Extended Immune Programming and Opposite-based PSO for Evolving Flexible Beta Basis Function Neural Tree

Souhir BOUAZIZ, Adel M. ALIMI
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[1,2]
[1]Machine Intelligence Research Labs, WA, USA
[2]IT4Innovations, VSB-Technical University of Ostrava, Czech Republic
ajith.abraham@ieee.org

Abstract— **In this paper, a new hybrid learning algorithm based on the global optimization techniques, is introduced to evolve the Flexible Beta Basis Function Neural Tree (FBBFNT). The structure is developed using the Extended Immune Programming (EIP) and the Beta parameters and connected weights are optimized using the Opposite-based Particle Swarm Optimization (OPSO) algorithm. The performance of the proposed method is evaluated for time series prediction area and is compared with those of associated methods.**

Keywords—Extended Immune Programming; Opposite-based Particle Swarm Optimization; Flexible Beta Basis Function Neural Tree; Time series prediction

I. Introduction

Global optimization plays, in the last thirty years, a pivoting role in the development of modern science and engineering areas. The most famous global optimization algorithms are the meta-heuristic algorithms that can be defined as higher level frameworks aimed at efficiently and effectively exploring a search space. The meta-heuristic optimization techniques based on natural phenomena can be divided into two large fields: algorithms inspired from natural evolution (Evolutionary Algorithms: EA) and algorithms inspired from natural swarm (Swarm Intelligence: SI). The conventional representation of these algorithms is a fixed-length coded representation that can be severely limited to the simple structure systems. To overcome this limitation, a hierarchical computer program representation (S-expression or stack instruction string or tree representation) is imposed to improve the response of some meta-heuristic optimization techniques, such as Genetic Programming [1], Ant Colony Programming [2] and Immune Programming [3].

On the other hand, Artificial Neural Network (ANN) is considered as one of the complex structure systems and its performance is mainly dependent on its architecture. Therefore, several researchers used tree representation and evolutionary computation to design and optimize automatically the ANN structure [4-7]. A second major problem in designing of ANNs is the learning algorithms which can be successfully used for the training of ANNs. ANN parameters (weights and transfer function parameters) can be learned by various methods such as back-propagation algorithm [8], genetic algorithm [9], particle swarm optimization algorithm [10], etc. Recently, there has been an increasing interest to optimize the ANN structure and parameters simultaneously [5-7, 11, 12]. The most used transfer function is the Gaussian function. However, the Beta function [13, 14] shows its performance for typical representation of ANN against the Gaussian function due to its large flexibility and its universal approximation capacity [9, 13, 14]. The Beta function-based ANN is introduced by Alimi in 1997 [13] and is called Beta Basis Function Neural Network (BBFNN). For all these reasons, a tree-based encoding method is adopted, in this paper, to design the BBFNN and a hybrid algorithm which optimize the structure and parameters simultaneously, 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 the Opposite-based Particle Swarm Optimization (OPSO) algorithm.

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 time series prediction 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 [21] 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

Identify applicable sponsor/s here. *(sponsors)*

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:

- β_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).

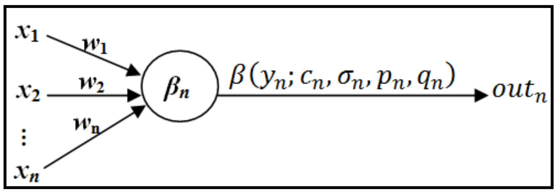

Fig. 1. A flexible Beta Basis Function.

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.

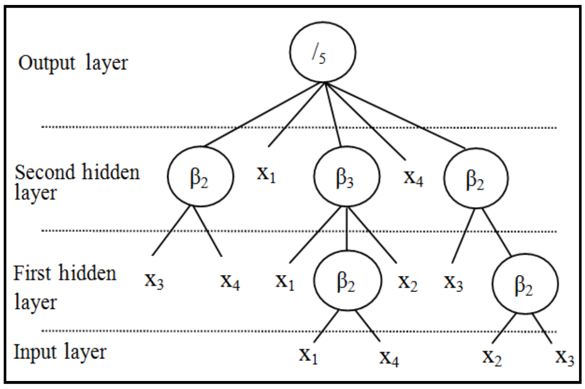

Fig. 2. A typical representation of FBBFNT: function node set F = $\{\beta_2, \beta_3, /_5\}$ and terminal node set T = $\{x_1, x_2, x_3, x_4\}$.

