Evolving Flexible Beta Basis Function Neural Tree for Nonlinear Systems

Souhir Bouaziz, Adel.M. Alimi, Ajith Abraham

Abstract— **In this paper, a new evolving artificial neural network using evolutionary computation is introduced. Based on the pre-defined Beta operator sets, this model called Flexible Beta Basis Function Neural Tree (FBBFNT), can be created and learned. The structure is developed using the Extended Immune Programming (EIP). The Beta parameters and connected weights are optimized using the Hybrid Bacterial Foraging Optimization algorithm. The performance of the proposed method is evaluated for nonlinear systems and compared with those of related methods.**

Keywords— **Extended Immune Programming; Hybrid Bacterial Foraging Optimization algorithm; Flexible Beta Basis Function Neural Tree; nonlinear systems.**

I. Introduction

Artificial Neural Network (ANN) is a growing interdisciplinary field which considers the systems as adaptive, distributed and mostly nonlinear, three of the elements found in the real applications. It is placed at the crossroads of various biological-inspired approaches where it is considered as an abstract simulation of a real nervous system.

The performance of the ANN can be mainly conditioned by the appropriate structure and the training algorithm. Many efforts have been provided in the literature to address these issues. Since 1991, Yao [1] is one of the first researchers who have exploited possible benefits arising from the interactions between ANNs and evolutionary computation, to design and evolve ANN. In such case the model is noted Evolving Artificial Neural Network (EANN).

To evolve the ANN structure, several researchers used tree representation and evolutionary computation to design and optimize automatically the ANN structure. The evolutionary computation used for solve this task include: Genetic Programming (GP) [2, 3, 4], probabilistic incremental program evolution algorithm (PIPE) [5], and Immune programming [6]. Moreover, the ANN parameters (weights and transfer function parameters) can be learned by various evolutionary computation such genetic algorithm [7], particle swarm optimization algorithm [8], Bacterial Foraging Optimization Algorithm [9], etc.

S. Bouaziz and A.M. Alimi is with REsearch Group on Intelligent Machines (REGIM), University of Sfax, National School of Engineers (ENIS), BP 1173, Sfax 3038, Tunisia, (souhir.bouaziz@ieee.org, adel.alimi@ieee.org)
Ajith Abraham is with IT4Innovations, VSB-Technical University of Ostrava, Czech Republic, (ajith.abraham@ieee.org).

Recently, there has been an increasing interest to optimize the ANN structure and parameters simultaneously [3-6, 10, 11].

The most used transfer function is the Gaussian function. However, the Beta function [12, 13] shows its performance against the Gaussian function for typical representation of ANN, due to its large flexibility and its universal approximation capacity [7, 12, 13]. The initiative of using Beta functions for designing Artificial Neural Network was introduced by Alimi [12] and in this case the network is called Beta Basis Function Neural Network (BBFNN).

Although matrix-representation of BBFNN has a number of advantages such as better approximation capabilities and simple network topologies, adapting such representation suffers from slow premature convergence characteristics and makes the BBFNN's structure difficult to regulate. For these reasons, a tree-based encoding method is adopted, in this paper, to design the BBFNN. A hybrid algorithm which simultaneously optimizes the structure and parameters, is used to evolve the new model. This model is named Flexible Beta Basis Function Neural Tree (FBBFNT). The structure is developed using the Extended Immune Programming (EIP). The fine tuning of the Beta parameters (centre, spread and the form parameters) and weights encoded in the structure is optimized using Hybrid Bacterial Foraging Optimization Algorithm (HBFOA).

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

II. Flexible Beta Basis Function Neural Tree Model

The first time where the Beta function was used as transfer function for neural networks was in 1997 by Alimi [12] and the corresponding model is named Beta basis function neural network.

In this study, the Beta basis function neural network is encoded by the tree-based encoding method instead of the matrix-based encoding method, since this method is more flexible and gives a more adjustable and modifiable architecture. This new representation is called Flexible Beta Basis Function Neural (FBBFNT). The FBBFNT is formed of a node set *NS* representing the union of function node set *F* and terminal node set *T*:

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

Where:

- B_n (n = 2, 3, …, 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 represents a linear transfer function.
- $x_1, x_2, \dots, x_M$ are terminal nodes and define the input vector values.

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

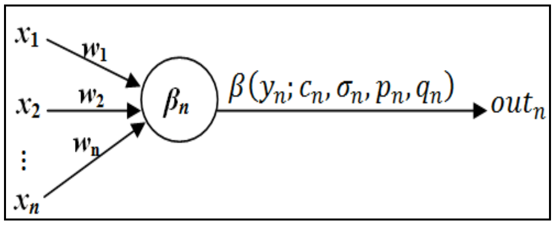


Fig. 1. A flexible Beta Basis Function.

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 transfer 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:

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

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 β_n is then calculated by:

$$out_n = \beta_n(y_n, c_n, \sigma_n, p_n, q_n) = \begin{cases} \left[1 + \frac{(p_n + q_n)(y_n - c_n)}{\sigma_n p_n}\right]^{p_n} \left[1 - \frac{(p_n + q_n)(c_n - y_n)}{\sigma_n q_n}\right]^{q_n} \\ \qquad if\ y_n \in \left]c_n - \frac{\sigma_n p_n}{p_n + q_n}, c_n + \frac{\sigma_n q_n}{p_n + q_n}\right[\\ 0 \qquad else \end{cases} \quad (3)$$

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 in Fig.2. The overall output of flexible Beta basis function neural tree can be computed recursively by depth-first method from left to right.

