A Hybrid Learning Algorithm For Evolving Flexible Beta Basis Function Neural Tree Model

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

[a] *REsearch Group on Intelligent Machines (REGIM), University of Sfax, National School of Engineers (ENIS), BP 1173, Sfax 3038, Tunisia,*

Email: souhir.bouaziz@ieee.org, habib.dhahri@ieee.org, adel.alimi@ieee.org

[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, Email: ajith.abraham@ieee.org*

Abstract—In this paper, a tree-based encoding method is introduced to represent the Beta basis function neural network. The proposed model called Flexible Beta Basis Function Neural Tree (FBBFNT) can be created and optimized based on the predefined Beta operator sets. A hybrid learning algorithm is used to evolving FBBFNT Model: the structure is developed using the Extended Genetic Programming (EGP) and the Beta parameters and connected weights are optimized by the Opposite-based Particle Swarm Optimization algorithm (OPSO). The performance of the proposed method is evaluated for benchmark problems drawn from control system and time series prediction area and is compared with those of related methods.

Keywords: Flexible Beta Basis Function Neural Tree model, Extended Genetic Programming, Opposite-based Particle Swarm Optimization algorithm, Time-series forecasting, Control System.

1. Introduction

The initiative of using Beta basis functions for designing Artificial Neural Network was introduced by Alimi in 1997 [21] and in this case the network is called Beta Basis Function Neural Network (BBFNN). The BBFNN is a three layer feed-forward neural network that generally adopts a linear transfer function for the output layer and a beta basis function as a non linear transfer function for the hidden units. The beta basis function has several advantages over the Gaussian function, such as its ability to generate more rich shapes (asymmetry, linearity, etc.) [20] and its great flexibility. Therefore several success researches have been achieved in the use of the BBFNN for classification (pattern recognition) [34-36], prediction [1-4], etc. The BBFNN has also the opportunities for an application to constraint optimization problems [10].

A BBF neural network's performance depends mainly on two issues which are the network structure and the Beta parameter's adjustment. For a given problem, a BBFNN structure is not unique and there can exist different ways to create a corresponded structure. Thus, the design of BBFNN automatically is required. Furthermore, connected weights and Beta parameters which include the center, width and form parameters of BBFNNs can be learned by many methods, i.e., back-propagation algorithm [35], genetic algorithm [10, 11, 12], differential evolution algorithm [1, 3, 4], particle swarm optimization algorithm [2] 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) [10] and Hierarchical Multi-dimensional Differential Evolution (HMDDE) [33].

* Correspondingauthor. Tel.+216 22 172 000,

E-mail Adresses : souhir.bouaziz@ieee.org (S. Bouaziz)

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 encourage us to use the tree-based encoding method which was introduced by Chen [13-19, 22], for representing a BBF neural network and so the new representation is called Flexible Beta Basis Function Neural Tree (FBBFNT). This model is more flexible than the classical BBFNN seen that it can find automatically the number of nodes as well as the number of hidden layers.

In this paper, the FBBFNT model is applied to benchmark problems drawn from control system and 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 the Opposite-based Particle Swarm Optimization algorithm (OPSO).

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 model

The Beta function is the name used by Legendre and Whittaker and Watson (1990) for the Beta integral (also called the Eulerian integral of the first kind). The first time where the Beta function was used as transfer function for neural networks was in 1997 by Alimi [21].

This function was chosen as a transfer function for many reasons [40- 43], including, its large flexibility (Fig.1), its universal approximation characteristics [11, 12] and its ability to generate rich shapes (asymmetry, linearity, etc.) [20].

The Beta basis function is defined by:

$$\beta(x, x_0, x_1, p, q) = \begin{cases} \left(\frac{x-x_0}{c-x_0}\right)^p \left(\frac{x_1-x}{x_1-c}\right)^q & if x \in]x_0, x_1[\\ 0 & else \end{cases} \quad (1)$$

Where $p > 0, q > 0$, x_0, x_1 are the real parameters, $x_0 < x_1$ and $c = \frac{p\, x_1 + q\, x_0}{p+q}$ is the center of beta function.

Let $\sigma = x_1 - x_0$, σ is the width of the beta function which can be seen as a scale factor for the distance $\|x - c\|$. So:

$$\begin{cases} x_0 = c - \frac{\sigma p}{p+q} \\ x_1 = c + \frac{\sigma q}{p+q} \end{cases} \quad (2)$$

(1) and (2) =>

$$\beta(x, c, \sigma, p, q) = \begin{cases} \left[1 + \frac{(p+q)(x-c)}{\sigma p}\right]^p \left[1 - \frac{(p+q)(c-x)}{\sigma q}\right]^q \\ if\ x \in \left]c - \frac{\sigma p}{p+q}, c + \frac{\sigma q}{p+q}\right[\\ 0 \qquad else \end{cases} \quad (3)$$

Some proprieties taken from [40] in the one-dimensional case are presented as following:

$$\beta(x_0) = \beta(x_1) = 0, \quad (4)$$

$$\beta(x_c) = 1, \quad (5)$$

$$\frac{d\beta(x)}{dx} = \left[\frac{p x_1 + q x_0 - (p+q)x)}{(x-x_0)(x_1-x)}\right] \beta(x), \quad (6)$$

$$\frac{d\beta(x_c)}{dx} = \frac{d\beta(x_0)}{dx} = \frac{d\beta(x_1)}{dx} = 0, \quad (7)$$

$$\frac{p}{q} = \frac{x_c - x_0}{x_1 - x_c}. \tag{8}$$

If p=1, q=0:
$$\beta(x) = \begin{cases} \left(\frac{x-x_0}{x_1-x_0}\right) & if\ x \in [x_0, x_1] \\ 0 & if \quad x < x_0 \\ 1 & if \quad x > x_1 \end{cases} \tag{9}$$

So, Beta function may be considered as a piecewise linear function of x, if (p=1, q=0) or (p=0, q=1).

For the multi-dimensional case, the Beta function has the some proprieties as the one-dimensional because the multi-dimensional Beta function is simply the product of m one-dimensional Beta functions.

In addition, for any given Gaussian function $Gauss(x,\mu,\sigma)$ and for any given precision ε, there exists a Beta function $\beta(x, x_0, x_1, p, q)$ that approximates the Gaussian function with an error of less than ε for any xϵℝ.

