Neurocomputing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Neurocomputing

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

Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network

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
[b] Faculty of Electrical Engineering and Computer Science, Technical University of Ostrava, Czech Republic
[c] Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, WA, USA

ARTICLE INFO

Article history:
Received 15 December 2011
Received in revised form
28 March 2012
Accepted 6 April 2012
Communicated by V. Palade

Keywords:
Hierarchical multi-dimensions differential evolution
Beta basis function neural networks
Time series prediction
Identification system

ABSTRACT

This paper proposes a hierarchical multi-dimensional differential evolution (HMDDE) algorithm, which is an automatic computational frame work for the optimization of beta basis function neural network (BBFNN) wherein the neural network architecture, weights connection, learning algorithm and its parameters are adapted according to the problem. In the HMDDE-designed neural network, the number of individuals of the population multi-dimensions is the number of beta neural networks. The population of HMDDE forms multiple beta networks with different structures at the higher level and each individual of the previous population is optimized at a lower hierarchical level to improve the performance of each individual. For the beta neural network consisting of m neurons, n individuals (different lengths) are formed in the upper level to optimize the structure of the beta neural network. In the lower level, the population within the same length is to optimize the free parameters of the beta neural network. To evaluate the comparative performance, we used benchmark problems drawn from identification system and time series prediction area. Empirical results illustrate that the HMDDE produces a better generalization performance.

1. Introduction

The neural networks have been effectively applied in many areas, such as time series prediction [21,22,24], pattern recognition [27], approximation function [28], etc. Among the variety of artificial neural networks, the beta basis function neural network (BBFNN) represents an interesting alternative in which we can approximate any function [1]. The BBFNN network is a three layer feed-forward network that generally uses a linear transfer function for the output units and a non linear transfer function (the beta function) for the hidden units. In spite of a number of advantages of BBFNN such as better approximation capabilities [30], faster learning algorithms and simple network topologies; especially the determination of the optimal number of hidden nodes is the most critical task. The development of BBFNN still involves difficulties in optimizing the topology of the network structure (the number of nodes). Today, hybridization in soft computing is becoming a promising research field of computational intelligence focusing on synergistic combinations of multiples soft computing methodologies an intelligent system. In order to overcome the soft computing method [2–4], the investigation of hybrid approaches will be necessary. In particular, in order to overcome the challenge in developing the neural network, the evolutionary algorithm is applied to optimize the structure of the neural network system. There are several works that deal the problem of automatic neural network design [5–8].

Methods found in literature can be divided into two classes:

(1) Methods in which the number of nodes must be given a priori; after this, computing the remaining neural networks' parameters (the centers and the widths) is done choosing randomly from the training set or performing any kind of clustering method with them [9]. In this case, the optimal results cannot be guaranteed, and often consumes time to find the acceptable results.
(2) Methods that automatically find the number of nodes and the remaining parameters. Several techniques have been proposed for this, such us the pruning algorithm [10–11], the growing algorithm [12–14] and the evolutionary algorithms [15]. The applying of evolutionary algorithms to construct neural nets is also well known in the literature. The most representative algorithms include Genetic Algorithms (GA) [16,39,40], Particle Swarm Optimization (PSO) [18,19,23,26], Flexible Neural Trees [41,42] and the method of Differential Evolution (DE) [17,20,22,24,25]. The DE algorithm has been shown to perform better than the Genetic Algorithm (GA) or the Particle Swarm Optimization (PSO) over several numerical benchmarks [26,29].

* Corresponding author. Tel.: +216 97 473 672.
E-mail addresses: habib.dhahri@ieee.org, habibdhahri@yahoo.fr (H. Dhahri), adel.alimi@ieee.org (A.M. Alimi), ajith.abraham@ieee.org (A. Abraham).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.04.008

The major drawback of the DE variants including the basic method is that they can only be applied to a search space with fixed dimensions. However, in the context of BBFNN, the optimum dimension (the optimum nodes number) is unknown and should be determined within the DE process. In order to address this problem, in this work, we present a multi-dimensional DE (MDDE), which negates the need of fixing the dimension of the solution space in advance. Therefore, no assumption is made about the number of nodes. In this paper; we propose a hierarchical differential evolution for the design of the based beta basis function neural network. In the upper level, the MDDE focuses on determining the optimum dimension that naturally corresponds to near-optimal BBFNN architecture. At the lower level, we use the Differential Evolution with a fixed dimension to search the others beta neural parameters (centers, widths and forms parameters). The weight parameters that connect the hidden layer with the output layer are determined by computing the pseudo-inverse matrix. In order to validate the performance of the proposed scheme, the developed BBFNN are used to predict four benchmarks examples. The comparison of the proposed method with the others existing results, in the bibliography, we find that HMMDE–BBFNN can predict the time series satisfactorily with the automatically selection of the model structure.

The remainder of this paper is organized as follow: Section 2 reviews some approaches to train the BBF neural network. Section 3 introduces hierarchical differential evolution for the design of BBFNN, while Section 4 provides a set of experimental results to predict the four time series prediction. Finally, the work is concluded in Section 5.

2. Related work

This section first describes the basic concept of the BBFNN to be designed in this study. The basic concept of DE employed in BBFNN optimization is then described.

2.1. The beta basis function neural network (BBFNN)

In this section, we want to introduce the beta basis functions neural network that will be used in the remainder of this paper. The BBFNN usually consists of three layers [1]: the input layer, the BBF layer (hidden layer) and the output layer. The input layer simply transfers the input vector $x=[x_1,x_2,...,x_n]^T$ through scalar weights to the next layer. Thus the whole input vector appears to each neuron in the hidden layer. Each hidden nodes perform the beta basis function over the incoming vector that appears at the input of each BBFNN neuron. The output layer yields a vector $y=[y_1,y_2,...,y_m]^T$ for m outputs by linear combination of the outputs of the hidden nodes to produce the final output. Fig. 1 presents the structure of a single output of BBF network; the network output can be obtained by

$$y=f(x)=\sum_{i=1}^{n} w_i B_i(x,c,\sigma,p,q), \quad 1\le i\le n \tag{1}$$

Besides the center c, the beta basis function may also present a width parameter σ, which can be seen as a scale factor for the distance $(x-c)$ and the parameter forms p and q.