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 Opposite-based Particle Swarm Optimization (OPSO) algorithm.

A. The Extended Immune Programming for structure optimization

Based on the results found by Musilek et al. in [31], 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 Algorithm (EIP) and it is formed by seven main steps:

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) Evalution: 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 Abi of the current population is selected to be treated; if its affinity is heigher than a radom genereted number so this antibody can be coloned with a probability Pc, and placed in the new population.

4) Mutation: if a selected high-affinity antibody in the previous step has not been cloned due to the stochastic character of the cloning process, it is submitted to hypermutation. 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 following: (1) Define a number M which represents a sample from a Poisson random variable, (2) Select randomly M mutation operators from above four mutation operator set, (3) 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, hypermutation 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, hypermutation, replacement, iteration–repertoire) until a terminal criterion is reached.

B. Opposite-based Particle Swarm Optimization for parameter Optimization

PSO was proposed by Kennedy and Eberhart [16] 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 the 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). The OPSO steps are described as follows:

1) Initialization: At iteration t = 0, the initial positions $x_i(0)$ (i = 1, …, *NP*) which are *NParam* × *NN arrays*, are generated uniformly distributed randomly; with *NP* is the number of particles, *NParam* is the number of parameters (Beta parameters and weights) and *NN* is the number of FBBFNT nodes:

$$x_i(0) = a_j + rand_i(b_j - a_j) \quad (4)$$

Where $[a_j, b_j]$ is the search space of each parameter.
Generate the opposite population as follows:

$$\overline{x}_i(0) = \begin{cases} \alpha_i\left(x_i(0) + \frac{a_j+b_j}{2}\right), \; if\; x_i(0) < \frac{a_j+b_j}{2} \\ \alpha_i\left(x_i(0) - \frac{a_j+b_j}{2}\right), \; if\; x_i(0) > \frac{a_j+b_j}{2} \end{cases}, \alpha_i \in \left]0,1\right[\quad (5)$$

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

2) Particle evaluation: Evaluate the performance of each particle in the population using a fitness function (section *C*).

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

$$v_i(t+1) = \Psi(t)\, v_i(t) + c_1\varphi_1\big(p_i(t) - x_i(t)\big) + c_2\varphi_2\left(p_g(t) - x_i(t)\right) \quad (6)$$

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 $[-v_{max}, +v_{max}]$.

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

$$x_i(t+1) = x_i(t) + \big(1 - \Psi(t)\big)\, v_i(t+1) \quad (7)$$

$$\overline{x}_i(t+1) = \overline{x}_i(t) + \big(1 - \Psi(t)\big)\, v_i(t+1) \quad (8)$$

5) 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.

6) End criterion: The OPSO learning process ends when a predefined criterion is met, otherwise the iteration number (*OPSO_Iter* = 1 during initialization) is incremented, *OPSO_Iter* = *OPSO_Iter* + 1 and the algorithm proceeds through steps 2–5. In this study, the criterion is the goal or total number of OPSO iterations.

C. Fitness Function

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

$$Fit(i) = \sqrt{\frac{1}{P}\sum_{j=1}^{P}(y_t^j - y_{out}^j)^2} \quad (9)$$

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.

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 OPSO 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 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); *GlobalIter* = *GlobalIter* + *OPSO_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 benchmark problems: Mackey-Glass chaotic, Jenkins–Box, sunspot number and Lorenz chaotic time series. After many experiences of the system parameters, the chosen parameters to be used for all problems are as listed in table 1. The complexity of the proposed method is determined by the number of Function Evaluations: NFEs.