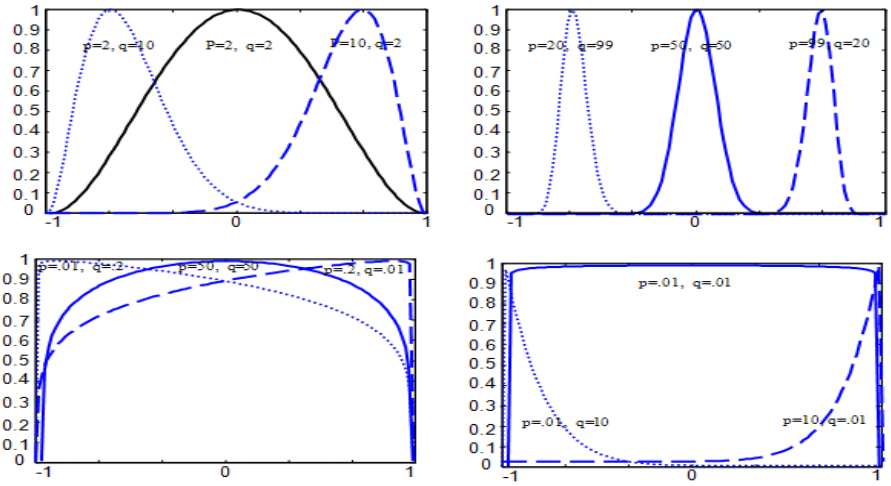

Fig. 1. Examples of Beta Basis Function.

In this study, we have adopted a tree-based encoding for representing the Beta basis function neural network instead of the matrix-based encoding as it is more flexible and gives a more modifiable and adjustable structure.

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

$$S = F \cup T = \{+_2, +_3, \ldots, +_N, /_N\} \cup \{x_1, \ldots, x_M\} \tag{10}$$

Where:

- $+_n$ (n = 2, 3, …, N) denote non-terminal nodes and represent flexible Beta basis functions 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,. . ., x_M are terminal nodes and defining the input vector values.

The output of a non-terminal node is calculated as a flexible neuron model (see Fig.2).

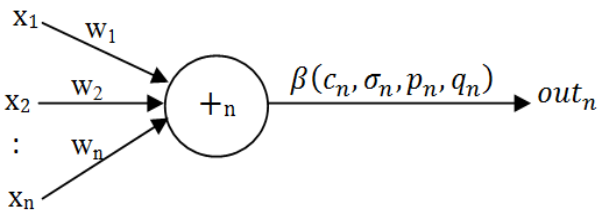

Fig. 2. A flexible neuron Beta operator.

In the creation process of Beta basis function neural tree, if a function node, i.e., $+_n$ 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 (3), 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:

$$y_n = \sum_{j=1}^{n} w_j * x_j \tag{11}$$

Where x_j (j = 1, …, n) are the inputs of the selected node and w_j (j = 1, …, n) are the connected weights.
The output of node $+_n$ is then calculated using (3) by:

$$out_n = \beta(y_n, c_n, \sigma_n, p_n, q_n) = \begin{cases} \left[1 + \frac{(p_n+q_n)(y_n-c_n)}{\sigma_n p_n}\right]^{p_n} \left[1 - \frac{(p_n+q_n)(c_n-y_n)}{\sigma_n q_n}\right]^{q_n} \\ \qquad if\ y_n \in \left]c_n - \frac{\sigma_n p_n}{p_n+q_n}, c_n + \frac{\sigma_n q_n}{p_n+q_n}\right[\\ 0 \qquad\qquad else \end{cases} \tag{12}$$

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.3. The overall output of flexible Beta basis function neural tree can be computed recursively by depth-first method from left to right.

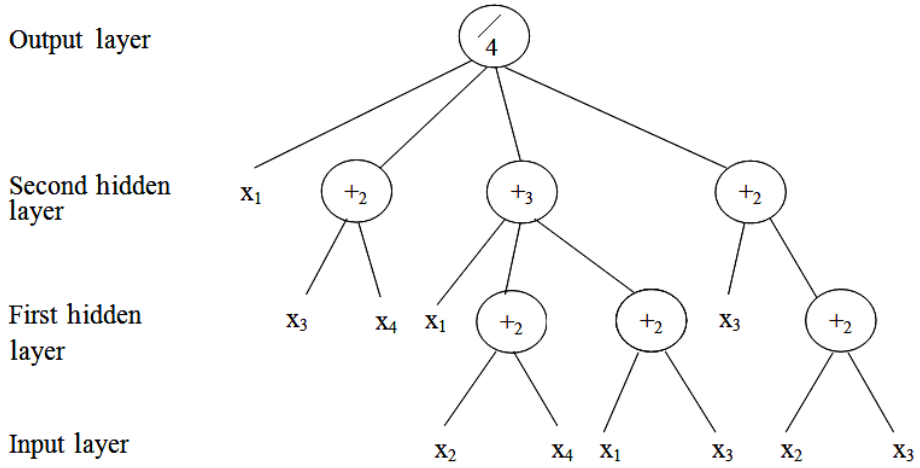

Fig. 3. A typical representation of FBBFNT: function node set F = {$+_2$, $+_3$, $+_4$, $/_4$}, and terminal node set T = {x_1, x_2, x_3, x_4}.

3. The hybrid FBBFNT evolving algorithm

The optimization of FBBFNT includes the tree-structure and parameter optimization. In this study, finding an optimal Beta basis function neural tree structure is achieved by using Extended Genetic Programming algorithm and the parameters implanted in a FBBFNT are optimized by Opposite-based PSO.

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 [0, *NL_Max*] and uniformly distributed random number of nodes for each layer in [0, *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 with the offspring nodes) of each tree are also randomly generated in its search spaces. Each individual is then evaluated according to the fitness function (section 3.4).

In the second time, some neural tree variation operators, which are an extension of standard GP, are applied to the population individuals as following in order to generate a new generation:

Selection: The selection operator is used 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. Then, a threshold *T* (between 0 and 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. Individuals used for the replacement (crossover or mutation) are selected by the binary tournament selection. 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.

Crossover: In EGP, the tree structure crossover operation is implemented by taking randomly selected two sub-trees in the individuals and selecting 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.4.