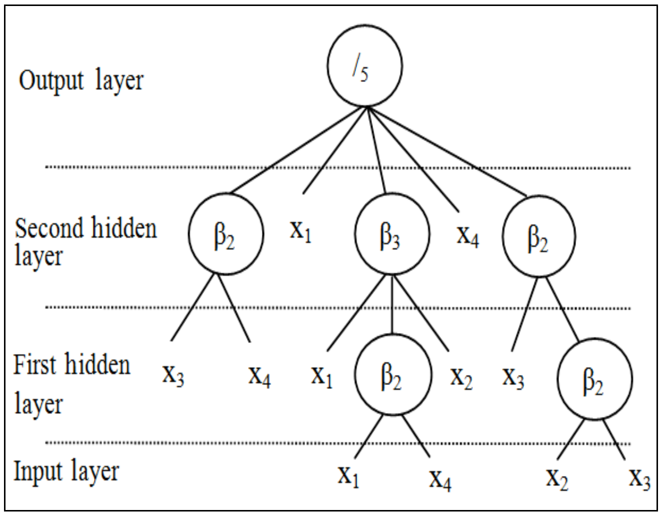


Fig. 2. A typical representation of FBBFNT: function node set F = {β_2, β_3, $/_5$}, and terminal node set T = {x_1, x_2, x_3, x_4}.

III. THE HYBRID LEARNING ALGORITHM FOR THE FBBFNT MODEL

The optimization of FBBFNT includes two issues which are structure optimization and parameter optimization. In this work, finding an optimal or a near optimal Beta basis function neural tree structure is achieved by using Extended Immune Programming (EIP) algorithm and the parameters implanted in a FBBFNT are optimized by Hybrid Bacterial Foraging Optimization Algorithm (HBFOA).

A. The Extended Immune Programming for structure optimization

Based on the results found by Musilek *et al.* in [14], IP has a more convergence capacity than GP: successful solutions are found in fewer generation numbers with the evident improvement when using a small antibody population. These reasons encouraged us to apply IP with an adapted version of our model in the search of the optimal structure. This new algorithm is called Extended Immune Programming (EIP). The EIP global process is formed by seven main steps as follow and it is summarized in Fig. 3:

1) Initialization: Firstly, the initial population (repertoire) of flexible Beta basis function neural trees (antibodies) is randomly generated with random structures (number of layers and number of nodes for each layer). The node parameters (Beta parameters and weights) of each tree are also randomly generated in its search spaces.

2) Evaluation: All of the antibodies (*NA* antibodies) are compared to an antigen representing the problem to be solved, and their fitness *Fit(i)* (affinity) with respect to the antigen is calculated (according to the section *C*).

3) Cloning: An antibody Ab_i of the current population is selected to be treated; if its affinity is higher than a random generated number so this antibody can be cloned with a probability P_c, and placed in the new population.

4) Mutation: if a selected high-affinity (corresponding to low RMSE value) antibody in the previous step has not been cloned due to the stochastic character of the cloning process, it is submitted to mutation. Four different mutation operators were used:

- Changing one terminal node: select one terminal node randomly in this antibody and replace it with another terminal node;
- Changing all the terminal nodes: select each terminal nodes in the antibody and replace it with another terminal node;
- Growing: select a random terminal node in hidden layer of the antibody and replace it with a randomly generated sub-tree;
- Pruning: randomly select a function node in the antibody and replace it with a random terminal node.

The EIP mutation operators were applied according to the method of Chellapilla [15] as follows:

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.

5) Replacement: if the current antibody Ab_i is not selected to be cloned or mutated, a new antibody is generated and placed into the new population with a certain probability, P_r. This way, low affinity antibodies are implicitly replaced.

6) Iteration-population: steps 3–5 (cloning, mutation and replacement) are repeated until a complete new population has been created.

7) Iteration-algorithm: after the new population has been constructed, the generation number (EIP_Iter = 1 during initialization) is incremented, $EIP\_Iter = EIP\_Iter + 1$. The algorithm so iteratively proceeds through steps 2–6 (evaluation, cloning, mutation, replacement, iteration–repertoire) until a terminal criterion is reached.

B. Hybrid Bacterial Foraging Optimization Algorithm

Recently, Bacterial Foraging Optimization Algorithm (BFOA) [16] has drawn the attention of researchers from diverse fields of knowledge especially due to its biological motivation and graceful structure. For these reasons it has been successfully applied for some engineering applications such as optimal control [16], harmonic estimation [17], etc. It is also effectively used, in recent years, to learn artificial neural network for many fields such as prediction systems [18], classification problems [19], and power transformers [20].