A. Example 1: Mackey–Glass time series prediction

A time-series prediction problem can be constructed based on the Mackey–Glass [17] differential equation:

$$\frac{d(x(t))}{dt} = \frac{ax(t-\tau)}{1+x^{c}(t-\tau)} - bx(t) \tag{10}$$

The setting of the experiment varies from one work to another. In this work, the same parameters of [5], [12], and [18] 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 are used to validate the model identified.

TABLE I. 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
OPSO	
Parameter	***Initial value***
Population size (*NP*)	50
Maximum iteration number	4000
c_1	0.2
c_2	0.6
Hybrid evolving algorithm	
Parameter	***Initial value***
Maximum global iteration number	40 000

The used node set for creating an optimal FBBFNT model is $NS = \{\beta_2, \beta_3, /_5\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i (i = 1, 2, 3, 4) denotes $x(t), x(t-6), x(t-12), x(t-18)$, respectively. After 18 generations (G = 18) and 1,616,648 global NFEs of the hybrid learning algorithm, an optimal FBBFNT model was obtained with RMSE 0.004194. The RMSE value for validation data set is 0.004299. The evolved FBBFNT, and the actual time-series data and the output of FBBFNT model for training ant testing data are shown in Fig. 4.

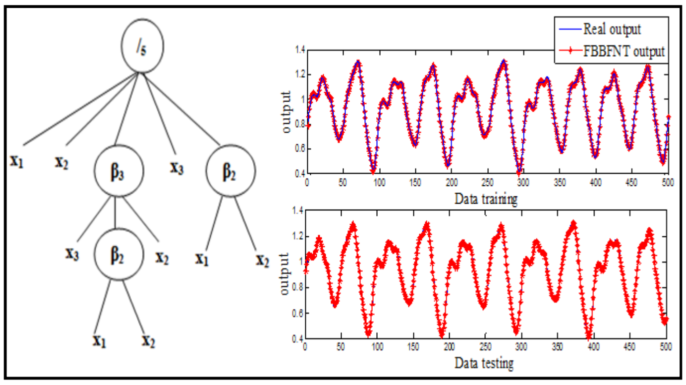

Fig. 3. The evolved FBBFNT (left), and the actual time series data and the output of the FBBFNT model for training and test samples (right) to forecast Mackey-Glass data.

The proposed method is essentially compared with Hierarchical multi-dimensional differential evolution for the design of Beta basis function neural network (HMDDE-BBFNN) [12], the FNT model with Gaussian function, PIPE for structure creation and simulated annealing for parameter optimization [5], the flexible Beta Basis Function neural tree (FBBFNT) model with EGP for structure optimization and OPSO for parameter optimization [18], and also with other systems.

The NFEs of FBBFNT_EGP&PSO is equal to 2,311,823 but it isn't mentioned in other systems. The number of function hidden units of FBBFNT_EGP&PSO is equal to six units. A comparison result of different methods for forecasting Mackey-Glass data is shown in Table 2. As observed, the FBBFNT_EIP&OPSO achieves the lowest training and testing errors. In addition, FBBFNT_EIP&OPSO uses less number of function evaluations as well as less number of function hidden units compared to FBBFNT_EGP&PSO.

TABLE II. COMPARISON OF DIFFERENT METHODS FOR THE PREDICTION OF MACKEY-GLASS TIME-SERIES.

Method	Training error (RMSE)	Testing error (RMSE)
HMDDE–BBFNN [12]	0.0094	0.0170
Aouiti [9]	-	0.013
Classical RBF [19]	0.0096	0.0114
CPSO [20]	0.0199	0.0322
HCMSPSO [21]	0.0095	0.0208
FNT [5]	0.0069	0.0071
FBONT_EGP& PSO [18]	0.0074	0.0076
FBBFNT_EIP& OPSO	**0.0041**	**0.0042**

B. *Example 2 : Box and Jenkins' Gas Furnace Problem*

The gas furnace data of Box and Jenkins [22] 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 $NS = \{\beta_3, /_3\} \cup \{x_1, x_2\}$, where x_i ($i = 1, 2$) denotes $y(t-1), u(t-4)$, respectively.