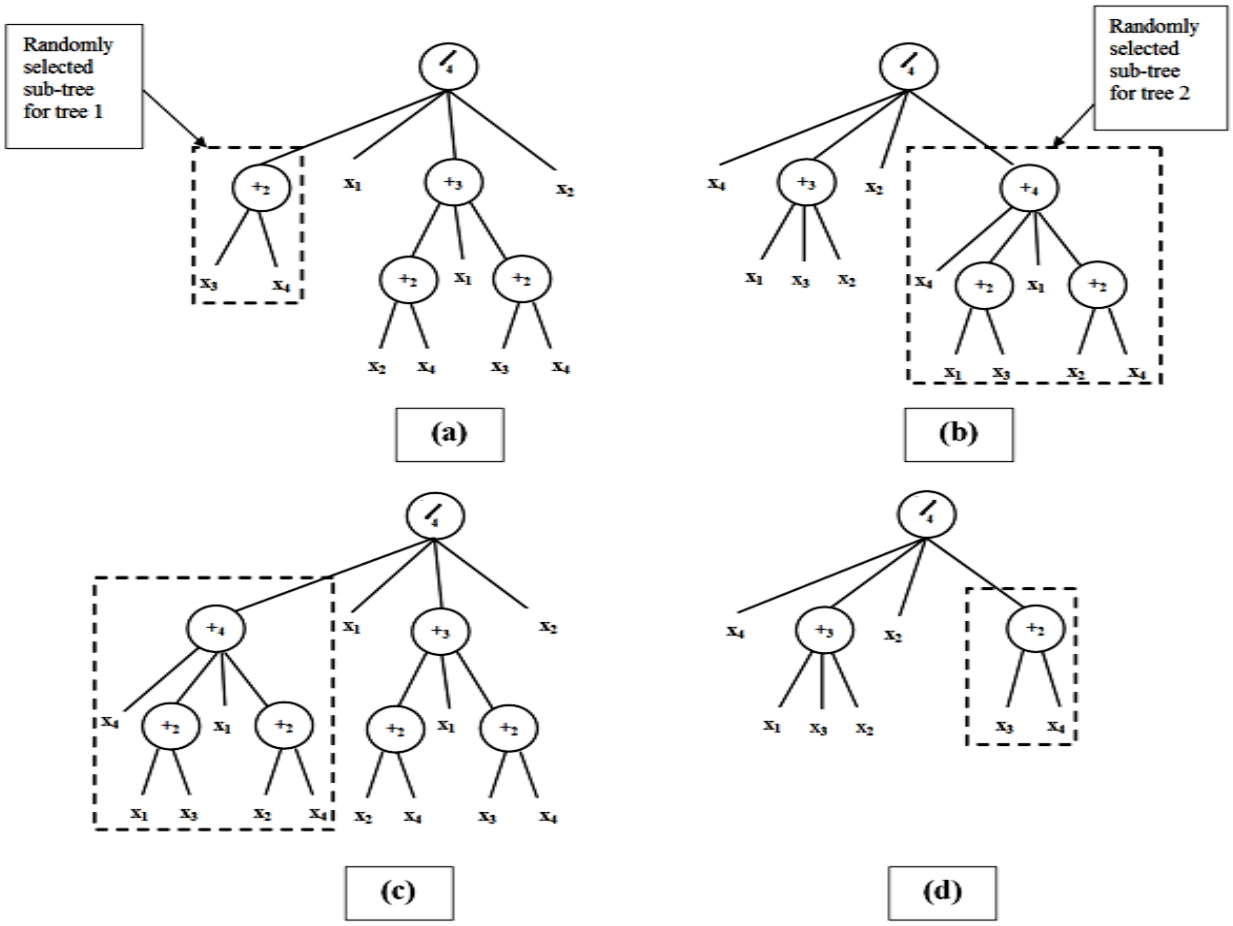

Fig. 4. Examples of the EGP crossover operator: (a) Selected tree 1. (b) Selected tree 2. (c) Tree 1 after crossover operator with tree 2. (d) Tree 2 after crossover operator with tree 2.

Mutation: four different mutation operators were used to generate offspring from the parents (Fig.5). These mutation operators are as follows:

1. *Changing one leaf node*: select one leaf node randomly in the neural Beta basis function tree and replace it with another leaf node;
2. *Changing all the leaf nodes:* select each leaf nodes in the neural Beta basis function tree and replace it with another leaf node;
3. *Growing:* select a random leaf node in hidden layer of the neural Beta basis function tree and replace it with a randomly generated sub-tree;

4. *Pruning:* randomly select a Beta operator node in the neural tree and replace it with a random leaf node.

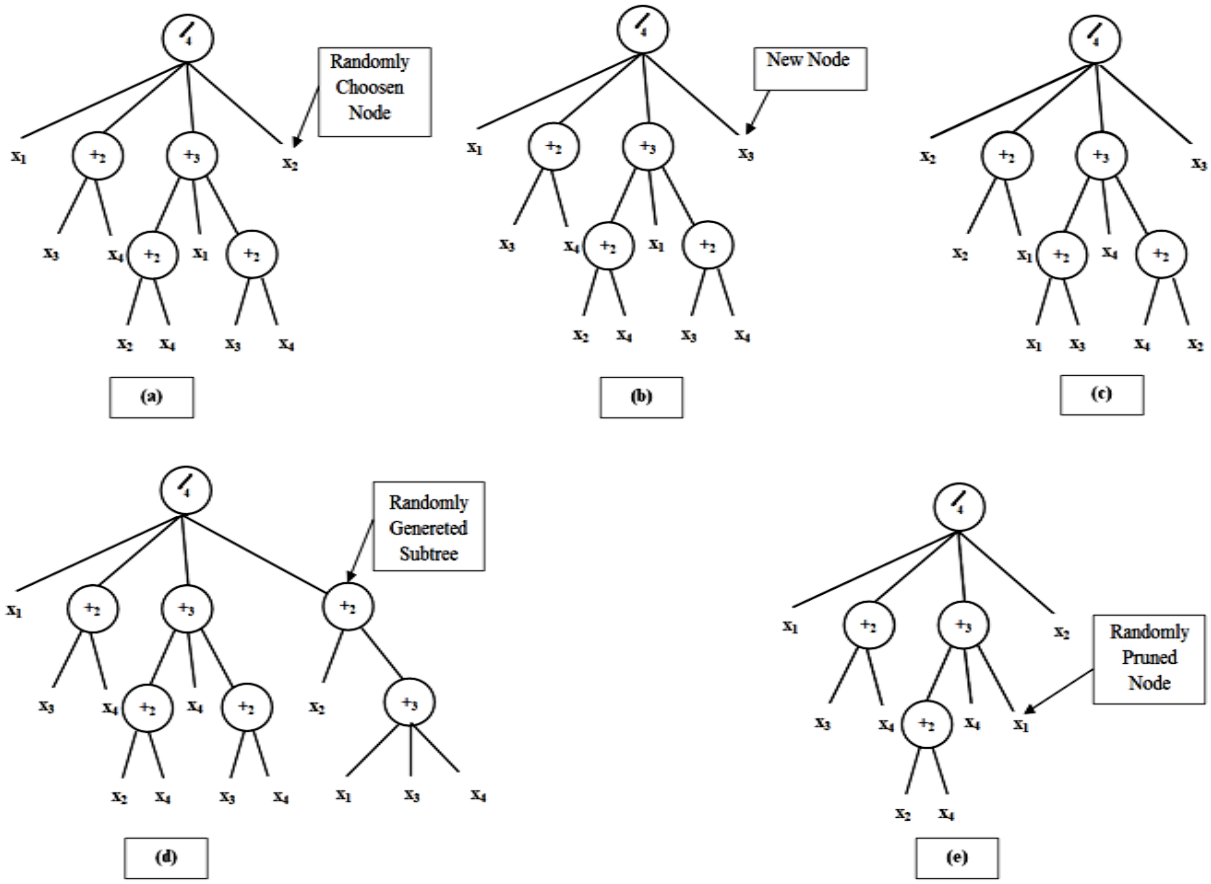

Fig. 5. Examples of the four EGP mutation operators: (a) Original tree. (b) Changing one leaf node. (c) Changing all the leaf nodes. (d) Growing. (e) Pruning.

Following the work of Chellapilla [23], the EGP tree mutation operators were applied to each of the parents to generate an offspring using the following steps:

(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.

After each mutation or crossover operator, a redundant terminals pruning operator will be applied, if it is possible; i.e. if a Beta operator node has more than two terminals, the redundant terminals should be deleted.