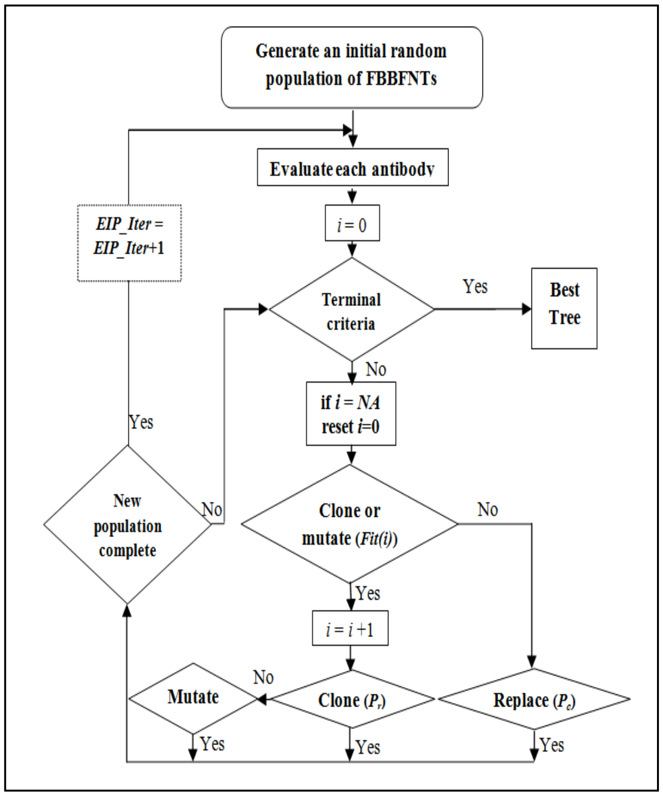


Fig. 3. Flowchart of the Extended Immune Programming algorithm.

According to the experimental results of [16] to several benchmark functions, BFOA possesses a poor convergence behavior over multimodal and rough fitness landscapes as compared to other naturally inspired optimization techniques like GA, PSO and DE. Its performance is also affected with the growth of search space dimensionality. In order to get better convergence and to improve the BFOA's performance on complex optimization problems, it becomes necessary to optimize this algorithm. That's why scientists are trying to hybridize BFOA with some other algorithms, i.e., PSO and DE.

We therefore are motivated to study this new trend of swarm intelligence and we proposed a new hybridization of BFOA called Hybrid Bacterial Foraging Optimization Algorithm (HBFOA) [21]. This hybrid algorithm has shown its efficiency mainly with multimodal and high dimensional functions and also in overcoming the problem of premature convergence. HBFOA is centered essentially on the chemotaxis step of the BFOA process by creating a new proposed adaptive chemotactic step size, and by integrating the ideas of PSO velocity and DE operators to update the movement of the bacterium. In fact, a new chemotactic step size is proposed depending on the current fitness value and the global best fitness value. It is expected so to provide better convergence behavior as compared to a fixed step size. Thus, to get a better improvement and to accelerate the convergence speed, the parameter *Fbest* was introduced in the adaptation equation of the step size *C* as follows:

$$S(i) = \frac{|Fit(i) - Fbest| + 1}{\square} \tag{4}$$

$S(i)$ is the new adaptive chemotactic step size for the i-th bacterium and *Fbest* is the parameter fitness function value for the globally best bacterium.

From (4), we can see that if $|\mathrm{Fit(i)} - \mathrm{Fbest}|$ is large, then the bacterium is far away from the global best, so *S* will be large also. On the other hand, if $|\mathrm{Fit(i)} - \mathrm{Fbest}|$ is small, then the bacterium is very close the global best and consequently *S* will be small also. Using this new adaptive chemotaxis step size, the bacterium with better function value (in a nutrient-rich zone) will try to take a smaller step and retain its current position. Moreover, the bacterium located at a poor nutrient region of the fitness landscape will take large step sizes to attain better fitness. After undergoing a chemotactic step of the BFO algorithm, each bacterium takes the mutation and crossover steps of the DE algorithm. Then, the result vector obtained was first coupled with the velocity operator of PSO algorithm, and then with the updated step size factor *S*. Also, we coupled this result vector in the same time with the adaptive *S* and with the velocity. Finally, a selection step of DE algorithm was introduced in a modified way to select the best vector result. The set of HBFOA parameters is as shown in table 1.

TABLE I. DESCRIPTION OF THE PARAMETERS

Parameters	Description
NP	The number of bacteria in the population
Nc	The number of chemotactic steps
Nre	The number of reproduction steps
Ned	The number of elimination-dispersal events
Ns	Swimming length
Ped	Elimination-dispersal probability
C(i)	The size of the step taken in the random direction specified by the tumble
F, CR	DE parameters
C_1, C_2, R_1, R_2, *w*, velocity	PSO parameters

This algorithm is adopted to be used for the FBBFNT parameter optimization. The initial population is formed by the initial positions of bacterium, X_i (i=1, …, *NP*), which are randomly generated *NParam* x *Size* matrix. Where: *NParam* is the number of parameters (Beta parameters and weights) and *Size* is the number of FBBFNT nodes. The learning process of HBFOA is described in the algorithm 1.

C. Fitness Function

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

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

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. $Fit(i)$ denotes the fitness value of ith individual.