The BBF network can be regarded as feed-forward neural network with a single layer of hidden units, whose responses are the outputs of the beta basis functions. Fig. 2 shows the effect of parameters forms to the beta function. The latter $B_i(x,c_i,\sigma_i,p_i,q_i)$, i=1,…,$n$, is defined by:

$$\beta(x)=\begin{cases}\left[1+\frac{(p+q)(x-c)}{\sigma p}\right]^p\left[1-\frac{(p+q)(c-x)}{\sigma q}\right]^q & \text{if } x\in]x_0,x_1[\\ 0 & \text{else}\end{cases} \tag{2}$$

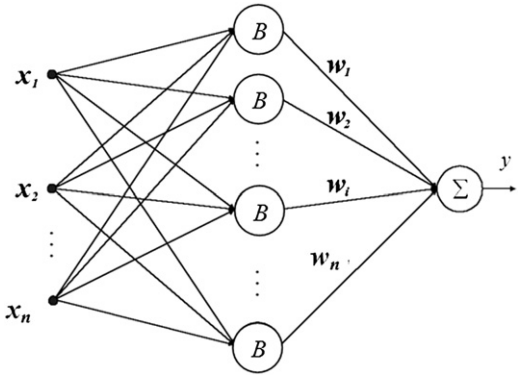

Fig. 1. Architecture of the BBFNN.

where $p>0$, $q>0$, x_0 and x_1 are the real parameters, such as $x_0<x_1$ and

$$c=\frac{px_1+qx_0}{p+q} \tag{3}$$

In the multi-dimensional case, the beta function is defined by

$$\beta(c,\sigma,p,q)(x)=\prod_{i=1}^{i=d}\beta_i(c_i,\sigma_i,p_i,q_i)(x) \tag{4}$$

where d is the dimension of the beta kernel.

The BBF neural network is usually trained to map a vector $x_k\in R^n$ into vector $y_k\in R^{n0}$ where the pairs (x_k,y_k), $1\le k\le M$ from the training set and R is set of real numbers. If this mapping is viewed as a function in the input space R^n, learning can be seen as a function approximation problem. According to this point of view, learning is equivalent to finding the surface in a multi-dimensional space that provides the best fit to the training data. Generalization is therefore synonymous with interpolation between the data points along the constraint surface generated by the fitting procedure as the optimum approximation to this mapping.

Alimi is the first to investigate the use of the beta basis function in the design of neural network as activation functions in artificial neural network [1]. In [30], the authors proved that BBF networks with one hidden layer are capable of universal approximation. Nevertheless, the BBF networks are capable of approximating arbitrarily well any function; also have the best approximation property.

The performance of the BBF neural network depends on the number of units of beta basis functions, their shapes, the parameters forms, and the method used to determine the associative weight matrix. Simon [31] classified the existing learning strategies for neural network as follows: (1) learning with a fixed number of units and centers selected randomly from the training data; (2) supervised learning for the selection of the centers of the network; and (3) unsupervised learning for the selection of the fixed number of units. In this paper, we are going to use the second strategy.

One of the main problems related to the development of neural network based system is the application of suitable learning algorithm to adjust the network parameters. The BBF network presents the following adjustable parameters: the position of BBFs centers c_i, the widths σ_i of the BBFs, the form parameters of the BBFs p_i and q_i and the output weights w_i.

There are a number of proposals on how to define these parameters in the literature. One first idea is to fix the number of nodes and use a gradient descent method to adjust the parameters [32], in a manner very similar to the error back-propagation

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

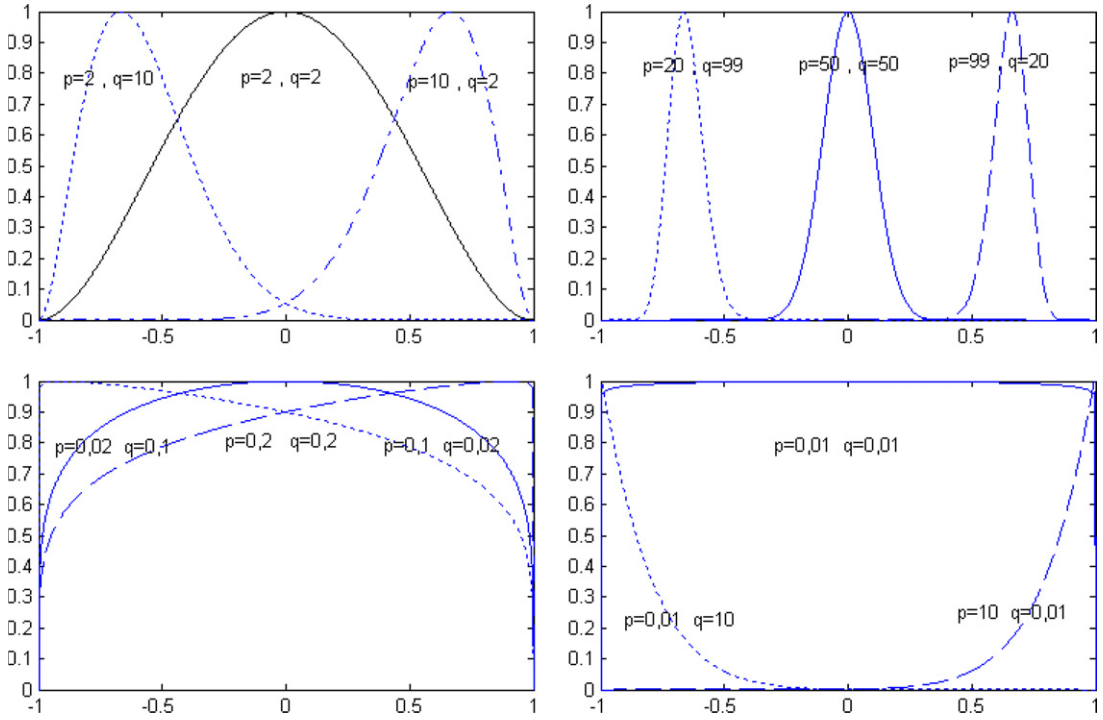

Fig. 2. The Beta plot in one dimension.

algorithm, often used with MLPs. Nevertheless, training the BBF network in such a way seems somewhat wasteful. There are several interesting approaches that exploit this potential. Although slightly different, all of them share the same idea: the definition of the hidden layer is considered as the major task, since the output weights can be computed according to linear optimization techniques [32]. In [33] the authors used the constructive method that allows BBF neural network to grow by inserting new units in the feature space where the mapping needs more details. In [17,18,20], the major task considered in these works is to optimize the beta parameters with a fixed number of nodes. Consequently the problem of determination of BBF architecture will be treated in this paper.

2.2. The basic DE algorithm

The discovery of differential evolution was introduced by Storn [17] as a population based stochastic search process. Like other evolutionary algorithms, DE is very effective for solving optimization problems with non-smooth objective functions. The DE algorithm was successfully applied in the optimization of some well-known nonlinear, non-differentiable and non-convex functions. DE uses a greedy and less stochastic approach with floating point coding in problem solving rather than the other evolutionary algorithms, such as genetic algorithms, evolutionary programming and evolution strategies. In DE, a candidate solution for a specific problem is called an individual or a chromosome and consists of a linear list of genes. Each individual represents a point in the search space, and hence a possible solution to the problem. A population consists of a finite number of individuals. Each individual is decided by an evaluating mechanism to obtain its fitness value. Based on this fitness value and undergoing DE operators, a new population is generated iteratively with each successive population referred to as a generation. The fitness of an offspring is one-to-one competed with that of the corresponding parent in DE. The DE algorithm uses simple arithmetical operators with the classical operators of recombination, mutation and selection to evolve from a randomly generated starting population to a final solution. Basically, the weighted difference between two individuals is added to a third individual in a population. This way, no separate probability distribution has to be used, which makes the scheme completely self-organizing.