Each individual of the new generation which is generated after the variation operators cited above, is evaluated using the fitness function (section 3.4). If this generation is better than the previous one, the population is updated. This procedure can be repeated until the terminal criteria are achieved; i.e., a better structure is found or a limit number of EGP iterations is reached.

3.2. The Particle Swarm Optimization algorithm

PSO was proposed by Kennedy and Eberhart [25] 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. At each iteration *t*, using $p_i(t)$ and $p_g(t)$, a new velocity for particle *i* is updated by :

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

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 [$-v_{max}$,$+v_{max}$]. Based on the calculated velocities, each particle changes its position as the following equation:

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

3.3. The Opposite-Based particle swarm optimization for parameter optimization

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 to form a new algorithm called Opposite-based Particle Swarm Optimization (OPSO).

In this study, a particle consists of the Beta parameters (centre, spread and the form parameters) and weights encoded in each flexible Beta basis function neural tree, which will be optimized by OPSO algorithm.

The learning process of OPSO is described as follows:

Step 0 (Initialization): At iteration $t = 0$, the initial positions $x_i(t = 0)$ ($i = 1, \ldots, NP$) which are *NParam* × *NN* matrix, are generated uniformly distributed randomly;

$$x_i(0) = a_j + rand_i(b_j - a_j) \quad (15)$$

Where:

- *NP* is the number of particles,
- *NParm* is the number of parameters (Beta parameters and weights),
- *NN* is the number of FBBFNT nodes,
- $[a_j, b_j]$ is the search space of each parameter.

Generate the opposite population as follows:

$$\bar{x}_i(0) = \begin{cases} \alpha_i\left(x_i(0) + \frac{a_j+b_j}{2}\right), & if\ x_i(0) < \frac{a_j+b_j}{2} \\ \alpha_i\left(x_i(0) - \frac{a_j+b_j}{2}\right), & if\ x_i(0) > \frac{a_j+b_j}{2} \end{cases}, \quad \alpha_i \in]0,1[\quad (16)$$

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 according to the beta neural system using a fitness function described in the next section.

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 (13).

Step 3 (Position update): Depending on their velocities, each particle changes its position and its opposite- position according to the equation (14):

$$x_i(t+1) = x_i(t) + \big(1 - \Psi(\mathrm{t})\big)\, v_i(t+1) \quad (17)$$

$$\bar{x}_i(t+1) = \bar{x}_i(t) + \big(1 - \Psi(\mathrm{t})\big)\, v_i(t+1) \quad (18)$$

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.

Step 5 (End criterion): The OPSO learning process ends when a predefined criterion is met. In this study, the criterion is the goal or total number of OPSO iterations.

3.4. Fitness function

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

$$Fit(i) = \sqrt{\frac{1}{P}\sum_{j=1}^{P}(y_t^j - y_{out}^j)^2} \quad (19)$$

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. $Fit(i)$ denotes the fitness value of i^{th} individual.

3.5. The Learning Algorithm for FBBFNT model

To find an optimal or near-optimal FBBFNT model, structure and parameters optimization are used alternately.
Combining of the EGP and OPSO algorithms, a hybrid algorithm for evolving FBBFNT model is described as follows and is depicted in Fig.6:

(a) Randomly create an initial population (FBBFNT trees and its corresponding parameters); $G = 0$, where G is the generation number of the learning algorithm;
global iterations = 0;
(b) Structure optimization is achieved by the Extended Genetic Programming (EGP) as described in section 3.1;
(c) If a better structure is found or a maximum number of EGP iterations is attained, then go to step (d),
global iterations = global iterations + EGP iterations;
otherwise go to step (b);
(d) Parameter optimization is achieved by the OPSO 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 particle;
(e) If the maximum number of OPSO iterations is attained, or no better parameter vector is found for a fixed time then go to step (f);
global iterations = global iterations + OPSO iterations;
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);

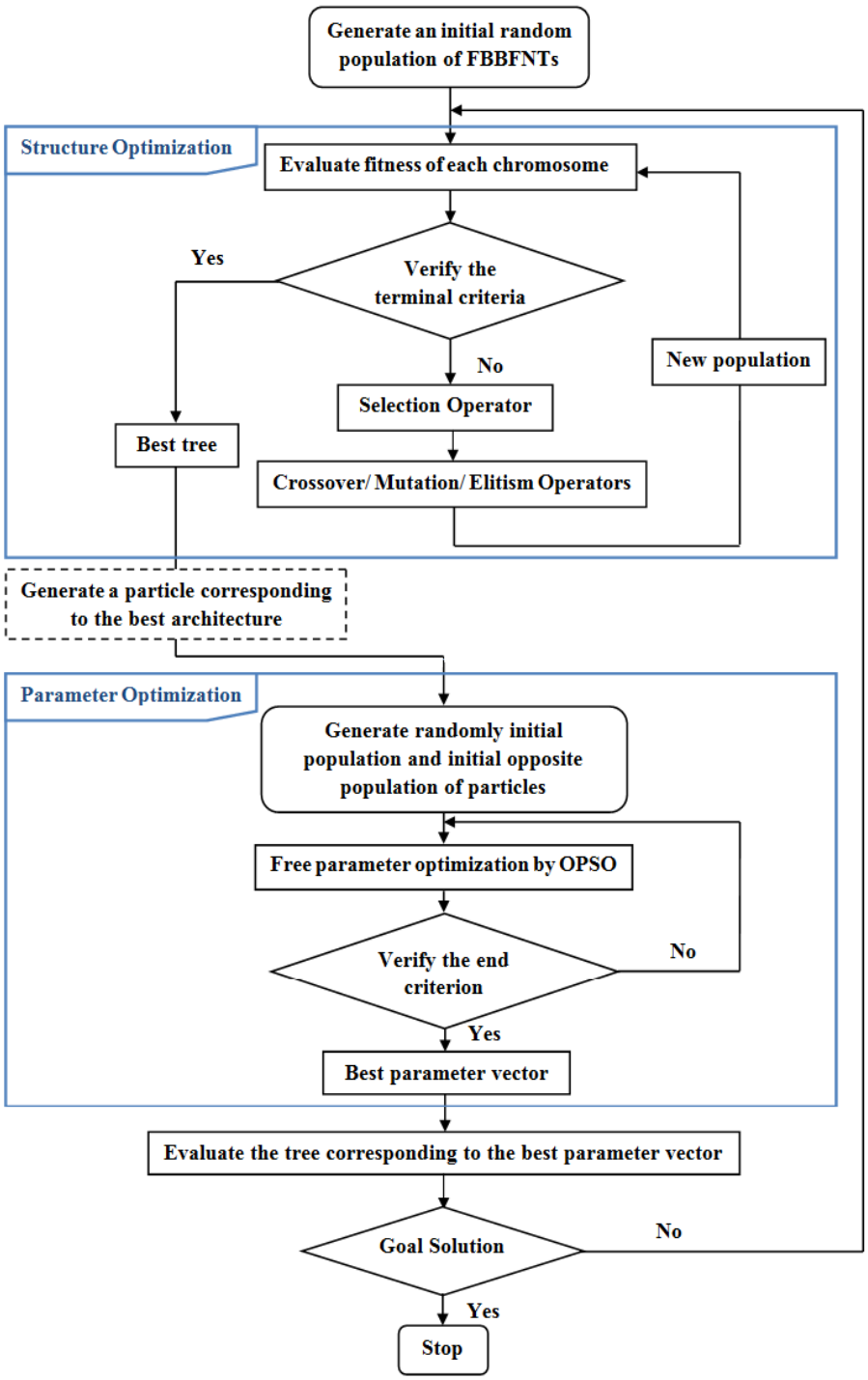

Fig. 6. The Hybrid Learning Algorithm for FBBFNT model.