After 22 generations ($G = 22$) and 936,031 global NFEs of the learning algorithm, the optimal FBBFNT model was obtained with the RMSE 0.008796. The RMSE value for validation data set is 0.009812. The evolved FBBFNT and the actual time-series data and the output of FBBFNT model are shown in Fig. 5. A comparison result of different methods for Jenkins-Box data prediction is shown in Table 3.

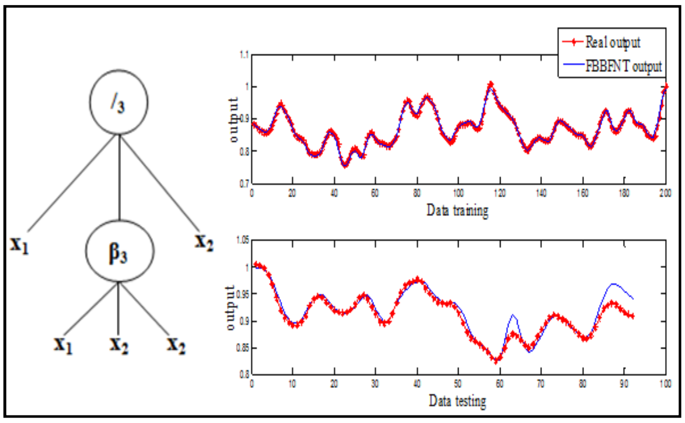

Fig. 4. The evolved FBBFNT for prediction of the Jenkins–Box time-series $(y(t-1), u(t-4))$.

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

Method	Prediction error (RMSE)
ANFIS model [23]	0.0845
FuNN model [24]	0.0714
FNN_AP&PSO [25]	0.0260
FNT [5]	0.0256
HMDDE [12]	0.2411
FBONT_EGP& PSO [18]	0.0116
FBBFNT_EIP& OPSO	**0.00981**

C. *Example 3: Prediction of sunspot number time series*

The current example presents the series of the sunspot annual average numbers which show the yearly average relative number of sunspot observed [26-30]. The data points between 1700 and 1920 are used for the FBBFNT training and for the test two sets are used the first one is from 1921 to 1955 and the second is from 1956 to 1979. The $y(t-4), y(t-3), y(t-2), y(t-1)$ are used as inputs to the FBBFNT_EIP&OPSO in order to predict the output $y(t)$.

The used node set for the FBBFNT model is $NS = \{\beta_3, /_6\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i (i = 1, 2, 3, 4) denotes $y(t-4), y(t-3), y(t-2), y(t-1)$, respectively.

After 26 generations ($G = 26$) and 864,703 global NFEs of the evolution, an optimal FBBFNT model was obtained with RMSE 4.2541e-13. The RMSE value for the first data set validation is 1.0589e-12 and for the second data set validation is 7.6715e-13. 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. 6. Table 4 illustrates the comparison of the proposed algorithm with other models according to the training and testing errors. As evident from Table 4, FBBFNT_EIP&OPSO shows again the efficiencies for the sunspot number time series.