The potentialities of DE are its simple structure, easy use and local searching property. This drawback could be overcome by employing a larger population. However, by doing so, much more computation time is required to estimate the fitness function.

Accordingly, the general pseudo-code of DE algorithm can be given as follows:

Step 1: Parameters setup:

Choose the parameters of population size, the boundary constraints of optimization variables, the mutation factor (F), the crossover rate (Cr), and the stopping criterion of the maximum number of generations.

Step 2: Initialization of the population

The initial population is generated uniformly distributed randomly;

For $j=1$ to NP do

$$P_{ij} = a_j + rand_i(b_i - a_i) \tag{5}$$

end

Step 3: Research of the best individual

While convergence criteria not yet met

For i de 1 to NP do

Select three parents Pr1, Pr2 randomly and Pbest from the current population where $r1 \neq r2$:

$$V_i = P_{best} + F(P_{r1} - P_{r2}) \tag{6}$$

where F is real number in [0,1]

For $j=1$ to d do

$$U_{ij} = \begin{cases} V_{ij}, & if \quad rand(0,1) \leq Cr \\ P_{ij} & otherwise \end{cases} \tag{7}$$

end

Evaluate the trial vectors U_i

$$if\ (f(U_i) \leq f(P_i)) \tag{8}$$

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

$P_i = U_i$
end
Select the fittest individual from P
End

3. The HMDDE technique for automatic BBFNN design

In this section, we will first introduce the MDDE technique, which presents an important improvement over DE classic variant. The next unit explains its use for the BBFNN.

3.1. MDDE algorithm

Instead of operating at a fixed dimension N, the MDDE algorithm is designed to seek the optima dimension. In the DE algorithm, the main idea is to construct, at each generation a mutant vector. This mutant vector is constructed through a specific mutation operation based on adding difference between randomly selected elements of the population to another element. For example one of most used variant to construct a mutant vector, x, starting from a current population is based on the following rule: $x = x_1 + F\cdot(x_2 - x_3)$ where $x_1 \in IR^{n1}$, $x_2 \in IR^{n2}$and $x_3 \in IR^{n3}$ are selected from the m individuals of the population and $F > 0$ is a scaling factor. In the multi-dimensions of DE, we select three chromosomes from the population to generate a new individual to add diversity to the population and to provide mechanism to favor the exploration of the search space. The difference based on the two vectors x_2 and x_3 is not allowed because these vectors are not in the same dimension. This opens the question of exactly how to create the new individual from chromosomes with different lengths.

In order to make this hypothesis feasible, we must take into account if we pass from the dimension i to the dimension j $(i < j)$, we add $(j - i)$ axis of supplementary coordinates that will be orthogonal to the i axes. In other words, we project the vectors on the space where the dimension is the maximum size of the three vectors.

Fig. 3 shows a sample of projection operator P which can be expressed as follows:

$$P(x_i) = [x_i \cdot zeros(1, max(x_1, x_2, x_3) - l(x_i))], \quad 1 \le i \le 3 \tag{9}$$

Where $[\cdot]$ is a vector, *zeros* is zeros array, *max* is the maximum of the three vectors and l is the length of the vector x_i. Once the projection P operator is applied, the classical variant of differential evolution DE is used.

3.2. MDDE for BBFNN

As depicted in Fig. 4, we conceived a hierarchical differential evolution algorithm for the design of the beta basis function neural network composed by two levels. The upper level is composed by heterogeneous DE (different sizes) and the lower level by homogenous DE (same size). At the higher level, as a stochastic search process in multi-dimensional search space, MDDE seeks for optimal networks in architecture space. The MDDE is used to build the BBFNN description or net topology, i.e., to choose the number of neurons. This level handles the task of updating the population for neural network–neuron optimization. The whole population of differential evolution algorithm presents all networks configurations. Each individual of the population defines a beta basis function neural network. Once optimizing the topology of BBFNN, the best individual of the higher population will be sent to DE at the lower level to optimize the other neural parameters, i.e., centers widths and form parameters. At the second level, the DE algorithm is used to train BBFNN by adjusting the neural parameters with individuals with the same size. After a predefined number of generations, the inner DE returns the best individuals that represent the optimal configuration of BBFNN. Accordingly, the HMDDE method can be expressed as follows:

Step (1): Initialize the DE parameters.
Step (2): Encode all the parameters into the chromosomes using the proposed encoding scheme.
Step (3): Initialize population pop randomly of NP individuals.
Step (4): Compute the connection weights by pseudo-inverse technique.
Step (5): Find the fittest individual. The fitness evaluation is computed as follows: $F = \alpha f_1 + \beta f_2 \quad \alpha, \beta \in [0, 1]$ where f_1 measures the performance of BBF neural network on the training data (e.g., root mean square error: RMSE), f_2 measures the complexity of BBF neural network and α, β are a user specified fitness coefficient that allow a trade-off between the objectives, specifically, $f_2 = N_n / N_{max}$, where N_n is the actual number

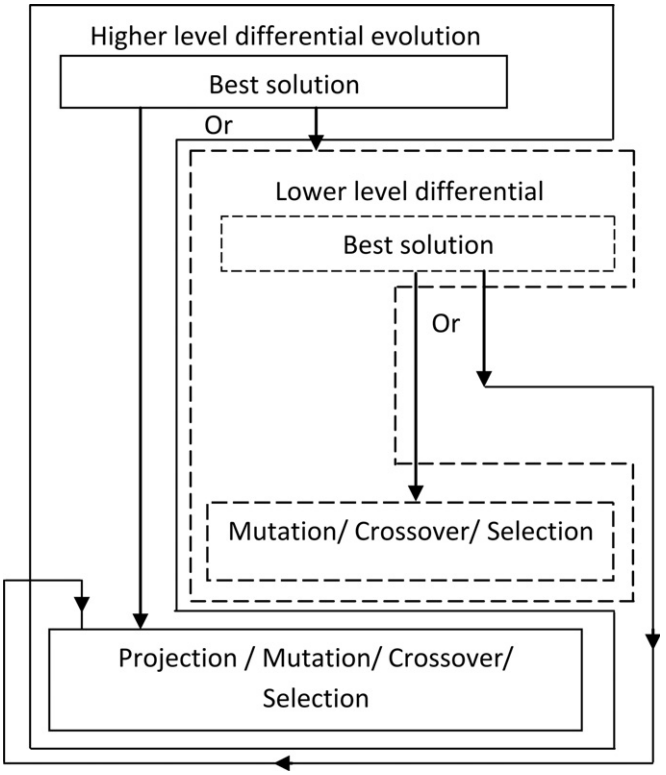

Fig. 4. Hierarchical multi-dimensional differential evolution.