4. Experimental results

To evaluate its performance, the proposed FBBFNT model is submitted to various benchmark problems: Mackey-Glass chaotic time series, the Jenkins–Box time series and an example of nonlinear control system. The set of parameters which we used for our algorithm are: Population size, Crossover probability Mutation probability, Generation gap and Maximum EGP iteration number for the Extended Genetic Programming; Population size, Maximum OPSO iteration number, c_1 and c_2 for the Opposite-based Particle Swarm Optimization; and Maximum global iteration number for the Hybrid learning algorithm. After many experiences of the system parameters, the chosen parameters to be used for all problems are as listed in table 1.

Table 1. FBBFNT Parameters

EGP	
Parameter	***Initial value***
Population size	50
Crossover probability	0.3
Mutation probability	0.6
Generation gap	0.9
Maximum EGP iteration number	1000
OPSO	
Parameter	***Initial value***
Population size	50
Maximum OPSO iteration number	4000
c_1	0.8
c_2	0.8
Hybrid learning algorithm	
Parameter	***Initial value***
Maximum global iteration number	40 000
Connected weights	rand[0, 1]
Beta centre	rand[*min*(*x*), *max*(*x*)]
Beta spread	rand[0, |*max*(*x*)-*min*(*x*)|]
Beta form parameters (p, q)	rand[0, 5]

For all examples the illustrated results are obtained by averaging the results in 15 runs.

4.1. Example 1: Mackey–Glass time series prediction

A time-series prediction problem can be constructed based on the Mackey–Glass [26] differential equation:

$$\frac{d(x(t))}{dt} = \frac{ax(t-\tau)}{1+x^c(t-\tau)} - bx(t) \tag{20}$$

The setting of the experiment varies from one work to another. In this work, the same parameters of [33] and [14], 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 $x(t-18)$. 1000 sample points are used in our study. The first 500 data pairs of the series are used as training data, while the remaining 500 [14, 33] are used to validate the model identified. The used Beta operator sets to create an optimal FBONT model is $S = F \cup T = \{+_2, +_3, +_4, +_5, /_5\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i (i = 1, 2, 3, 4) denotes $x(t), x(t-6), x(t-12)$ and $x(t-18)$, respectively. After 16 generations (G = 16) and 23020 global iterations of the hybrid learning algorithm, an optimal FBBFNT model was obtained with RMSE 0.006101. The RMSE value for validation data set is 0.006823. The evolved FBBFNT is shown in Fig.7. The actual time-series data and the output of FBBFNT model are shown in Fig.8.

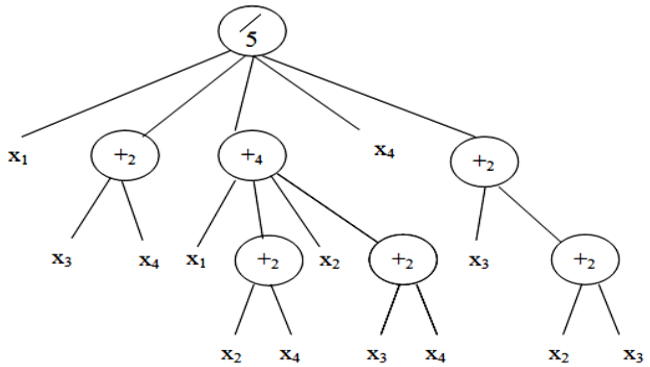

Fig. 7. The evolved FBBFNT for prediction of the Mackey-Glass time-series.

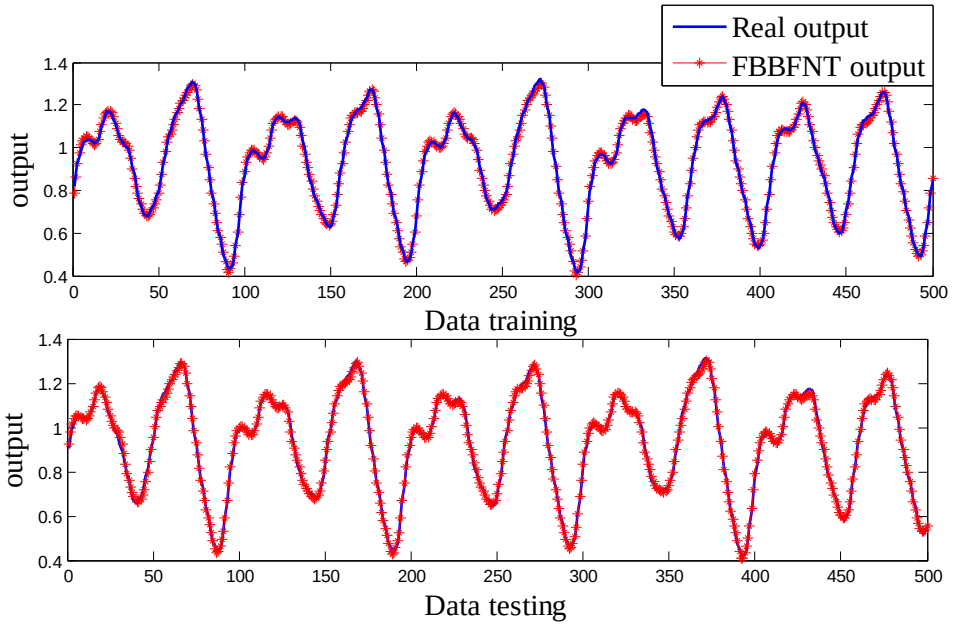

Fig. 8. The actual time series data and the output of the evolved FBBFNT model for forecasting Mackey-Glass data.

The proposed system is essentially compared with Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network (HMDDE-BBFNN) [33] and the FNT model with Gaussian function as flexible neuron operator [14] and also with other systems. The HMDDE-BBFNN approach adopts for parameters: 50 to the population size and 10000 to a total number of iterations. Moreover, the parameter setting for the FNT system [14] are 30 to the population size and 135 as generation number. A comparison result of different methods for forecasting Mackey-Glass data is shown in Table 2. As observed, the FBBFNT achieves the lowest testing and training error.

Table 2. Comparison of different methods for the Mackey-Glass time-series.

