Paper:

Genetic and Bacterial Programming for B-Spline Neural Networks Design

János Botzheim[*], Cristiano Cabrita[**], László T. Kóczy[*,*], and Antonio E. Ruano[**]**

[*]Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics
Magyar Tudósok Krt. 2, Budapest, H-1117, Hungary
E-mail: {botzheim, koczy}@tmit.bme.hu
[**]Centre for Intelligent Systems, University of Algarve
Campus de Gambelas, 8000 Faro, Portugal
E-mail: {ccabrita, aruano}@ualg.pt
[***]Institute of Information Technology and Electrical Engineering, Széchenyi István University, Győr, Hungary
[Received June 21, 2006; accepted October 17, 2006]

The design phase of B-spline neural networks is a highly computationally complex task. Existent heuristics have been found to be highly dependent on the initial conditions employed. Increasing interest in biologically inspired learning algorithms for control techniques such as Artificial Neural Networks and Fuzzy Systems is in progress. In this paper, the Bacterial Programming approach is presented, which is based on the replication of the microbial evolution phenomenon. This technique produces an efficient topology search, obtaining additionally more consistent solutions.

Keywords: constructive algorithms, B-splines, bacterial programming, genetic programming

1. Introduction

B-spline neural networks offer definite advantages over more commonly used neural networks, such as multilayer perceptrons or radial basis function networks. B-spline networks store the information locally, which means that learning in one part of the input space affects the rest only minimally. For this reason, they are suitable for on-line adaptive modeling and control applications. Their grid-based structure makes them transparent, thus, in contrast to other networks, it is possible to understand the knowledge stored in these networks. B-spline networks have better generalization capability in high dimensional problems than some radial basis function models [1]. They can be used easily in control applications and nonlinear modeling (see e.g. [19]) because their locally structured scheme provides fast convergence and suitable complexity properties.

The design process for B-spline networks involves two different phases: the determination of the best topology, which can be seen as a system identification problem, and the determination of its parameters, which can be envisaged as a parameter estimation problem. This latter issue, the determination of the net parameters (linear weights and interior knots) is the simplest task and is usually solved using gradient or hybrid schemes [3, 17]. The former issue, the topology determination, is an extremely complex task, especially if dealing with real-world problems. The current paper deals with this topology determination task. The aim here is to devise a "tractable" model, in terms of computational efficiency, which delivers a "good" compromise, in terms of accuracy. A key issue of the identification process is to find a suitable balance between computational complexity and accuracy. This issue is discussed in Kóczy and Zorat [10], where the 'Fuzzy Cat and Mouse' problem is analyzed, the task of optimizing the fuzzy rule base of a cat hunting a random moving mouse. It is shown that in many special cases a well-defined optimum exists.

Nature inspired some evolutionary optimization algorithms suitable for global optimization of even non-linear, high-dimensional, multimodal, and discontinuous problems. The original Genetic Algorithm (GA) was developed by Holland [9] and was based on the process of evolution of biological organisms. Recognized as a powerful global search technique, genetic algorithms have been applied to wide variety of problems. They exhibit a remarkable balance between search domain exploration and exploitation [8]. On one hand, every part of the domain is searched enough and on the other hand, the search effort is concentrated around the best solutions and their neighborhood, producing even better solutions. Recently, approaches like Genetic Programming (GP) [11] and Bacterial Evolutionary Algorithm (BEA) [14] present an alternative to the former algorithms. GP optimization uses the same operators as GA, though it requires an expression tree for gene representation as a combination of functions [5]. On the other hand, operations of the bacterial evolutionary algorithm were inspired by the microbial evolution phenomenon. In this paper the Bacterial Programming is introduced, which is a fusion of the principles of BEA and GP.

Evolutionary algorithms are useful tools for identification problems, they can help with finding a good com-

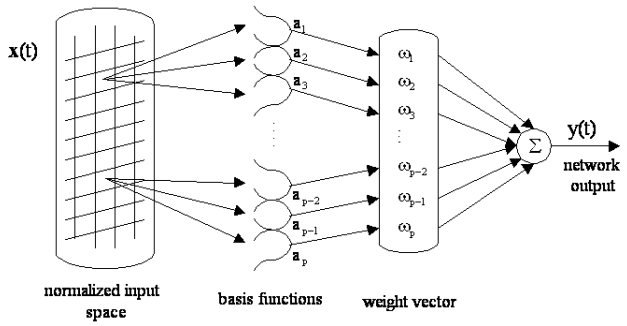


Fig. 1. Structure of a lattice-based network.

promise between the accuracy and the complexity of the model. The bacterial approach was successfully applied for fuzzy model identification by various authors [2, 14]. The goal of the paper is to show that this technique is applicable for B-spline neural networks' design, too. At a higher level, the principles of B-spline neural networks and rule based fuzzy systems are the same. This parallelism between these two different model types allows learning algorithms derived for B-spline networks to be employed for fuzzy models and vice versa under certain conditions [17].