Algorithm1 : HBFO Algorithm.

```
input : Nc, Nre, Ned, Ns, Ped, F, CR, $c_1$, $c_2$.
output: $p_g$: the global best position.
begin
  for $l = 1$ to Ned do /* Ned:  number of elimination-dispersal events */
    for $k = 1$ to Nre do          /* Nre:  number of reproduction steps */
      for $j = 1$ to Nc do        /* Nc :  number of chemotactic steps */
        for $i = 1$ to NP do                    /* for each bacterium */
          [a ] Take a chemotactic step for every bacterium ($i$) as follows;
          [b ] Compute fitness function $Fit_{par}(i)$;
          [c ] Flast = $Fit_{par}(i)$;
          [d ] Tumble: calculates the velocity term corresponding to the $i$-th
          bacterium $velocity(i)$ according to equation (4.6);
          [e ] Move: $X_i(j+1) = X_i(j) + C(i) * velocity(i)$; Evaluate $X_i(j+1)$;
          [f ] Swim: $m = 0$        /* counter for swim length */;
          while $m < Ns$ do
            m = m + 1;
            if $Fit_{par}(i)) < Flast$ then
              $Flast = Fit_{par}(i)$;
              $X_i(j+1) = X_i(j) + C(i) * velocity(i)$; Evaluate $X_i(j+1)$;
            else
              $m = Ns$;
            end
          end
          Hybridization :
          - DE Mutation: $V_i(j+1) = p_g(j+1) + F * (X_{r_1^i}(j) - X_{r_2^i}(j))$;
          - DE Crossover: according to equation (2.7) using $V_i(j+1)$, the trial
          vector $U_i(j+1)$ is obtained;
          - PSO: $H_i(j+1) = U_i(j+1) + C(i) * velocity$ ;
          - adaptive BFOA: chemotaxis step size $S(i)$ is updated according to
          equation (4.15) $K_i(j+1) = U_i(j+1) + S(i) * velocity$ ;
          - Adaptive DE selection: The original vector $X_i(j+1)$ is replaced by
          the new best position between ($U_i(j+1)$ or $H_i(j+1)$ or $K_i(j+1)$) if
          its fitness function value is smaller.
        end
        - Update $p_i(j+1)$ and $p_g(j+1)$ local/global best positions, respectively;
      end
      - Reproduction: Compute the health of the bacterium $i$:
      $Fit_{health}^i = \sum_{j=1}^{Nc+1} Fit_{par}(i)$;
      - Bacteria with the highest $Fit_{health}$ values die, the remaining bacteria
      reproduce;
    end
    - Elimination-dispersal: Eliminate and disperse bacteria with probability Ped;
  end
end
```

D. The hybrid evolving algorithm for FBBFNT model

To find an optimal or near-optimal FBBFNT model, structure and parameter optimization are used alternately. Combining of the EIP and HBFO algorithms, a hybrid algorithm for learning FBBFNT model is described as follows:

a) Randomly create an initial population (FBBFNT trees and its corresponding parameters);

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

GlobalIter = 0, where *GlobalIter* is the global iteration number of the learning algorithm;

b) Structure optimization is achieved by the Extended Immune Programming (EIP) as described in section *A*;

c) If a better structure is found or a maximum number of EIP iterations is attained, then go to step (d),

GlobalIter = *GlobalIter* + *EIP_Iter*;
otherwise go to step (b);

d) Parameter optimization is achieved by the HBFO 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 bacteria

e) If the maximum number of HBFOA iterations is attained, or no better parameter vector is found for a fixed time then go to step (f);

GlobalIter = *GlobalIter* + *HBFOA_Iter*;
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).

IV. Experimental results

To evaluate its performance, the proposed FBBFNT model is submitted to various nonlinear systems especially for prediction, i.e., Mackey-Glass chaotic, Jenkins–Box, and sunspot number time series. After many experiences of the system parameters, the chosen parameters to be used for all problems are as listed in table 2.

TABLE II. FBBFNT Parameters

EIP	
Parameter	***Initial value***
Population size (*NA*)	50
Cloning probability (P_c)	0.7
Replacement probability (P_r)	0.5
Maximum generation number	1000
HBFOA	
Parameter	***Initial value***
Population size (*NP*)	50
Maximum iteration number	4000
Nc	100
Nre	16
Ned	2
Ns	12
Ped	0.25
F	0.5
CR	0.9
c_1	1.2
c_2	0.5
Hybrid evolving algorithm	
Parameter	***Initial value***
Maximum global iteration number	40 000
Connected weights	rand[0, 1]
Beta center c	rand[$min(x)$, $max(x)$]
Beta spread σ	rand[0, $\lvert max(x)-min(x)\rvert$]
Beta form parameters (p, q)	rand[0, 5]

A. Example 1: Mackey–Glass time series prediction

A time-series prediction problem can be constructed based on the Mackey–Glass [22] differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{c}(t-\tau)} - bx(t) \qquad (4)$$

The setting of the experiment varies from one work to another. In our case, we take a = 0.2, b = 0.1, c = 10, and τ = 17. These values are the same ones used by the comparison systems [5, 11, 23-25]. As in the studies mentioned above, the task 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 are used to validate the model identified.

