vol. 5 (2010) 2586–2590.
[19] H. Dhahri, A.M. Alimi, Opposition-based particle swarm optimization for the design of beta basis function neural network, in: International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 2010, pp. 18–23.
[20] M.A. Alimi, On-line analysis of handwritten Arabic: a new approach to recognize segmented characters from cursive script, Les Annales Maghrébines de l'Ingénieur 14 (1) (2000) 7–27.
[21] C. Aouiti, A.M. Alimi, K. Karray, A. Maalej, Genetic algorithms to construct beta neuro-fuzzy systems, in: Proc. Int. Conf. Artificial & Computational Intelligence for Decision, Control & Automation, Monastir, Tunisia, 2000, pp. 88–93.
[22] H. Dhahri, A.M. Alimi, Opposition-based differential evolution for beta basis function neural network, in: IEEE Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 1–8.
[23] H. Dhahri, A.M. Alimi, F. Karray, Designing beta basis function neural network for optimization using particle swarm optimization, in: IEEE International Joint Conference on Neural Networks, Hong Kong, China, 2008, pp. 2564–2571.
[24] C. Aouiti, A.M. Alimi, K. Karray, A. Maalej, The design of bate basis function neural network and beta fuzzy systems by a hierarchical genetic algorithm, Fuzzy Sets Syst. 154 (2005) 251–274.
[25] H. Dhahri, A.M. Alimi, A. Abraham, Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing 79 (2012) 131–140.
[26] Y. Chen, B. Yang, J. Dong, A. Abraham, Time-series forecasting using flexible neural tree model, Inf. Sci. 174 (3/4) (2005) 219–235.
[27] Y. Chen, A. Abraham, B. Yang, Feature selection and classification using flexible neural tree, Neurocomputing 70 (2006) 305–313.
[28] Y. Chen, B. Yang, Q. Meng, Small-time scale network traffic prediction based on flexible neural tree, Appl. Soft Comput. 12 (2012) 274–279.
[29] S. Bouaziz, H. Dhahri, A.M. Alimi, Evolving flexible beta operator neural trees (FBONT) for time series forecasting, Doha-Qatar, in: T. Hung, et al. (Eds.), 19th International Conference in Neural Information Processing (ICONIP'12), Proceedings, Part III, Series: Lecture Notes in Computer Science, vol. 7665, 2012, pp. 17–24.
[30] A.M. Alimi, The beta fuzzy system: approximation of standard membership functions, Nabeul, Tunisia, November, in: Proc. 17eme Journees Tunisiennes d'Electrotechnique et d'Automatique: JTEA'97, vol. 1, 1997, pp. 108–112.
[31] C. Aouiti, A.M. Alimi, A. Maalej, A genetic designed beta basis function neural networks for approximating of multi-variables functions, in: Proc. Int. Conf. Artificial Neural Nets and Genetic Algorithms Springer Computer Science, Prague, Czech Republic, 2001, pp. 383–386.
[32] A.M. Alimi, The beta system: toward a change in our use of neuro-fuzzy systems, Int. J. Manag. (June) (2000) 15–19 (invited paper).
[34] K. Chellapilla, Evolving computer programs without subtree crossover, IEEE Trans. Evol. Comput. 1 (September (3)) (1997) 209–216.
[35] B.T. Zhang, H. Mühlenbein, Balancing accuracy and parsimony in genetic programming, Evol. Comput. 3 (1) (1995) 17–18.
[36] J. Kennedy, R.C. Eberhart, Particle swarm optimization, IEEE Press, Piscataway, NJ, in: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[37] L. Glass, M.C. Mackey, Pathological physiological conditions resulting from instabilities in physiological control systems, Ann. N. Y. Acad. Sci. 316 (1979) 214–235.
[38] K.B. Cho, B.H. Wang, Radial basis function based adaptive fuzzy systems their application to system identification and prediction, Fuzzy Sets Syst. 83 (1995) 325–339.
[39] F. Van Den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Trans. Evol. Comput. 8 (June (3)) (2004).
[40] C.F. Juang, C.M. Hsiao, C.H. Hsu, Hierarchical cluster-based multispecies particle-swarm optimization for fuzzy-system optimization, IEEE Trans. Fuzzy Syst. 18 (1) (2010) 14–26.
[41] G.E.P. Box, G.M. Jenkins, Time Series Analysis—Forecasting and Control, Holden Day, San Francisco, CA, 1976.