The structure of the paper is as follows. Section 2 describes the topology of B-spline neural networks. Section 3 makes an overview of Genetic Programming applied to B-spline neural networks. Section 4 introduces our proposed new technique, the Bacterial Programming. Simulation results are given in Section 5 and conclusions are drawn in Section 6.

2. B-Spline Neural Networks

B-spline neural networks belong to the class of networks termed *grid or lattice-based associative memories networks* (AMN). This type of networks is composed of three layers: a *normalized input space* layer, a *basis functions* layer and a *linear weight* layer (**Fig. 1**).

2.1. Normalized Input Layer

The normalized input layer can take different forms but is usually a grid on which the basis functions are defined. In order to define a grid in the input space, vectors of *knots* must be defined, one for each input axis. There are usually different numbers of knots for each dimension, and they are generally placed at different positions.

The *interior knots* for the i^{th} axis are $\lambda_{i,j}, j = 1,\ldots,r_i$. They are arranged in such a way that:

$$x_i^{\min} < \lambda_{i,1} \le \lambda_{i,2} \le \cdots \le \lambda_{i,r_i} < x_i^{\max} \quad . \; . \; . \; . \quad (1)$$

where $x_i^{\min}$ and $x_i^{\max}$ are the minimum and maximum values of the i^{th} input, respectively.

At the extremes of each axis, a set of k_i *exterior knots* must be given which satisfy:

$$\lambda_{i,-(k_i-1)} \le \cdots \le \lambda_{i,0} = x_i^{\min} \quad . \; . \; . \; . \; . \; . \; . \quad (2)$$

$$x_i^{\max} = \lambda_{i,r_i+1} \le \cdots \le \lambda_{i,r_i+k_i}. \quad . \; . \; . \; . \; . \; . \; . \quad (3)$$

These exterior knots are needed to generate the basis functions that are close to the boundaries. These knots usually coincide with the extreme of the input axes, or are equidistant. The network input space is $\left[x_1^{\min}, x_1^{\max}\right] \times \cdots \times \left[x_n^{\min}, x_n^{\max}\right]$, and so the exterior knots are only used for defining these basis functions at the extreme of the grid.

The j^{th} interval of the i^{th} input is denoted as $I_{i,j}$ and is defined as:

$$I_{i,j} = \begin{cases} [\lambda_{i,j-1}\ \lambda_{i,j}) & \text{for } j = 1,\ldots,r_i \\ [\lambda_{i,j-1}\ \lambda_{i,j}] & \text{if } j = r_i+1. \end{cases} \quad . \; . \; . \quad (4)$$

This way, within the range of the i^{th} input, there are r_i+1 intervals (possibly empty, if the knots coincide), which means that there are $p' = \prod_{i=1}^{n}(r_i+1)$ n-dimensional cells in the grid.

2.2. The Basis Functions Layer

The output of the hidden layer is determined by a set of p B-spline basis functions defined on the n-dimensional grid. B-spline functions possess easy local adjusting, calculation and implementation.

The shape, size and distribution of the basis functions are characteristics of the particular AMN employed, and the support of each basis function is bounded. In B-spline neural networks, the *order* of the spline implicitly sets the size of the basis function's support and its shape. The support of univariate B-spline basis function of order k equals k intervals. Hence, each input is assigned to k basis functions.

The j^{th} univariate basis function of order k is denoted $N_k^j(x)$, and it is defined by the following relationships:

$$N_k^j(x) = \left(\frac{x-\lambda_{j-k}}{\lambda_{j-1}-\lambda_{j-k}}\right) N_{k-1}^{j-1}(x) + \left(\frac{\lambda_j - x}{\lambda_j - \lambda_{j-k+1}}\right) N_{k-1}^{j}(x) \quad . \; . \; . \quad (5)$$

$$N_1^j(x) = \begin{cases} 1 & \text{if } x \in I_j \\ 0 & \text{otherwise.} \end{cases} \quad . \; . \; . \; . \; . \; . \; . \quad (6)$$

Multivariate basis functions are formed by taking the *tensor product* of the univariate basis functions (see **Fig. 2**). Therefore, each multivariable basis function is formed from the product of n univariate basis functions, one from each input axis, and every possible combination of univariate basis function is taken:

$$N_{\mathbf{k}}^{j}(\mathbf{x}) = \prod_{i=1}^{n} N_{\mathbf{k}_i,i}^{j}(\mathbf{x}_i). \quad . \; . \; . \; . \; . \; . \; . \; . \; . \; . \quad (7)$$

The number of basis functions of order k_i defined on an axis with r_i interior knots is $r_i + k_i$. Therefore, the total number of basis functions for a multivariate B-spline is $p = \prod_{i=1}^{n}(r_i+k_i)$.