Method	**Training error (RMSE)**	**Testing error (RMSE)**
PSO-BBFN [2]	-	0.027
HMDDE–BBFNN [33]	0.0094	0.0170
Aouiti [11]	-	0.013
Classical RBF [27]	0.0096	0.0114
CPSO [39]	0.0199	0.0322
HCMSPSO [38]	0.0095	0.0208
FNT [14]	0.0069	0.0071
FBBFNT	**0.0061**	**0.0068**

4.2. Example 2 : Box and Jenkins' Gas Furnace Problem

The gas furnace data of Box and Jenkins [28] 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.

The used instruction set for creating a FBBFNT model is $S = F \cup T = \{+_2, +_3, +_4, /_4\} \cup \{x_1, x_2\}$, where x_i (i = 1, 2) denotes $y(t-1), u(t-4)$, respectively.

After 10 generations (G = 10) and 11031 global iterations of the learning algorithm, the optimal Beta basis function neural tree model was obtained with the RMSE 0.004796. The RMSE value for

validation data set is 0.011618. The evolved FBBFNT is shown in Fig.9. The actual time-series data and the output of FBBFNT model are shown in Fig.10. A comparison result of different methods for forecasting Jenkins-Box data is shown in Table 3.

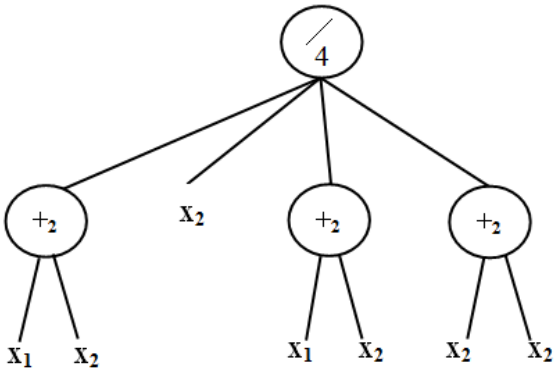

Fig. 9. The evolved FBBFNT for prediction of the Jenkins–Box time-series $(y(t-1), u(t-4))$.

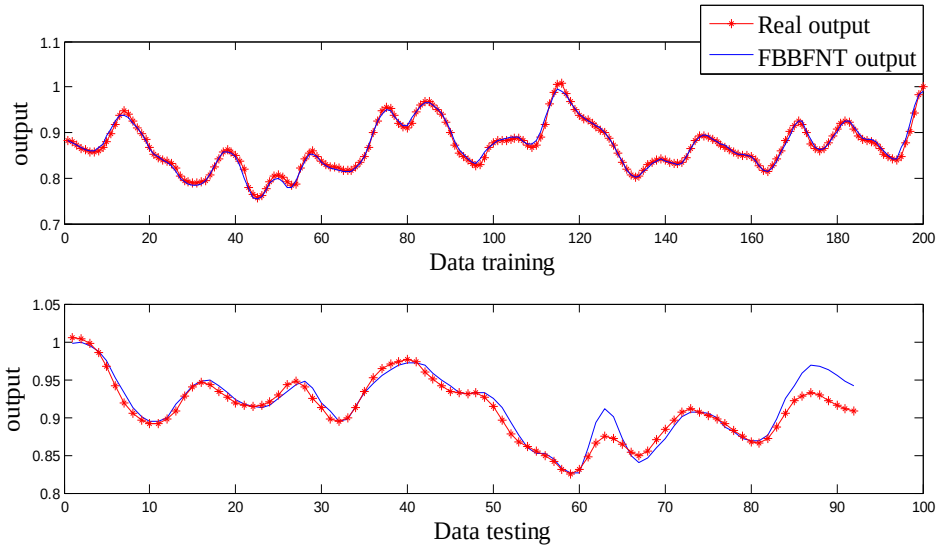

Fig. 10. The actual time series data, output of the evolved FBBFNT model for forecasting Jenkins–Box data $(y(t-1), u(t-4))$.

Table 3. Comparison of testing errors of Box and Jenkins.

Method	Prediction error (RMSE)
ANFIS model [29]	0.0845
FuNN model [30]	0.0714
HyFIS model [31]	0.0648
FNT [14]	0.0256
HMDDE [33]	0.2411
FBBFNT	**0.011618**

In order to illustrate again the performance of our approach, 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 4.

Table 4. Comparison of training and testing errors of Box and Jenkins.

Input	Training error (RMSE)			Testing error (RMSE)		
	ODE [37]	HMDDE [33]	FBBFNT	ODE [37]	HMDDE [33]	FBBFNT
y(t-1), u(t-3)	0.1411	0.1328	**0.003460**	0.4194	0.2276	**0.011457**
y(t-3), u(t-4)	0.2850	0.0210	**0.006017**	0.7773	0.4224	**0.020998**
y(t-2), u(t-4)	0.2898	0.1365	**0.005719**	0.6602	0.3200	**0.016534**
y(t-1), u(t-2)	0.2924	0.1735	**0.005757**	0.6801	0.2334	**0.011496**
y(t-1), u(t-4)	0.2926	0.2411	**0.011618**	0.5132	0.3745	**0.004796**
y(t-4), u(t-4)	0.3428	0.1594	**0.007112**	0.8894	0.4549	**0.023775**
y(t-2), u(t-3)	0.3051	0.1702	**0.006611**	0.7199	0.2700	**0.018575**
y(t-1), u(t-1)	0.4151	0.1598	**0.012439**	0.6056	0.2577	**0.012123**
y(t-4), u(t-3)	0.4301	0.1921	**0.014273**	1.2771	0.6148	**0.028954**
y(t-1), u(t-6)	0.5661	0.6619	**0.012475**	0.8410	0.6638	**0.012193**
y(t-3), u(t-3)	0.5176	0.1600	**0.010645**	1.0347	0.2521	**0.025013**
y(t-2), u(t-2)	0.9753	0.2773	**0.015724**	0.6261	0.1615	**0.025640**
y(t-1), u(t-5)	0.6303	0.3333	**0.012425**	0.6518	0.5595	**0.012250**
y(t-4), u(t-5)	0.6373	0.0178	**0.011611**	0.9698	0.0203	**0.010810**
y(t-2), u(t-1)	0.6844	0.1960	**0.023624**	1.2726	0.2759	**0.022670**
y(t-2), u(t-5)	0.6804	0.2165	**0.023612**	1.1808	0.4021	**0.022377**
y(t-3), u(t-5)	0.7338	0.1346	**0.013797**	1.0470	0.2307	**0.018032**
y(t-3), u(t-2)	0.8600	0.2128	**0.032620**	1.4138	0.2760	**0.030713**
y(t-4), u(t-6)	1.1126	0.1379	**0.020954**	1.4677	0.2635	**0.024996**
y(t-2), u(t-6)	0.8600	0.2128	**0.023613**	1.2639	0.5590	**0.022504**
y(t-4), u(t-2)	1.1963	0.2152	**0.022968**	1.6377	0.2737	**0.033545**
y(t-3), u(t-6)	0.8600	0.2128	**0.033015**	1.4641	0.4027	**0.030156**
y(t-3), u(t-1)	1.2702	0.2135	**0.032628**	1.6475	0.2803	**0.030708**
y(t-4), u(t-1)	2.0217	0.2695	**0.039824**	2.0217	0.2695	**0.036392**

From the above simulation results, it can be seen that the proposed FBBFNT model works well for generating prediction models of Box and Jenkins' gas furnace problem.