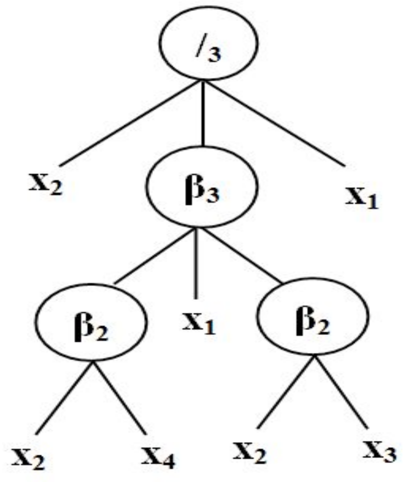
The used node set for creating an optimal FBBFNT model is$NS = \{\beta_2, \beta_3 / _3\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i (i = 1, 2, 3, 4) denotes $x(t)$, $x(t-6)$, $x(t-12)$, and $x(t-18)$, respectively. After 16 generations (G = 16) and 6,004,148 global number of function evaluations of the hybrid learning algorithm, an optimal FBBFNT model was obtained with RMSE 5.3430e-10. The RMSE value for validation data set is 1.8630e-09. The evolved FBBFNT_EIP&HBFOA, the actual time-series data and the output of FBBFNT model for training ant testing samples are shown in Fig. 4.

The FBBFNT_EIP&HBFOA evolving model is essentially compared with the FBONT model [23] and FBBFNT_EGP&OPSO [24] with the same initial parameter's values and number of generations (G = 16). The comparison is mainly based on the prediction error (RMSE) / Number of Function Evaluations (NFEs) compromise. In fact, for FBONT model [23], RMSE = 0.0076, NFEs = 2,934,112 and for FBBFNT_EGP&OPSO [24], RMSE = 0.0068, NFEs = 1,966,825. It is clear that FBBFNT_EIP&HBFOA significantly reduces the prediction error over the other two models, but with much greater number of function evaluations.

Other comparisons are also shown in Table 3. As observed, the FBBFNT_EIP&HBFOA achieves the lowest training and testing errors.

TABLE III. Comparison of different methods for the prediction of Mackey-Glass time-series.

Method	**Training error (RMSE)**	**Testing error (RMSE)**
HMDDE–BBFNN [11]	0.0094	0.0170
Aouiti [7]	-	0.013
Fuzzy&MRB [25]	0.000990	0.000884
CPSO [26]	0.0199	0.0322
HCMSPSO [27]	0.0095	0.0208
FNT [5]	0.0069	0.0071
FBONT [23]	0.0074	0.0076
FBBFNT_EGP& OPSO [24]	0.0061	0.0068
FBBFNT_EIP&HBFOA	**5.3430e-10**	**1.8630e-09**



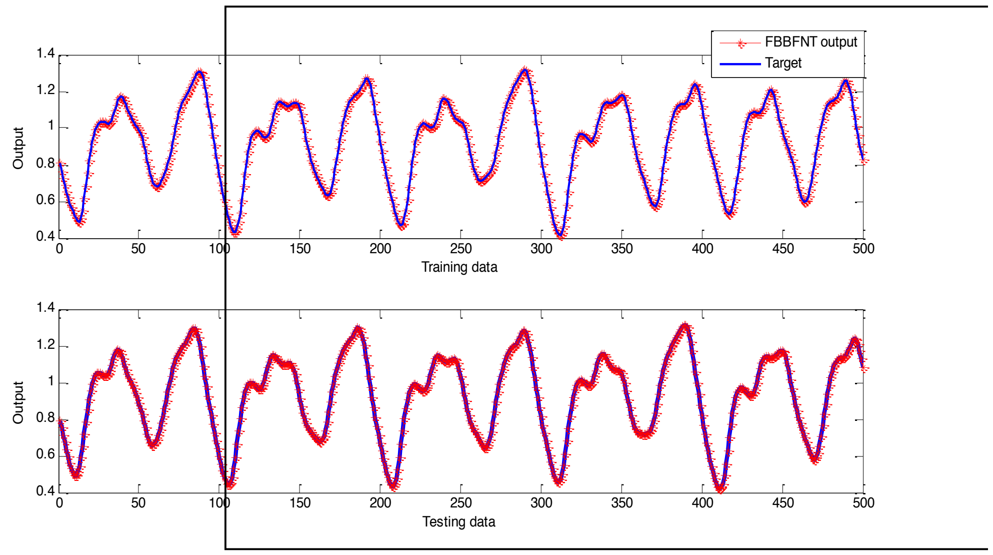
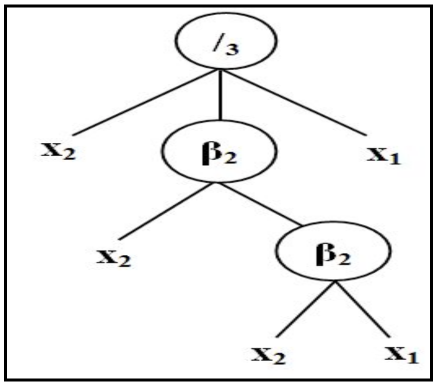


Fig. 4. Evolved FBBFNT_EIP&HBFOA architecture (left), The actual time series data and the output of the FBBFNT model for training and test samples (right) to forecast Mackey-Glass data.