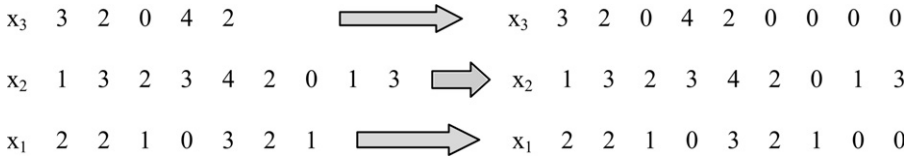

Fig. 3. Projection operator in one dimension.

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

of nodes and N_{max} is the maximum number of nodes. (In the simulation, we use $\alpha=0.95$ and $\beta=0.05$.)If the specified number of iterations *iterN1* exceeded in the upper level go to Step (a) else go to Step (6).

Step (a): Choose randomly the initial population; each individual of this population represents a candidate BBF neural network where the number of neurons is equal to one of the best solution in Step (5).

Step (b): Encode all the parameters into the chromosomes.

Step (c): Generate a population *pop* randomly.

Step (d): For each candidate of the population, select the random variables $r_1,r_2,r_3 \in \{1,2,...,NP\}$.

Step (e): Apply mutation operator to each candidate in the population Eq. (6).

Step (f): Apply crossover operator that each vector from the population is recombined with a mutant vector to produce a trial vector Eq. (7).

Step (g): Apply a selection operator according to Eq. (8).

Step (h): Compute the connection weights by pseudo-inverse technique.

Step (i): Find the fittest individual by RMSE performance.

Step (j): If the performance of the best individual is less than the predefined goal then go out else return to Step (d).

Step (6): For each candidate of the upper population (higher level), select the random variables $r_1,r_2,r_3 \in \{1,2,...,NP\}$.

Step (7): Apply a projection operator specifically by Eq. (9).

Step (8): Apply the step from Step (d) until Step (i).

Step (9): If the performance of the best individual is less than the predefined goal then go out else return to Step (6).

3.3. Encoding scheme of BBFNN

Once applying the DE algorithm to design the BBFNN network, the main key is to encode the BBF neural network into the chromosome with an efficient approach. In order to define the BBFNN, the topology and the network parameters should be specified. In the proposed encoding scheme consists simply of the second parts (network parameters) because the first part (neurons number) is determined at the upper level of HMDDE. Here, we adopt the real coded DE and each sequence of neural parameters represents one node. Each chromosome represents a candidate BBFNN neural network. Since the weights parameters are computed by the pseudo-inverse technique, therefore it is only necessary to encode the four parameters, i.e., centers c_i, widths σ_i, and form parameters p_i and q_i which are necessary to represent the beta form of BBF.

4. Experimental results

In order to evaluate the performance of the proposed algorithm, a number of simulations studies are carried out for various benchmark problems. The aim is to test the "optimality" of BBF neural network and the "generalization" ability whilst performing comparative evaluations against several popular techniques with the latter. These problems are Mackey-Glass, identification and control problem and Box–Jenkins time series; these problems are taken from literature in order to be able to make a direct performance comparison.

For comparison of the proposed algorithm with the basic PSO and relevant modified PSO algorithm using the speciation, both PSO and CPSO use a global version of PSO constriction for velocity update. In CPSO, the number of swarm populations is equal to three numbers of rules, on the other side, in SPSO, each particle represents a whole fuzzy system. Some of the advanced PSO algorithms are also compared with the HMDDE. These algorithms include HPSO–TVAC, HGAPSO and PSO–CREV. For these algorithms, the coefficients used to update the particle velocities are the same [23]. The performance of HMDDE is also compared with the advanced genetic algorithm. The MOGUL–TSK [37] is an iterative rule learning approach. The software knowledge extraction based on evolutionary learning (KEEL) [38] is used to implement the MOGUL–TSK. For the MOGUL–TSK parameter setting, the population size is the same as the HCMSPSO and is equal to 50.

The HMDDE is also compared with DE–NN and ODE–NN (Example 2 and 4) in the system identification. For DE and ODE parameter setting, the population size is 50, the upper and lower bounds of weights are [0, 1], the mutation constant M is 0.6, the crossover constant C is 0.5 and the jumping probability Jr is 0.3.

Example 1. Mackey-Glass time series prediction

In this sub-section, we describe some applications of the above beta basis function neural network architecture to infer the models associated with some chaotic time series. We consider the Mackey-Glass [34], one of the most popular and dynamical systems defined as

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

The resulting series presents a chaotic behavior and is recognized as a reference problem in the study of neural networks generalization ability. Unfortunately, the exact configuration of the experiment varies from one work to another. Here, the initial values of times series are fixed as follows $a=0.2$, $b=0.1$ and $\tau=17$.

In this paper, the neural network BBFNN is set to predict $x(t+6)$ based on $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$. A total of 1000 patterns are generated from the Eq. (10), where the first 500 patterns are selected as the training data points to build the proposed HMDDE–BBFNN of Mackey Glass time series, and the last 500 samples as the testing data of the proposed model.

In the proposed HMDDE, the parameters (Table 1) are set as, the population size to 50, the number of evolving generation to 10,000, the beta parameters are defined as the center c_i, the spread σ_i are limited to the upper and lower bound of the input, the form parameters in the interval [0, 5] and the maximum number of nodes is defined as 20.

In order to overcome the random initialization problem of the parameter, a total of 30 runs were indecently performed with the randomly generated initials parameters. Among the 30 runs, the number of nodes and the root mean square error (RMSE) of training and testing data obtained using the proposed algorithm are listed in Table 2. After the training, the performance of HMDDE is compared with the reported performance of different models that were applied to the same prediction problem [23]. Table 2 shows the comparison of the proposed method to the PSO-based algorithms and the GA-based algorithms. As shown in Table 2, we obtain over 30 runs, the average value for the number of nodes as four and the average RMSE testing data is 0.017. As observed, the HMDDE achieves the lowest testing and training

Table 1
Parameters of HMDDE.

Total number of iterations	10,000
Population size, *NP*	50
Mutation constant ,*M*	0.7
Crossover constant, *Cr*	0.6
Wight factor, *F*	0.7
Maximum number of nodes	20
Form parameters, *P*, *Q*	[0, 5]

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

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

Methods	CPSO	SPSO	HPSO–TVAC	HGAPSO	PSO–CREV	MOGUL–TSK	MOGUL–TSK	HCMSPSO	**HMDDE–BBFNN**
Neurons number	4 ± 0	4 ± 0	4 ± 0	4 ± 0	4 ± 0	4.2 ± 1.4	11.8 ± 1.5	4 ± 1.5	**4 ± 0**
Training RMSE	0.0199	0.0428	0.0345	0.0354	0.0664	0.0444	0.0213	0.0095	**0.0094**
Testing RMSE	0.0322	0.0450	0.0477	0.0478	0.0777	0.0450	0.0253	0.0208	**0.0170**

Results are taken from [23].