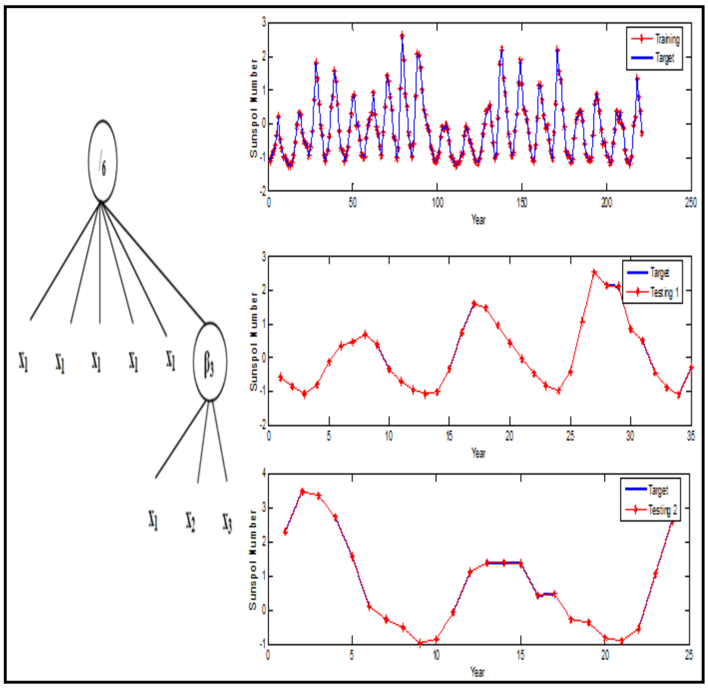

Fig. 5. The evolved FBBFNT (left), and the actual time series data and the output of the FBBFNT model for training, testing 1 and testing 2 to predict the sunspot number time-series.

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

Model	RMSE Training	RMSE Testing 1	RMSE Testing 2
Transversal Net [27]	0.0987	0.0971	0.3724
Recurrent net [27]	0.1006	0.0972	0.4361
RFNN [28]	-	0.074	0.21
FWNN-R [29]	0.0796	0.1099	0.2549

FWNN-M [29]	0.0828	0.0973	0.1988
ABC_BBFNN [26]	0.0012	0.0018	0.0044
FBBFNT_EIP& OPSO	**4.2541e-13**	**1.0589e-12**	**7.6715e-13**

V. 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. In fact, the structure is developed using Extended Immune Programming (EIP) and the Beta parameters and connected weights are optimized by the Opposite-based Particle Swarm Optimization algorithm (OPSO). The results show that the FBBFNT_EIP&OPSO method can effectively predict time-series problems such as Mackey-Glass chaotic, Jenkins–Box, sunspot number and Lorenz chaotic 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] J. Koza, "Genetic Programming: On the programming of Computers by Means of Natural Evolution", MIT Press: Cambridge, 1992.

[2] M. Boryczka, "Ant Colony Programming for Approximation Problems", Proceedings of the Eleventh International Symposium on Intelligent Information Systems, Sopot, Poland, pp. 147-156, June 3–6 2002.

[3] M. Petr, L. Adriel, R. Marekt, W.-S Loren, "Immune Programming", Information Sciences, vol. 176, issue 8, pp. 972–1002, April 2006.

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

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

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

[7] 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.

[8] S.W Stepniewski, A.J Keane, "Pruning Back-propagation Neural Networks Using Modern Stochastic Optimization Techniques", Neural Computing & Applications, vol. 5, pp. 76-98, 1997.

[9] C. Aouiti, , A.M. Alimi, A. Maalej, "A Genetic Designed Beta Basis Function Neural Net-works 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.

[10] H. Dhahri, A.M. Alimi, , F. Karray, "Designing beta basis function neural network for op-timization using particle swarm optimization", IEEE International Joint Conference on Neural Networks, Hong Kong, China, pp. 2564-2571, 2008.

[11] 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.

[12] 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.

[13] A.M. Alimi, "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.

[14] A.M. Alimi, "The Beta System: Toward a Change in Our Use of Neuro-Fuzzy Systems", In-ternational Journal of Management, Invited Paper, June, pp. 15-19, 2000.

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

[16] J. Kennedy, R.C. Eberhart, "Particle swarm optimization", Proceedings of the IEEE International Conference on Neural Networks. IEEE Press, Piscataway, NJ, vol. 4, pp. 1942–1948, 1995.

[17] M. Lzvbjerg, T. Krink, "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.

[18] 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.

[19] K.B. Cho, B.H. Wang, "Radial basis function based adaptive fuzzy systems their application to system identification and prediction", Fuzzy Sets and Systems 83, pp. 325–339, 1995.

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] A. Hussain, "A new neural network structure for temporal signal processing", Proc Int Conf on Acoustics Speech and Signal Processing, pp. 3341-3344, 1997.

[30] 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.

[31] P. Musilek, A. Lau, M. Reformat and L. Wyard-scott, "Immune programming. Information Sciences", vol. 176, no. 8, pages 972–1002, 2006.