B. *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 used as a benchmark example for testing prediction methods. 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 study, 200 data samples are used for training and the remaining data samples are used for testing the performance of the proposed model. The used instruction set is $NS = \{\beta_2\ /_3\} \cup \{x_1, x_2\}$, where x_i (i = 1, 2) denotes $y(t-1), u(t-4)$, respectively. After 22 generations (G = 22) of the learning algorithm, the optimal FBBFNT model was obtained with the RMSE 0.008026. The RMSE value for validation data set is 0.009121. The evolved FBBFN tree is shown in Fig. 5. The actual time-series data and the output of FBBFNT model are shown in Fig. 6. A comparison result of different methods for Jenkins-Box data prediction is shown in Table 4.



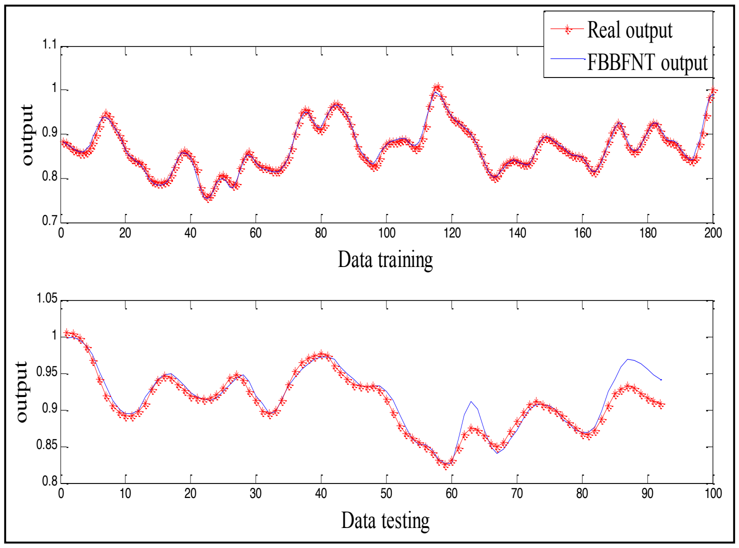


Fig. 5. The evolved FBBFNT for prediction of the Jenkins–Box time-series $(y(t-1), u(t-4))$.

Fig. 6. The actual time series data and the output of the FBBFNT model for training and test samples to forecast the Jenkins–Box time-series $(y(t-1), u(t-4))$.

TABLE IV. COMPARISON OF TESTING ERRORS OF BOX AND JENKINS.

Method	**Prediction error (RMSE)**
ANFIS model [29]	0.0845
FuNN model [30]	0.0714
FNN_AP&PSO [31]	0.0260
FNT [5]	0.0256
HMDDE [11]	0.2411
FBBFNT_EGP& OPSO [24]	0.011618

FBBFNT_EIP& HBFOA	**0.009121**

C. Example 3: Prediction of sunspot number time series

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 [32- 35], 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 $y(t)$. The used node set for the FBBFNT model is $NS = \{\beta_2, /_6\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i $(i = 1, 2, 3, 4)$ denotes $y(t-4), y(t-3), y(t-2)$ and $y(t-1)$, respectively.

After 26 generations of the evolution ($G = 26$), an optimal FBBFNT model was obtained with RMSE 1.9566e-10. The RMSE value for the first data set validation is 4.1519e-10 and for the second data set validation is 7.2714e-10. The evolved FBBFNT is shown in Fig. 7. The actual time-series data and the output of FBBFNT model for training and the two test cases are shown in Fig. 8. Table 5 illustrates the comparison of the proposed algorithm with other models according to the training and testing errors. As evident from Table 5, FBBFNT_EIP&HBFOA shows again the efficiencies for the sunspot number time series.

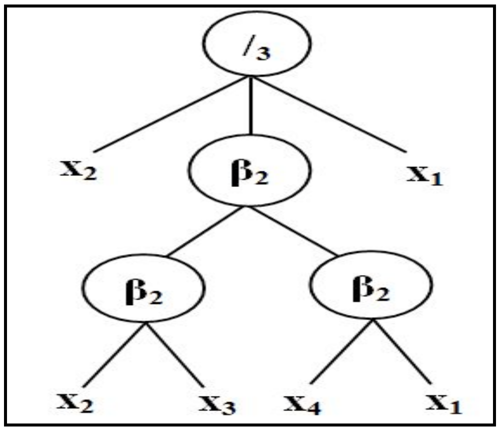


Fig. 7. The evolved FBBFNT for prediction of the sunspot number time-series.

TABLE V. COMPARISON OF DIFFERENT MODELS OF SUNSPOT TIME SERIES PREDICTION.

Model	RMSE Training	RMSE Testing 1	RMSE Testing 2
Transversal Net [32]	0.0987	0.0971	0.3724
Recurrent net [32]	0.1006	0.0972	0.4361
RFNN [33]	-	0.2749	0.6249
FWNN-M [34]	0.0828	0.0973	0.1988
ABC_BBFNN [35]	0.0012	0.0018	0.0044
FBBFNT_EGP&	**1.9566e-10**	**4.1519e-10**	**7.2714e-10**
HBFOA			