4.3. Example 3 : Nonlinear Plant Control

In this example, the nonlinear system [28] to be controlled is expressed by:

$$y_p(t+1) = \frac{y_p(t)[y_p(t-1)+2][y_p(t)+2.5]}{8.5+[y_p(t)]^2+[y_p(t-1)]^2} + u(t) \quad (21)$$

where $y_p(t)$ is the output of the system at the t^{th} time step and $u(t)$ is the plant input which is uniformly bounded in the region [-2, 2]. The identification model is in the form of:

$$y_p(t+1) = f(y_p(t), y_p(t-1)) + u(t) \quad (22)$$

Where $f(y_p(t), y_p(t-1))$ is the nonlinear function of y_p(t) and y_p(t-1) that will be the inputs of FBBFNT. The output from neural system will be y_p(t+1). In this example, 500 data samples are used for training and 500 data samples are used for testing the performance of the evolved model. The input signal u(t) is calculated as following:

$$u(t) = \begin{cases} 2\cos(2\pi t/100) & if\ t \le 200 \\ 1.2\sin(2\pi t/20) & if\ 200 < t \le 500 \end{cases} \quad (23)$$

The used instruction set for creating a FBBFNT model $S = F \cup T = \{+_2, +_3, +_4, /_4\} \cup \{x_1, x_2\}$, where x_i (i = 1, 2) denotes $y_p(t), y_p(t-1)$, respectively.

After 17 generations (G = 17) and 20024 global iterations, the optimal Beta operator neural tree model was obtained with the RMSE 0.01882. The RMSE value for validation data set is 0.10161. The evolved FBBFNT is shown in Fig.11. The actual time-series data and the output of FBBFNT model are

shown in Fig.12. A comparison result of different methods for forecasting nonlinear plant control data is shown in Table 5. The performance of FBBFNT in error fitness is better than that of ODE and those HMDDE. Results show that applying the FBBFNT for the nonlinear plant control system improves the generalization error.

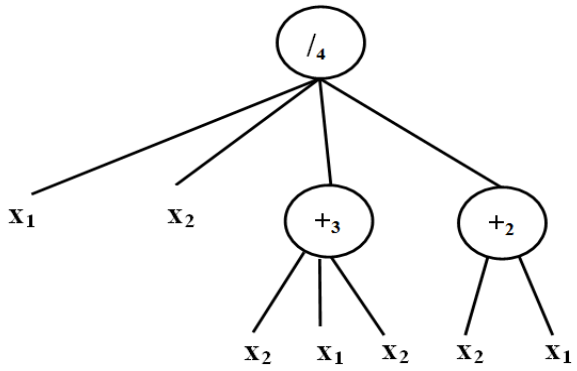

Fig. 11. The evolved FBBFNT for nonlinear plant control.

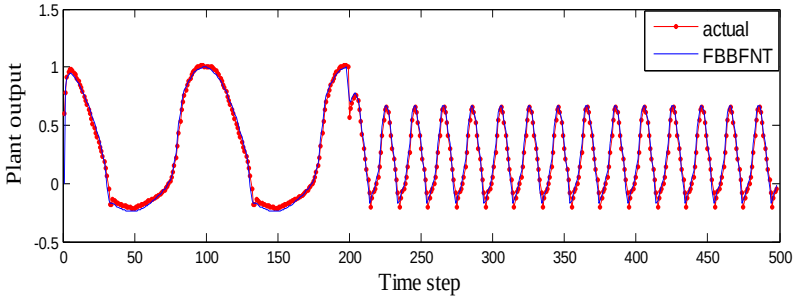

Fig. 12. FBBFNT identification performance.

Table 5. Comparison of training and testing errors of nonlinear plant control.

Input	**Training error (RMSE)**			**Testing error (RMSE)**		
	ODE [37]	**HMDDE [33]**	**FBBFNT**	**ODE [37]**	**HMDDE [33]**	**FBBFNT**
y(k),y(k − 1) u(k)	0.019	0.019	**0.01882**	0.1137	0.110	**0.10161**

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 various benchmark problems. The work demonstrates that the new learning algorithm can successfully optimize the structure and parameters of Beta basis function neural network simultaneously by using a 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 Opposite-based Particle Swarm Optimization algorithm (OPSO).

The experiment results show that the FBBFNT model can effectively predict the time-series problem such as Mackey-Glass chaotic time series, the Jenkins–Box time series and an example of nonlinear plant control system.