This number depends exponentially on the size of the

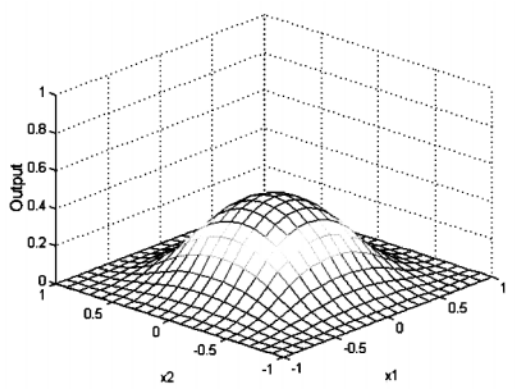


Fig. 2. Bivariate B-spline function.

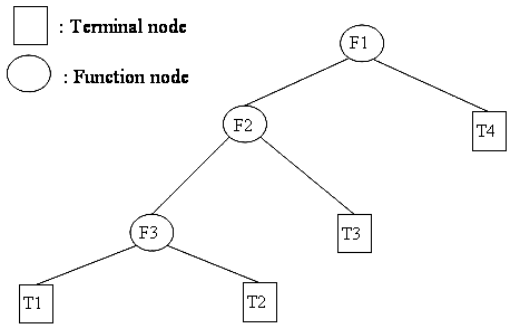


Fig. 3. General layout for an expression tree.

input, hence the B-splines are only applicable for problems where the input dimension is small (typically ≤ 5).

2.3. The Weight Layer

The output of an AMN is a linear combination of the outputs of the basis functions. The linear coefficients are the adjustable weights, and as the mapping is linear, finding the weights is just a linear optimization problem. The output is therefore:

$$y = \sum_{i=1}^{p} \mathbf{a}_i \mathbf{w}_i = \mathbf{a}^T \mathbf{w} \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad (8)$$

where $\mathbf{a}_i = N_k^i(\mathbf{x})$, $i = 1, \ldots, p$. As only $p'' = \prod_{i=1}^{n} k_i$ are active at any one time, the calculation of Eq. (8) can be reduced to:

$$y = \sum_{i=1}^{p''} \mathbf{a}_{act(i)}(\mathbf{x}) \mathbf{w}_{act(i)} \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad (9)$$

where $\mathbf{a}_{act(i)}(\mathbf{x})$ denotes the i^{th} active basis function for input $\mathbf{x}$.

2.4. Sub-Modules

To overcome the "curse of dimensionality", it is common to employ, instead of a single module covering all inputs, a linear sum of smaller sub-models, each one with a lower input dimensionality. The output of such a network is:

$$y(\mathbf{x}) = \sum_{u=1}^{n_u} S_u(\mathbf{x}_{\underline{u}}) \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad (10)$$

where $S_i(\mathbf{x}_{\underline{i}})$ denotes the i^{th} sub-model, and $\mathbf{x}_{\underline{i}}$ is the set of input variables ($\underline{i}$) which compose sub-model i.

2.5. B-Spline Neural Networks Design

The design of a B-spline network involves the following design phases:

1. The determination of the number of its sub-models;
2. For each sub-model, the set of its inputs;
3. The order of the splines for each input;
4. The number of the interior knots for each input;
5. The location of the interior knots for each input;
6. The values of the linear output weights.

The determination of the two last points can be envisaged as a non-linear least-squares problem, and therefore complete supervised training algorithms can be employed for their determination (for more details see [17]). The former points constitute a very complex combinatorial problem. There are different constructive algorithms to help in this task, such as the *ASMOD* (*Adaptive Spline Modelling of Observed Data*) algorithm [18], the *MARS (Multivariate Adaptive Regression Splines*) algorithm [7], the *LOLIMOT* [16] algorithm and the genetic programming technique [4]. This paper proposes a new method for B-spline neural networks design, namely the *Bacterial Programming* approach, and compares this algorithm with the genetic programming technique.