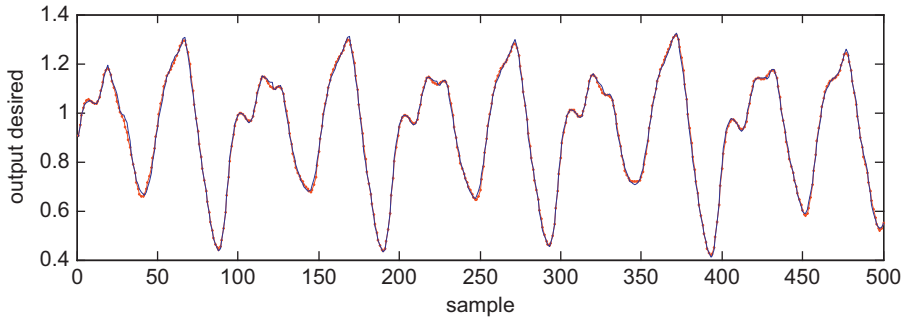

Fig. 5. Identification of the Mackey-Glass time series.

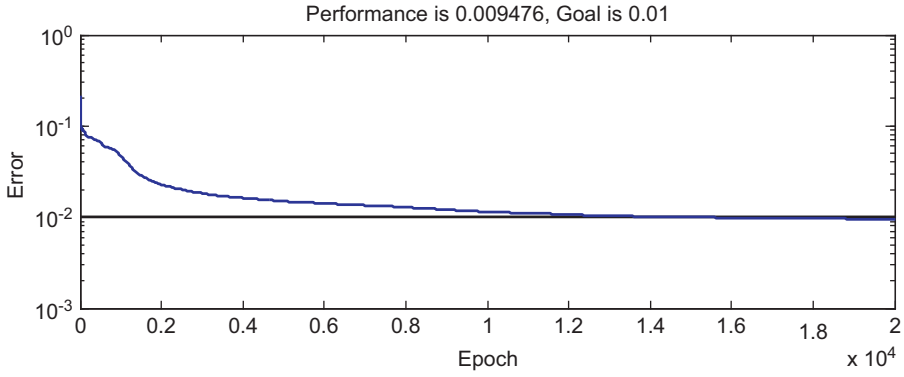

Fig. 6. The training error.

error. The average test error of HCMSPSO with about 4 nodes is smaller than those of the PSO based algorithm, it is still larger than that of the HMDDE with four nodes. Fig. 5 shows the best predicted and desired values for both the training and testing data. Fig. 6 shows the training RMSE for each evaluation.

Example 2. Nonlinear plant control

In this example, the plant to be controlled is expressed by

$$y_p(t+1)=\frac{y_p(t)[y_p(t-1)+2][y_p(t)+2.5]}{8.5+[y_p(t)]^2+[y_p(t-1)]^2}+u(t) \tag{11}$$

The same plant is used in [22]. The current output of the plant depends on two previous outputs values and one previous input values. The input $u(k)$ was assumed to be random signal uniformly in the interval $[-2, 2]$. The identification model to be in the form of:

$$y_{pi}(t+1)=f(y_p(t),y_p(t-1))+u(t) \tag{12}$$

where $f(y_p(t),y_p(t-1))$ is the nonlinear function of $y_p(t)$ and $y_p(t-1)$ which will be the inputs for HMDDE–BBFNN neural system identifier. The output from neural network will be y_{pi}. In this experiment, 500 training patterns are generated to train the BBFNN network and 500 for the testing data. After training, the following same test signal $u(k)$ of the other compared models is used for testing the performance of BBFNN models:

$$u(t)=\begin{cases}2\cos(2\pi t/100) & \text{if } t\le 200\\ 1.2\sin(2\pi t/20) & \text{if } 200<t\le 500\end{cases} \tag{13}$$

The RMSE value is taken as the performance measure criterion. The parameters of HMDDE were chosen as the previous example. Fig. 7 shows the actual and predicted output of the plant for the test signal with the beta BBFNN model. Fig. 8 gives the identification error of the 500 data points. From the figures it is clear that the desired output and the identified by HMDDE is nearly the same.

Table 3 gives the comparison of performance of HMMDE for the design of beta basis function neural network to DE based methods for Artificial Neural Network. The comparison of these methods is applied in terms of root mean squared error (RMSE). From the results it is clear that the proposed HMDDE algorithm has a root mean squared error test (RMSE) of 0.110 with four beta basis function neural net. Finally it is concluded that the proposed HMDDE is having a better identification performance than that of the other approaches.

Example 3. Nonlinear—plant identification

The HMDDE–BBFNN is applied to the model of the following nonlinear plant:

$$y=(3e^{x2}-1)\tanh(x_1)+\frac{2}{30}(4+e^{x2})\sin(\pi(x_1+\frac{4}{10})) \tag{14}$$

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

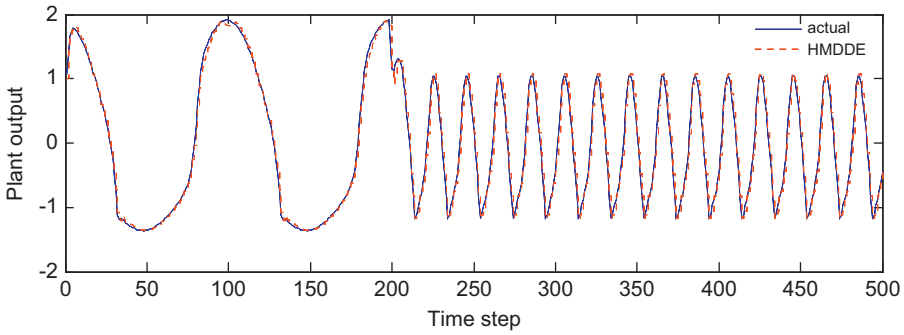

Fig. 7. HMDDE identification performance.

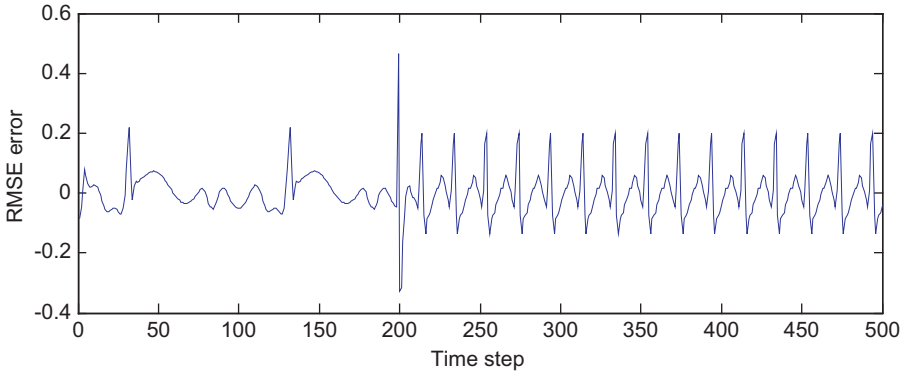

Fig. 8. HMDDE identification error.