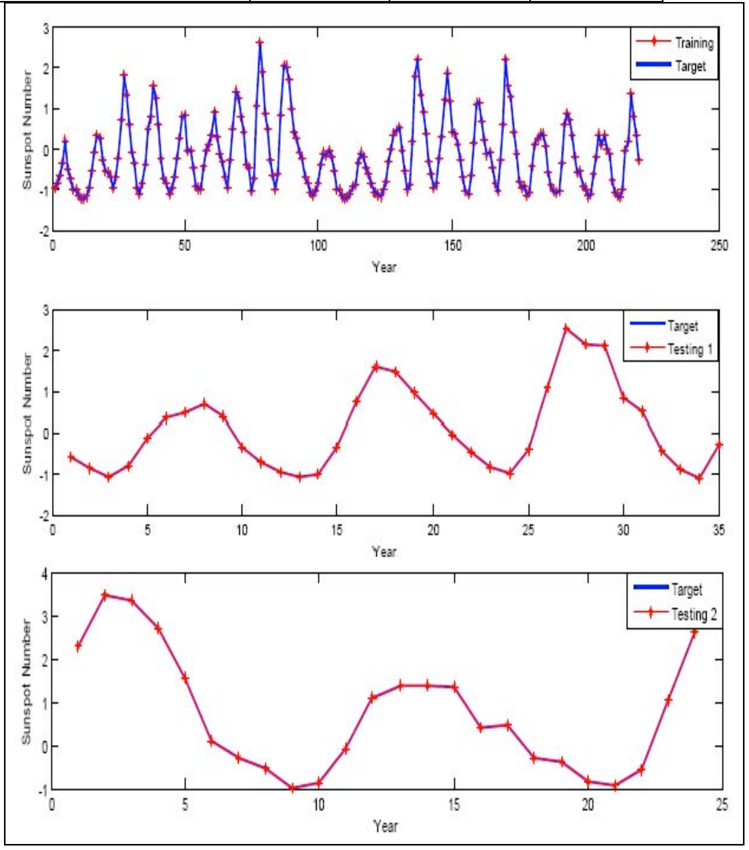


Fig. 8. The actual time series data and the output of the FBBFNT model for training, testing 1 and testing 2 to predict the sunspot number time-series.

CONCLUSION

In this paper, a hybrid learning algorithm based on the evolutionary computation is introduced to create and evolve the Flexible Beta Basis Function Neural Tree (FBBFNT) model. The proposed algorithm can successfully optimize simultaneously the structure and the parameters of the FBBFNT. The structure is developed using Extended Immune Programming (EIP). The Beta parameters and connected weights are optimized by the Hybrid Bacterial Foraging Optimization Algorithm (HBFOA). The results show that the FBBFNT_EIP&HBFOA method can effectively predict nonlinear systems such as Mackey-Glass chaotic time series, Jenkins–Box time series, and sunspot number time series.

ACKNOWLEDGMENT

The authors would like to acknowledge the financial support of this work by grants from General Direction of Scientific Research (DGRST), Tunisia, under the ARUB program. This work was also supported in the framework of the IT4 Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 by operational programme 'Research and Development for Innovations' funded by the Structural Funds of the European Union and state budget of the Czech Republic, EU.

REFERENCES

[1] X. Yao, “Influence of phosphorus runoff from agricultural areason enclosed sea downsteram”, Int. Symp. AI, Reasoning and Creativity, pp. 49–52, 1991.

[2] B.T. Zhang, P. Ohm, H. Miihlenbein, “Evolutionary induction of sparse neural trees”, Evolutionary Computation, vol. 5, pp. 213-236, 1997.

[3] Y. Chen, A. Abraham, B. Yang, “Feature Selection and Classification using Flexible Neural Tree”, Neurocomputing, vol. 70, pp. 305-313, 2006.

[4] Y. Chen, B. Yang, Q. Meng, “Small-time Scale Network Traffic Prediction Based on Flexible Neural Tree”, Applied Soft Computing, vol.12, pp. 274-279, 2012.

[5] Y. Chen, B. Yang, J. Dong, A. Abraham, “Time-series forecasting using flexible neural tree model”, Information Sciences, vol. 174, pp. 219–235, 2005.

[6] Y. Chen, F. Chen, J.Y. Yang, “Evolving MIMO Flexible Neural Trees for Nonlinear System Identification”, in International Conference on Artificial Intelligence, pp. 373-377, Nevada, USA, June 25-28, 2007.

[7] C. Aouiti, , A.M. Alimi, A. Maalej, “A Genetic Designed Beta Basis Function Neural Net-works for approximating of multi-variables functions”, in Proc. Int. Conf. Artificial Neural Nets and Genetic Algorithms Springer Computer Science, Prague, Czech Republic, pp. 383-386, 2001.

[8] 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, pp. 2564-2571, 2008.

[9] I.A.A. Al-Hadi, S.Z. M. Hashim and S. M. H. Shamsuddin, “Bacterial Foraging Optimization Algorithm for neural network learning enhancement”, in 11th International Conference on Hybrid Intelligent Systems (HIS), pp. 200–205, Malacca, Malaysia, December 5-8 2011. IEEE.

[10] 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 and Systems, vol. 154, pp. 251-274, 2005.

[11] H. Dhahri, A.M. Alimi, A. Abraham, “Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network”, Neurocomputing, vol. 79, pp.131-140, 2012.

[12] A.M. Alimi, “The Beta Fuzzy System: Approximation of Standard Membership Functions”, in Proc. 17eme Journees Tunisiennes d'Electrotechnique et d'Automatique: JTEA'97, Nabeul, Tunisia, vol. 1, pp. 108-112, 1997.