[42] S. Bouaziz, H. Dhahri, A.M. Alimi, A. Abraham, A hybrid learning algorithm for evolving flexible beta basis function neural tree model, Neurocomputing 117 (2013) 107–117.
[43] A.J. Izeman, J.R. Wolf and the Zurich sunspot relative numbers, Math. Intell. 7 (1) (1985) 27–33.
[44] J.R. McDonnell, D. Waagen, Evolving recurrent perceptrons for time-series modeling, IEEE Trans. Neural Netw. 5 (1) (1994) 24–38.
[45] R.A. Aliev, B.G. Guirimov, R.R. Aliev, Evolutionary algorithm-based learning of fuzzy neural networks. Part 2: recurrent fuzzy neural networks, Fuzzy Sets Syst. 160 (17) (2009).
[46] H. Dhahri, A.M. Alimi, A. Abraham, Designing beta basis function neural network for optimization using artificial bee colony (ABC), in: Proceedings of the International Joint Conference on Neural Networks, Brisbane, Australia , 10–15 June, 2012, pp. 1–7.
[47] A. Hussain, A new neural network structure for temporal signal processing, Proc. Int. Conf. on Acoustics Speech and Signal Processing (1997) 3341–3344.
[48] S. Yilmaz, Y. Oysal, Fuzzy wavelet neural network models for prediction and identification of dynamical systems, IEEE Trans. Neural Netw. (2010) 1599–1609.
[49] E.N. Lorenz, Deterministic nonperiodic flow, J. Atmos. Sci. 20 (March (2)) (1963) 130–141.
[50] C. Xiang, W. Zhou, Z. Yuan, Y. Chen, Xi. Xiong, A new parameters joint optimization method of chaotic time series prediction, Int. J. Phys. Sci. 6 (10) (2011) 2565–2571.
[51] D.T. Mirikitani, N.I. Nikolaev, Recursive Bayesian recurrent neural networks for time-series modeling, IEEE Trans. Neural Netw. 21 (2) (2010) 262–274.
[52] A. Miranian, M. Abdollahzade, Developing a local least-squares support vector machines-based neuro-fuzzy model for nonlinear and chaotic time series prediction, IEEE Trans. Neural Netw. Learn. Syst. 24 (2) (2013) 207–218.
[53] S. Bouaziz, A.M. Alimi, A. Abraham, Evolving flexible beta basis function neural tree for nonlinear systems, in: International Joint Conference on Neural Networks, Dallas Texas, 2013, pp. 1–8.
[54] S. Bouaziz, A.M. Alimi, A. Abraham, Universal approximation propriety of flexible beta basis function neural tree, International Joint Conference on Neural Networks, Beijing–China (2014).
[55] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, ACM Comput. Surv. 45 (3) (2013) (article 35).
[56] S.-H. Liu, M. Mernik, D. Hrnčič, M. Črepinšek, A parameter control method of evolutionary algorithms using exploration and exploitation measures with a practical application for fitting Sovova's mass transfer model, Appl. Soft Comput. 13 (9) (2013) 3792–3805.
[57] S. Rahnamayan, H.S. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Trans. Evol. Comput. 12 (1) (2008) 64–79.
[58] S. Rahnamayan, H.S. Tizhoosh, M.M.A. Salama, Opposition versus randomness in soft computing techniques, Appl. Soft Comput. 8 (2008) 906–918.
[59] L. Ljung, System Identification: Theory for the User, 2nd édition, Prentice Hall, 1999.
[60] J. Sjöberg, B. Zhang, L. Ljung, A. Benveniste, B. Deylon, P. Glorennec, H. Hjalmarsson, A. Juditsky, Nonlinear black-box modeling in system identification: a unified overview, Automatica 31 (1995) 1691–1724.
[61] B. Subudhi, D. Jena, A differential evolution based neural network approach to nonlinear system identification, Appl. Soft Comput. 11 (1) (2011) 861–871.
[62] S. Bouaziz, FBBFNT' Matlab code http://www.regim.org/publications/codes/fbbfnt/.