Table 3
Comparison of training and testing errors.

Input	Training error (RMSE)			Testing error (RMSE)		
	DE	ODE	**HMDDE**	DE	ODE	**HMDDE**
$y(k)$,$y(k-1)$ $u(k)$	0.0207	0.0190	**0.0190**	0.1186	0.11370	**0.110**

Results are taken from [21] and [22].

Table 4
Comparison of different models for system identification example.

Methods	HPSO–TVAC	HGAPSO	PSO–CREV	MOGUL–TSK	MOGUL–TSK	HCMSPSO	**HMDDE–BBFNN**
Neuron number	3.6 ± 0.9	3.6 ± 0.9	3.6 ± 0.9	3.7 ± 1.13	$27.9 \pm 2.$	13.6 ± 0.9	**3.44**
Training RMSE	0.0145	0.0249	0.0539	0.679	0.0054	0.0052	**0.0048**
Testing RMSE	0.0283	0.0312	0.0761	0.734	0.058	0.0064	**0.0050**

Results are taken from [23].

where the domains of x_1 and x_2 are both in $[-1, 1]$. The input of BBFNN is x_1 and x_2. A threefold cross-validation procedure is performed. For each cross-validation dataset, the learning process is repeated for ten runs, i.e., there are a total of 30 runs for statistical analysis.

Table 4 shows the comparison performance of the proposed model to genetic based-algorithm andother advanced PSO based-algorithms in terms of number of neurons number, training RMSE and testing RMSE. The results in Table 4 show that the average test error and the average of number of neurons of HMDDE–BBFNN. The performance of HMDDE is better than that of HCMSPSO and those PSO-based algorithms and this advanced is in error fitness and also in number of neurons. Results show that applying the HMMDE for the design of the beta basis function neural network improves the generalization error. Fig. 9 shows the identification performance of the plant whereas the Fig. 10 shows the error identification.

Example 4. Box and Jenkins' gas furnace problem

In this section, the proposed HMDDE–BBFNN are applied to the Box–Jenkins time series data (gas furnace data), which have been intensively studied as a benchmark problem in previous literature [35,36]. The data set originally consists of 296 data points $[y(t), u(t)]$. For the design of the experiment, the delayed term of the observed gas furnace process data, $y(t)$, is used as

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

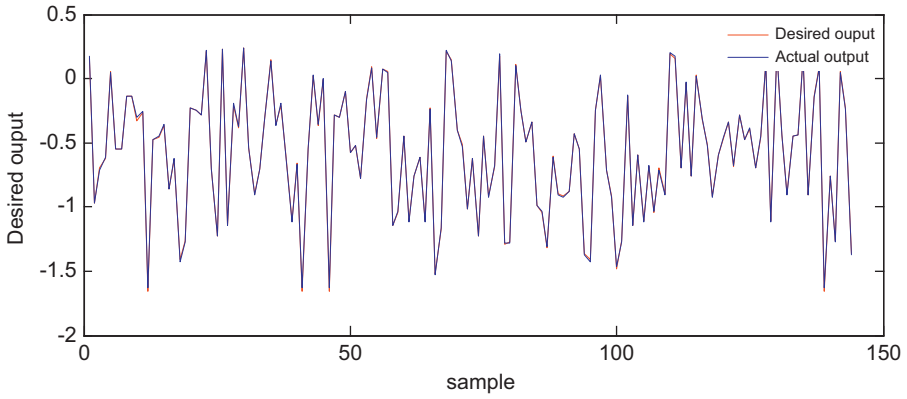

Fig. 9. Identification performance.

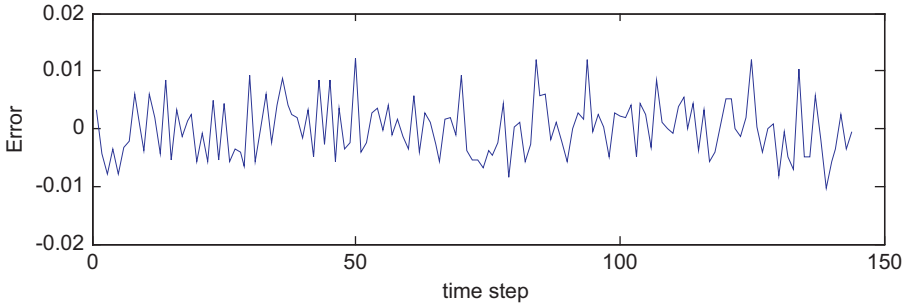

Fig. 10. Identification error.

Table 5
Comparison of training and testing errors of Box and Jenkins.

Input	Testing error (RMSE)			Training error (RMSE)		
	DE	ODE	**HMDDE**	DE	ODE	HMDDE
$y(t-1)$, $u(t-3)$	0.4400	0.4194	**0.2276**	0.1501	0.1411	**0.1328**
$y(t-3)$, $u(t-4)$	0.7838	0.7773	**0.4224**	0.3402	0.2850	**0.0210**
$y(t-2)$, $u(t-4)$	0.6733	0.6602	**0.3200**	0.3256	0.2898	**0.1365**
$y(t-1)$, $u(t-2)$	0.4906	0.6801	**0.2334**	0.2909	0.2924	**0.1735**
$y(t-1)$, $u(t-4)$	0.5430	0.5132	**0.3745**	0.2991	0.2926	**0.2411**
$y(t-4)$, $u(t-4)$	12.259	0.8894	**0.4549**	0.3274	0.3428	**0.1594**
$y(t-2)$, $u(t-3)$	1.1340	0.7199	**0.2700**	0.2968	0.3051	**0.1702**
$y(t-1)$, $u(t-1)$	0.6183	0.6056	**0.2577**	0.4638	0.4151	**0.1598**
$y(t-4)$, $u(t-3)$	1.2405	1.2771	**0.6148**	0.7266	0.4301	**0.1921**
$y(t-1)$, $u(t-6)$	0.8469	0.8410	**0.6638**	0.6012	0.5661	**0.6619**
$y(t-3)$, $u(t-3)$	1.0067	1.0347	**0.2521**	0.5172	0.5176	**0.1600**
$y(t-2)$, $u(t-2)$	0.9889	0.9753	**0.2773**	0.6314	0.6261	**0.1615**
$y(t-1)$, u(t−5)	0.6873	0.6518	**0.5595**	0.6220	0.6303	**0.3333**
$y(t-4)$, $u(t-5)$	1.0149	0.9698	**0.0203**	0.7038	0.6373	**0.0178**
$y(t-2)$, u(t−1)	1.8368	1.2726	**0.2759**	0.8934	0.6844	**0.1960**
$y(t-2)$, $u(t-5)$	0.9176	1.1808	**0.4021**	0.7222	0.6804	**0.2165**
$y(t-3)$, $u(t-5)$	0.9536	1.0470	**0.2307**	0.7138	0.7338	**0.1346**
$y(t-3)$, $u(t-2)$	1.8184	1.4138	**0.2760**	0.8766	0.8600	**0.2128**
$y(t-4)$, $u(t-6)$	1.7628	1.4677	**0.2635**	1.3988	1.1126	**0.1379**
$y(t-2)$, $u(t-6)$	1.3352	1.2639	**0.5590**	1.6264	1.1945	**0.3389**
$y(t-4)$, $u(t-2)$	1.6725	1.6377	**0.2737**	1.1799	1.1963	**0.2152**
$y(t-3)$, $u(t-6)$	27.468	1.4641	**0.4027**	1.2063	1.2424	**0.2175**
$y(t-3)$, $u(t-1)$	1.7123	1.6475	**0.2803**	1.5725	1.2702	**0.2135**
$y(t-4)$, $u(t-1)$	2.0821	2.0217	**0.2695**	1.4250	1.4352	**0.2270**