Acknowledgment

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. Dhahri, H., Alimi, A. M.: Opposition-based differential evolution for beta basis function neural network, IEEE Congress on Evolutionary Computation, pp. 1 - 8, Barcelona, Spain (2010).
2. Dhahri, H., Alimi, A. M., Karray, F.: Designing beta basis function neural network for optimization using particle swarm optimization, IEEE International Joint Conference on Neural Networks, pp. 2564-2571, Hong Kong, China (2008).
3. Dhahri, H., Alimi, A. M.: Automatic Selection for the Beta Basis Function Neural Networks, NICSO, pp. 461-474 (2007).
4. Dhahri, H., Alimi, A. M.: The modified differential evolution and the RBF (MDE-RBF) neural network for time series prediction, Proc. of the International Conference, pp. 5245–5250 (2006).
5. Liu, H., Liu, D., Deng, L.-F.: Chaotic time series prediction using fuzzy sigmoid kernel-based support vector machines, Chin. Phys. 15 (6), 1196–1200 (2006).
6. Li, H., Zhang, J., Xiao, X.: Neural Volterra filter for chaotic time series prediction, Chin. Phys. 14 (11), 2181–2188 (2005).
7. Meng, Q., Zhang, Q., Mu, W.: A novel multi-step adaptive prediction method for chaotic time series, Acta Phys. Sin. 55 (4), 1666–1671 (2006).
8. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3, 82–102 (1999).
9. Stanley, K. O., Miikkulainen, R.: Evolving neural networks through augmenting topologies, Evolutionary Computation, vol. 10, 99–127 (2002).
10. Aouiti, C., Alimi, A. M., Karray, K., Maalej, A.: The design of bate basis function neural network and beta fuzzy systems by a hierarchical genetic algorithm, fuzzy Sets and Systems, vol. 154, 251-274 (2005).
11. Aouiti, C., Alimi, A. M, Maalej, A.: A Genetic Designed Beta Basis Function Neural Networks for approximating of multi-variables functions, Proc. Int. Conf. Artificial Neural Nets and Genetic Algorithms Springer Computer Science, Prague, Czech Republic, pp. 383-386 (2001).
12. Aouiti, C., Alimi, A. M., Karray, K., Maalej, A.: Genetic Algorithms to Construct Beta Neuro-Fuzzy Systems, Proc. Int. Conf. Artificial & Computational Intelligence for Decision, Control & Automation, pp. 88-93, Monastir, Tunisia (2000).
13. Chen, Y., Yang, B., Meng, Q.: Small-time Scale Network Traffic Prediction Based on Flexible Neural Tree, Applied Soft Computing, Vol.12, 274-279 (2012).
14. Chen, Y., Yang, B., Dong, J., Abraham, A.: Time-series forecasting using flexible neural tree model, Inf. Sci. 174 (3/4), 219–235 (2005).
15. Chen, Y., Abraham, A., Yang, B.: Feature Selection and Classification using Flexible Neural Tree, Neurocomputing, 70, 305-313 (2006).
16. Chen, Y., Jiang, S., Abraham, A.: Face Recognition Using DCT and Hybrid Flexible Tree, In Proc. of the International Conference on Neural Networks and Brain, pp. 1459-1463 (2005).
17. Chen, Y., Peng, L., Abraham, A.: Gene Expression Profile Using Flexible Neural Trees, IDEAL2006, 4224, pp.1121-1128 (2006).
18. Chen, Y., Peng, L., Abraham, A.: Exchange Rate Forecasting Using Flexible Neural Trees, Lecture Notes on Computer Science, pp. 3973, 518- 523 (2006).
19. Chen, Y., Meng, Q., Zhang, Y.: Optimal Design of Hierarchical B-Spline Networks for Nonlinear System Identification, Proceedings of the International Conference on Sensing, Computing and Automation, pp. 3263-3268 (2006).
20. Alimi, A.M: The Beta System: Toward a Change in Our Use of Neuro-Fuzzy Systems, International Journal of Management, Invited Paper, June, 15-19 (2000).
21. Alimi, A.M.: The Beta Fuzzy System: Approximation of Standard Membership Functions, Proc. 17eme Journees Tunisiennes d'Electrotechnique et d'Automatique: JTEA'97, Nabeul, Tunisia, Nov., vol. 1, pp. 108-112 (1997).
22. Chen, Y., Yang, B., Dong, J.: Evolving flexible neural networks using ant programming and PSO algorithm, International Symposium on Neural Networks (ISNN'04), Lecture Notes on Computer Science, vol. 3173, pp. 211–216 (2004).
23. Chellapilla, K.: Evolving computer programs without subtree crossover, IEEE Transactions on Evolutionary Computation, 1(3): 209-216 (1998).
24. Novosád, T., Platoš, J., Snášel, V., Abraham, A., Fiala, P.: Heavy Facilities Tension Prediction Using Flexible Neural Trees, International Conference of Soft Computing and Pattern Recognition, pp.396-401 (2011).
25. Kennedy, J., Eberhart, R.C.: Particle swarm optimization, Proceedings of the 1995 IEEE International Conference on Neural Networks. IEEE Press, Piscataway, NJ, vol. 4, pp. 1942–1948 (1995).

26. Lzvbjerg, M., Krink, T.: Extending particle swarms with self-organized criticality, Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002). IEE Press, Piscataway, NJ, USA, vol. 2, pp. 1588–1593 (2002).
27. Cho, K.B., Wang, B.H.: Radial basis function based adaptive fuzzy systems their application to system identification and prediction, Fuzzy Sets and Systems 83, pp. 325–339 (1995).
28. Box, G. E. P., Jenkins, G.M.: Time Series Analysis--Forecasting and Control. Holden Day, San Francisco, CA (1976).
29. Nie, J.: Constructing fuzzy model by self-organising counter propagation network, IEEE Transactions on Systems Man and Cybernetics 25, 963–970 (1995).
30. Jang, J.-S.R., Sun, C.-T., Mizutani, E.: Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence, Prentice-Hall, Upper Saddle River, NJ (1997).
31. Kasabov, N., Kim, J.S, Watts, M., Gray, A.: FuNN/2—a fuzzy neural network architecture for adaptive learning and knowledge acquisition, Information Science 101, pp. 155–175 (1996).
32. Stepniewski, S.W, Keane, A.J: Pruning Back-propagation Neural Networks Using Modern Stochastic Optimization Techniques, Neural Computing & Applications 5, 76-98 (1997).
33. Dhahri, H., Alimi, A.M., Abraham, A.: Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, neurocomputing 79, 131-140 (2012).
34. Alimi, M.A.: The recognition of arabic handwritten characters with the Beta neuro-fuzzy network. Proc. 17éme Journées Tunisiennes d'Électrotechnique et d'Automatique, JTEA'97, Nabeul, Tunisia, Vol. 1, pp. 349–356 (1997).
35. Alimi, M.A.: On-Line Analysis of Handwritten Arabic: A New Approach to Recognize Segmented Characters from Cursive Script, Les Annales Maghrébines de l' Ingénieur, vol. 14, no. 1, pp. 7-27 (2000).
36. Bezine, H., Alimi, M.A., Derbel, N.: Handwriting Trajectory Movements Controlled by a Beta-Elliptic Model, Proc. 7th International Conference on Document Analysis and Recognition: ICDAR'2003, Edinburgh, UK, August, pp. 1228-1232 (2003).
37. Subudhi, B. and Jena, D.: A differential evolution based neural network approach to nonlinear system identification, Applied Soft Computing 11(1), 861-871, (2011).
38. Juang, C.F., Hsiao, C.M. and Hsu, C.H.: Hierarchical Cluster-Based Multispecies Particle-Swarm optimization for Fuzzy-System Optimization. IEEE Transactions on Fuzzy Systems, 18(1), 14-26 (2010).
39. Van Den Bergh, F. and Engelbrecht, A. P.: A cooperative approach to particle swarm optimization, IEEE Trans. Evol. Comput., vol. 8, no. 3, 225–239, Jun. (2004).
40. Alimi, A. M., Beta neuro-fuzzy systems, TASK Quarterly Journal, Special Issue on "Neural Networks" edited by W. Duch and D. Rutkowska, vol. 7, no. 1, 23-41 (2003).
41. Alimi, A.M., What are the Advantages of Using the Beta Neuro-Fuzzy System ?, Proc. IEEE/IMACS Multiconference on Computational Engineering in Systems Applications, Tunisia, vol. 2, pp. 339-344 (1998).
42. Masmoudi, M., Samet, M., and Alimi ,M.A., A bipolar implementation of the Beta Neuron, International Journal of Electronics, vol. 87, no. 6, 675-682 (2000).
43. Njah, M. Alimi, A. M. and Chtourou, M., A learning algorithm for the Beta neuro-fuzzy network, Proc. Int. Conf. Artificial and Computational Intelligence for Decision, Control and Automation, Monastir, Tunisia, pp. 76- 81 (2000).

Biography

Souhir Bouaziz was born in Sfax (Tunisia) in 1984. She graduated in Computer Engineering 2008. She is currently working toward the Ph.D degree with the University of Sfax.
Her research interest includes computational intelligence: neural network, evolutionary computation, swarm intelligence.

Habib Dhahri was born in Sidi Bouzid (Tunisia) in 1975. He graduated in computer science 2001. He is currently working toward the Ph.D degree with the University of Sfax.
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 graduated in Electrical Engineering 1990, obtained a PhD 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