[13] A.M. Alimi, “The Beta System: Toward a Change in Our Use of Neuro-Fuzzy Systems”, International Journal of Management, Invited Paper, June, pp. 15-19, 2000.

[14] P. Musilek, A. Lau, M. Reformat and L. Wyard-scot, “Immune Programming”, Information Sciences, vol. 176, issue 8, pp. 972–1002, April 2006.

[15] K. Chellapilla, “Evolving computer programs without subtree crossover”, IEEE Transactions on Evolutionary Computation, vol. 1(3), pp. 209-216, 1998.

[16] K. M. Passino, “Biomimicry of bacterial foraging for distributed optimization and control”, IEEE Control Systems Magazine, vol. 22, pp. 52–67, 2002.

[17] S. Mishra, “A hybrid least square-fuzzy bacterial foraging strategy for harmonic estimation”, IEEE Transactions on Evolutionary Computation, vol. 9, no. 1, pp. 61– 73, February 2005.

[18] H.K. Dong and H.C. Chae, “Bacteria Foraging Based Neural Network Fuzzy Learning”, in Proceedings of the Indian International Conference on Artificial Intelligence, pp. 2030–2036, Pune India, December 20-22, 2005.

[19] I.A.A. Al-Hadi, S.Z.M. Hashim and S. M.H. Shamsuddin, “Bacterial Foraging Optimization Algorithm for neural network learning enhancement”, in 11th International Conference on Hybrid Intelligent Systems (HIS), pp. 200–205, Malacca-Malaysia, December 5-8, 2011.

[20] M. Geethanjali, V. Kannan and A. V. R. Anjana, “Bacterial foraging optimization algorithm trained ANN based differential protection scheme for power transformers”, in Proceedings of the Second international conference on Swarm, Evolutionary, and Memetic Computing - Volume Part II, SEMCCO’11, pp. 267–277, Berlin, Heidelberg, 2011. Springer-Verlag.

[21] Y. Jarraya, S. Bouaziz and A.M. Alimi. Adaptive Bacterial Foraging Swarm Optimization with Chemotactic Differential Evolution Algorithm. In Proceedings of International conference on Metaheuristic and Nature Inspired Computing (META’12), Port El-Kantaoiui, Sousse-Tunisia, October 27-31 2012.

[22] E.N. Lorenz, “Deterministic non-periodic flows”, J. Atm. Sci., vol. 20 pp. 130–141, 1963.

[23] S. Bouaziz, H. Dhahri, A.M. Alimi, “Evolving Flexible Beta Operator Neural Trees (FBONT) for Time Series Forecasting”, T. Hung et al. (Eds.) : 19th International Conference in neural information Processing (ICONIP’12), Proceedings, Part III, Series: Lecture Notes in Computer Science, Doha-Qatar, vol. 7665, pp. 17-24, 2012.

[24] S. Bouaziz, H. Dhahri, A.M. Alimi, A. Abraham, “A hybrid learning algorithm for evolving Flexible Beta Basis Function Neural Tree Model”, Neurocomputing, In Press-Corrected Proof, 2013.

[25] C.G. Coy and D. Kaur, “Improving evolutionary training for Sugeno Fuzzy Inference Systems using a Mutable Rule Base”, 2010 Annual Meeting of the North American, Fuzzy Information Processing Society (NAFIPS), pp. 1–6, 12-14 July, 2010.

[26] F. Van Den Bergh, A.P. Engelbrecht, “A cooperative approach to particle swarm optimization”, IEEE Trans. Evol. Comput., vol. 8, no. 3, pp. 225–239, Jun. 2004.

[27] C.F. Juang, C.M. Hsiao, C.H. Hsu, “Hierarchical Cluster-Based Multispecies Particle-Swarm optimization for Fuzzy-System Optimization”, IEEE Transactions on Fuzzy Systems, vol. 18(1), pp. 14-26, 2010.

[28] G.E.P. Box, G.M. Jenkins, “Time Series Analysis--Forecasting and Control”, Holden Day, San Francisco, CA, 1976.

[29] J. Nie, “Constructing fuzzy model by self-organising counter propagation network”, IEEE Transactions on Systems Man and Cybernetics 25, pp. 963–970, 1995.

[30] J.-S.R. Jang, C.-T. Sun, E. Mizutani, “Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence”, Prentice-Hall, Upper Saddle River, NJ, 1997.

[31] Y. Chen, B. Yang, J. Dong, “Evolving Flexible Neural Networks Using Ant Programming and PSO Algorithm”, International Symposium on Neural Networks (ISNN’04), Lecture Notes on Computer Science 3173, pp.211-216, 2004.

[32] J.R. McDonnell, D. Waagen, “Evolving recurrent perceptrons for time-series modeling”, IEEE Trans Neural Netw, vol. 5(1), pp. 24-38, 1994.

[33] 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 and Systems, vol. 160 (17), 2009.

[34] S. Yilmaz, Y. Oysal, “Fuzzy wavelet neural network models for prediction and identification of dynamical systems”, IEEE Transactions on Neural Networks, pp. 1599-1609, 2010.

[35] H. Dhahri, A.M. Alimi, “Designing Beta Basis Function Neural Network for Optimization Using Artificial Bee Colony (ABC)”, International Joint Conference on Neural Networks, Brisbane, pp. 2161-4393, 2012.