Results are taken from [21] and [22].

system input variables made up of by ten terms given as follows: $y(t-1)$, $y(t-2)$, $y(t-3)$, $y(t-4)$, $y(t-5)$, $u(t-1)$, $u(t-2)$, $u(t-3)$, $u(t-4)$, and $u(t-6)$. Consequently, the effective number of data points is reduced to 296 providing 100 for training and 190 samples for testing. Here we have taken two inputs for simplicity one is from furnace output and other is from furnace input so we have build 24 models of different input and output. The criterion used was the Root Mean Square Error (RMSE). Each case is trained for 2000 epochs and the number of neurons belongs to the interval [2,10]. Table 5 gives the training and testing performances of these 24 models. As can be seen from this table, HMDDE–BBFNN model is powerful for Box–Jenkins process in training. Compared with the recent results presented in [21,22], we can see that the proposed algorithm can achieve accuracy than the result in [21,22] with a smaller number of nodes.

The training and testing gas furnace process data are shown in Fig. 11. Fig. 12 shows the predicted time series and the desired time series. From the Table 5, it is clear that HMDDE is having less training and testing errors in comparison to DE, ODE counterpart. The RMSE for testing turned out to be the least for 24 cases in HMDDE–BBFNN approach.

5. Conclusions

This paper proposed the hierarchical multi-dimensional differential evolution (HMDDE) algorithm for optimization of beta basis function neural network (BBFNN). The design of the topology of BBFNN is defined automatically at the higher level of the proposed algorithm, whereas in lower level, we optimize the free parameters of beta neural network. The optimization of neural parameters is based on differential evolution. The second contribution of this work is to propose the multi-dimensional differential evolution, which represents the key point in determining the optimal number of beta basis function neural network. Several simulation studies are carried out for both identification and control purposes. The plant models are taken from the literature to enable a direct performance comparison. In both the identification and the control cases, the performance is much better, resulting in smaller RMSE values, with a smaller number of nodes. Future works on the topic includes the use of Particle

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

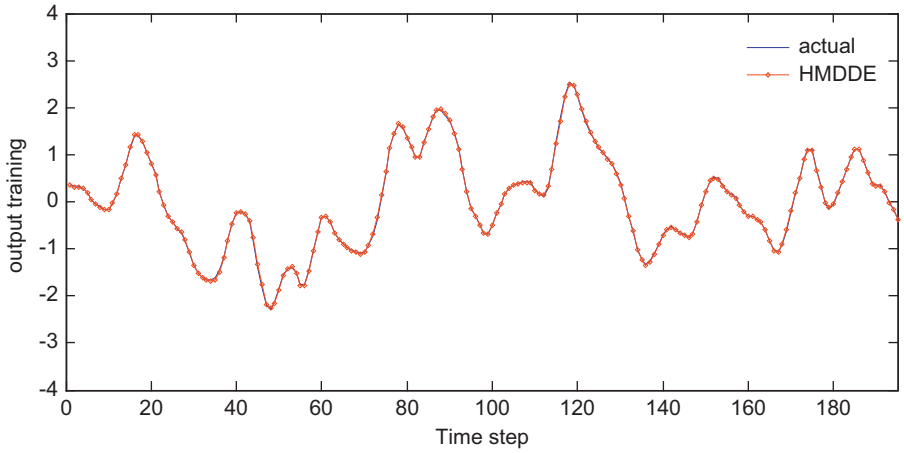

Fig. 11. Training of Box–Jenkins time series ($(y(t-4), u(t-5))$).

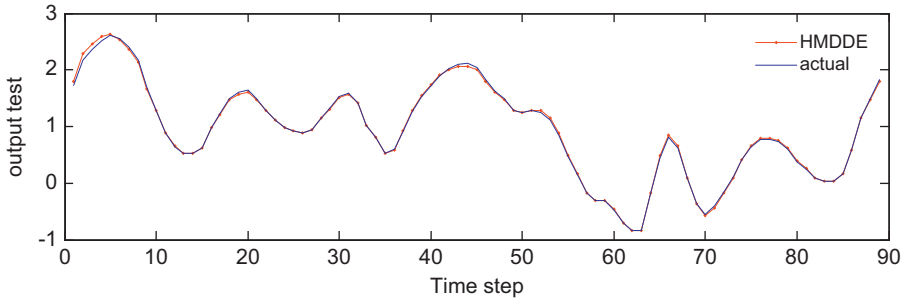

Fig. 12. Identification performance ($y(t-4), u(t-5)$).

Swarm Optimization (PSO) for the optimization of BBFNN parameters.

Acknowledgment

The authors would like to acknowledge the financial support of this work by grants from the General Direction of Scientific Research (DGRST), Tunisia, under the ARUB program.