Souhir Bouaziz was born in Sfax (Tunisia) in 1984. She is graduated in Computer Engineering 2008, obtained a Ph.D. Computer Engineering in 2014 from the National Engineering School of Sfax (ENIS). She is currently assistant in the National Engineering School of Gabes (ENIG), and a member of the REsearch Groups in Intelligent Machines (REGIM-Lab). Her research interest includes computational intelligence: neural network, evolutionary computation, swarm intelligence.

Habib Dhahri was born in Sidi Bouzid (Tunisia) in 1966. He is graduated in computer science 2001, obtained a Ph.D. Computer Engineering in 2013 from the National Engineering School of Sfax (ENIS). His research interest includes computational intelligence: neural network, swarm intelligence, differential evolution and genetic algorithm.

Adel M. Alimi was born in Sfax (Tunisia) in 1966. He is graduated in Electrical Engineering 1990, obtained a Ph.D. and then an HDR both in Electrical & Computer Engineering in 1995 and 2000, respectively. He is now professor in Electrical & Computer Engineering at the University of Sfax. His research interest includes applications of intelligent methods (neural networks, fuzzy logic, evolutionary algorithms) to pattern recognition, robotic systems, vision systems, and industrial processes. He focuses his research on intelligent pattern recognition, learning, analysis and intelligent control of large scale complex systems. He is associate editor and member of the editorial board of many international scientific journals (e.g., "Pattern Recognition Letters", "Neurocomputing", "Neural Processing Letters", "International Journal of Image and Graphics", "Neural Computing and Applications", "International Journal of Robotics and Automation", "International Journal of Systems Science", etc.). He was guest editor of several special issues of international journals (e.g., Fuzzy Sets & Systems, Soft Computing, Journal of Decision Systems, Integrated Computer Aided Engineering, Systems Analysis Modelling and Simulations). He was the general chairman of the International Conference on Machine Intelligence ACIDCA-ICMI'2005 & 2000. He is an IEEE senior member and member of IAPR, INNS and PRS. He is the 2009–2010 IEEE Tunisia Section Treasurer, the 2009–2010 IEEE Computational Intelligence Society Tunisia Chapter Chair, the 2011 IEEE Sfax Subsection, the 2010–2011 IEEE Computer Society Tunisia Chair, the 2011 IEEE Systems, Man, and Cybernetics Tunisia Chapter, the SMCS corresponding member of the IEEE Committee on Earth Observation, and the IEEE Counselor of the ENIS Student Branch.

Ajith Abraham received the Ph.D. degree in Computer Science from Monash University, Melbourne, Australia. He is currently the Director of Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, USA, which has members from more than 85 countries. He has a worldwide academic and industrial experience of over 20 years. He works in a multi-disciplinary environment involving machine intelligence, network security, various aspects of networks, e-commerce, Web intelligence, Web services, computational grids, data mining, and their applications to various real-world problems. He has numerous publications/citations (h-index 40) and has also given more than 50 plenary lectures and conference tutorials in these areas. Since 2008, he is the Chair of IEEE Systems Man and Cybernetics Society Technical Committee on Soft Computing and a Distinguished Lecturer of IEEE Computer Society representing Europe (since 2011). Dr. Abraham is a Senior Member of the IEEE, the Institution of Engineering and Technology (UK) and the Institution of Engineers Australia (Australia), etc. He is the founder of several IEEE sponsored annual conferences, which are now annual events. More information at: http://www.softcomputing.net.