References

[1] A.M. Alimi, The beta system: toward a change in our use of neuro-fuzzy systems, Int. J. Manage. (2000) 15–19.
[2] C. Harpham, W. Dawson, R. Brown, A review of genetic algorithms applied to training radial basis function networks, Neural Comput. Appl. 13 (3) (2004) 193–201.
[3] S.J. Ovaska, A. Kamiya, Y.Q. Chen, Fusion of soft computing and hard computing: computational structures and characteristic features, IEEE Trans. Syst. Man Cybern. 36 (3) (2006) 439–448.
[4] P.M. Pawar, R. Ganguli, Matrix crack detection in thin-walled composite beam using genetic fuzzy system, J. Intell. Mater. Syst. Struct. 16 (5) (2005) 395–409.
[5] X. Yao, Y. Liu, Towards designing artificial neural networks by evolution, Appl. Math. Comput. 91 (1) (1998) 83–90.
[6] X. Yao, Y. Liu, A new evolutionary system for evolving artificial neural networks, IEEE Trans. Neural Networks 8 (3) (1997) 694–713.
[7] X. Yao, Designing artificial neural networks using co-evolution, in: Proceedings of the IEEE Singapore International Conference on Intelligent Control and Instrumentation, 1995, pp. 149–154.
[8] F. Mascioli, G. Martinelli, A constructive algorithm for binary neural networks: the oil spot algorithm, IEEE Trans. Neural Networks 6 (3) (1995) 794–797.
[9] L.M. Fernanda, T.B. Ludermir, Clustering and co-evolution to construct neural network ensembles: an experimental study, Neural Networks 21 (2008) 1363–1379.
[10] C.L. Giles, C.W. Omlin, Pruning recurrent neural networks for improved generalization performance, Trans. Neural Networks 5 (5) (1994) 848–851.
[11] S.W. Stepniewski, A.J. Keane, Pruning back-propagation neural networks using modern stochastic optimization techniques, Neural Comput. Appl. 5 (1997) 76–98.
[12] P.T. Baffles, J.M. Zelle, Growing layers of perceptrons: introducing the exentron algorithm, in: Proceedings of the International Joint Conference on Neural Networks, vol. 2, 1992, pp. 392–397.
[13] J. Herrero, A. Valencia, J. Dopazo, A hierarchical unsupervised growing neural network for clustering gene expression patterns, Bioinformatics 17 (2) (2001) 26–36.
[14] H. Guang-Bin, P. Saratchandran, N. Sundararajan, A generalized growing and pruning RBF (GGAP–RBF) neural network for function approximation, IEEE Trans. Neural Networks 16 (1) (2005) 57–67.
[15] X. Yao, Y. Liu, Making use of population information in evolutionary artificial neural networks, IEEE Trans. Syst., Man Cybern. B: Cybern. 28 (3) (1998) 417–425.
[16] R. Sexton, R. Dorsey, J.D. Johnson, Optimization of neural networks: a comparative analysis of the genetic algorithm and simulated annealing, Eur. J. Oper. Res. 114 (1999) 589–601.
[17] R. Storn, Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous space, J. Global Optim. 4 (1995) 341–359.
[18] H. Dhahri, A.M. Alimi, F. Karray, The modified particle swarm optimization for the design of the beta basis function neural networks, in: Proceedings of the Congress on Evolutionary Computation, Hong Kong, China, 2008, pp. 3874–3880.
[19] H. Dhahri, A.M. Alimi, F. Karray, Opposition-based particle swarm optimization for the design of beta basis function neural network, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 2010, pp. 18–23.
[20] H. Dhahri, A.M. Alimi, F. Karray, Opposition-based differential evolution for beta basis function neural network, in: Proceedings of the IEEE Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 1–8.
[21] B. Subudhi, D. Jena, A differential evolution based neural network approach to nonlinear system identification, Appl. Soft Comput. 11 (1) (2011) 861–871.
[22] B. Subudhi, D. Jena, Nonlinear system identification using opposition based learning differential evolution and neural network techniques, Appl. Soft Comput. 11 (1) (2011) 861–871.
[23] C.F. Juang, C.M. Hsiao, C.H. Hsu, Hierarchical cluster-based multispecies particle-swarm optimization for fuzzy-system optimization, IEEE Trans. Fuzzy Syst. 18 (1) (2010) 14–26.
[24] L.D. Santos, D. Coelho, et al., Improved differential evolution algorithms for handling economic dispatch optimization with generator constraints, Energy Convers. Manage. 48 (2007) 1631–1639.

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008

[25] N. Noman, H. Iba, Accelerating differential evolution using an adaptive local search, IEEE Trans. Evol. Comput. 12 (1) (2008) 107–125.
[26] X. Xu, Y. Li, Comparison between particle swarm optimization, differential evolution and multi-parents crossover, in: Proceedings of the International Conference on Computational Intelligence and Security, 2007, pp. 124–127.
[27] H.A. Bourlard, N. Morgan, Connectionist Speech Recognition: A Hybrid Approach, Kluwer Academic Publishers, Boston, 1994.
[28] A. Cichocki, R. Unbehauen, Neural Networks for Optimization and Signal Processing, John Wiley & Sons, NY, 1993.
[29] J. Vesterstrøm, R. Thomson, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, in: Proceedings of the Sixth Congress on Evolutionary Computation, San Diego, USA, 2004, pp. 1980–1987.
[30] R. Hassine, F. Karray, A.M. Alimi, et al., Approximation properties of fuzzy systems for smooth functions and their first-order derivative, IEEE Trans. Syst., Man, Cybern A: Syst. Humans 33 (2) (2003) 160–168.
[31] H. Simon, Neural Networks: A Comprehensive Foundation, Printice Hall, New Jersey, 1994.
[32] H. Dhahri, A.M. Alimi, Hierarchical learning algorithm for the beta basis function neural network, in: Proceedings of the Third International Conference on Systems, Signals & Devices, Tunisia, 2005.
[33] M. Njah, A.M. Alimi, M. Chtourou, R. Tourki, Algorithm of maximal descent AMD for training radial basis function neural networks, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics SMC'02, Hammamet, Tunisia, October 2002.
[34] M.C. Mackey, L. Glass, Oscillation and chaos in physiological control systems, Science 197 (1977) (1977) 287–289.
[35] G.E.P. Box, G.M. Jenkins, Time Series Analysis, Forecasting and Control, Holden day, CA, 1970.
[36] Y. Chen, B. Yang, J. Dong, Time-series prediction using a local linear wavelet neural network, Neurocomputing 69 (2006) 449–465.
[37] R. Alcalá, J. Alcalá-Fdez, J. Casillas, O. Cordón, F. Herrera, Local identification of prototypes for genetic learning of accurate TSK fuzzyrule-based systems, Int. J. Intell. Syst. 22 (2006) 909–941.
[38] J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, F. Herrera, KEEL: a software tool to assess evolutionary algorithms to data mining problems, Soft Comput. 13 (3) (2009) 307–318.
[39] A. Abraham, Optimization of evolutionary neural networks using hybrid learning algorithms, in: Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN02), 2002 IEEE World Congress on Computational Intelligence, Hawaii, IEEE Press, vol. 3, 2002, pp. 2797–2802, ISBN:0780372786.
[40] A. Abraham, Meta-Learning Evolutionary Artificial Neural Networks Neurocomputing Journal, 56c, Elsevier Science, Netherlands, 2004, pp. 1–38.
[41] Y. Chen, B. Yang, J. Dong, A. Abraham, Time Series Forecasting Using Flexible Neural Tree Model, Information Sciences, 174, Elsevier Science, 2005, pp. 219–235.
[42] Y. Chen, A. Abraham, Tree-Structure based Hybrid Computational Intelligence: Theoretical Foundations and Applications Intelligent Systems Reference Library Series, Springer Verlag, Germany, 2009.

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

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

Please cite this article as: H. Dhahri, et al., Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing (2012), http://dx.doi.org/10.1016/j.neucom.2012.04.008