3. Genetic Programming

There exist various optimization algorithms, which were inspired by processes in the nature. The advantage of these algorithms is their ability to solve and quasi-optimize problems with non-linear, high-dimensional, multimodal, and discontinuous character. These processes can easily be applied in optimization problems where one individual corresponds to one possible solution of the problem. In the original genetic algorithm an individual is represented by a sequence of numbers, for example a sequence of bits. This sequence is called *chromosome*. The GA uses three operators: selection, crossover, and mutation. A more recent approach is genetic programming, which uses the same operators as GA, though the individuals are represented by the so called expression tree (see **Fig. 3**). This tree is composed of the function nodes (inner ones) and the terminal nodes (outer ones), representing the functions and their input arguments, respectively.

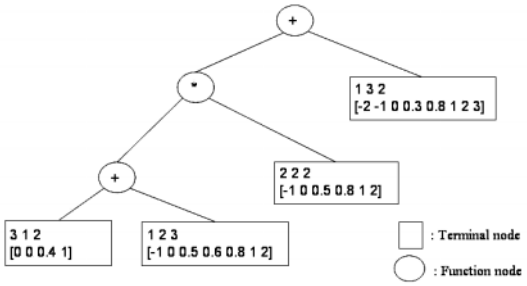


Fig. 4. A sample expression tree for B-spline networks.

3.1. Genetic Programming for B-Splines Design

As mentioned in the previous sections, in B-splines design, sub-models must be *added* (+), sub-models of higher dimensionality must be *created* from smaller sub-models (*), and sub-models of higher dimensionality must be *split* into lower dimensional sub-models (/). This is the set of primitive functions that are defined for B-splines design.

In contrast with the application of GP to other neural networks, the node terminals do not represent only one *input variable*, but also the *spline order*, the *number of interior knots*, and *their locations*. **Fig. 4** shows the composition of the nodes in such a tree. For the tree given in **Fig. 4**, for instance the left-most leaf of the tree means that this terminal node contains the input variable 3, its spline order is 1, the number of its interior knots is 2, and they are located in 0 and 0.4 positions. One would expect the model's output to be given as a function of the addition of 3 sub-models, two of each would be bi-dimensional, so that:

$$y(\mathbf{X}) = f(\mathbf{X}_3 \times \mathbf{X}_2) + f(\mathbf{X}_1 \times \mathbf{X}_2) + f(\mathbf{X}_1),$$

where $\boldsymbol{X}_i$ denotes the input variable i.

Obviously these functions and terminals must be well defined, so that they may receive any value as argument returned by another set of functions or terminals in the lower sub-trees.

Whenever any model presents a complexity higher than the number of samples within the training set, it is desirable to perform a convenient change in its structure so that this candidate may still participate in the evolution and be not completely discarded. In order to obtain a valid candidate, during evaluation, the expression tree is traversed and, at every node, the complexity value of the tree beneath it is evaluated. If this value is greater than the number of input patterns, the complexity will be reduced in the following way:

1. By replacing the tensor product function by the addition function if the inner node is a tensor product function.
2. By replacing the addition function by the lower subtree that corresponds to the least complex ramification.

Steps 1 and 2 are performed until the candidate is designated valid. This means that some functions and terminals in the tree may be discarded.

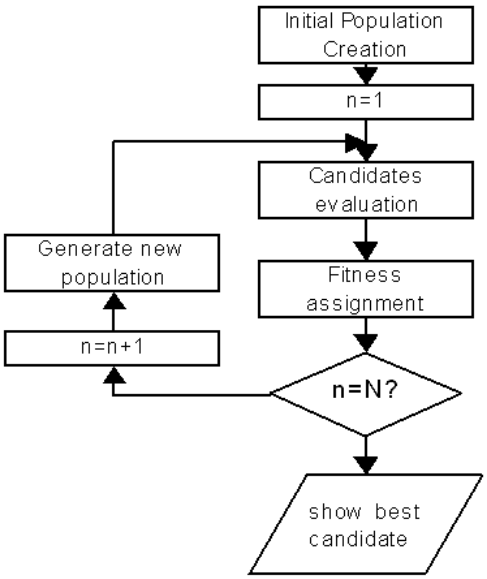


Fig. 5. Flowchart of Genetic Programming.

The evolutionary process involves the following steps:

- The creation of an initial population, and the determination of the size of the population;
- The evaluation of the candidates using **B**ayes **I**nformation **C**riterion [16], and their fitness assignment;
- The application of some of genetic operators, such as:

  Selection: Pairs of parent trees are selected based on their fitness for reproduction.

  Crossover: A node in the tree is selected at random and exchanging the associated sub-trees produces a pair of offspring trees.

  Mutation: This is performed by either replacing a node selected at random by a sub-tree generated randomly or by changing its type.

  Replacement: All parents are replaced by the offspring (generational approach).

- The termination criterion, which is usually the maximum number of generations defined.

The cycle of evolution is summarized in **Fig. 5**. Mutation on a function implies the replacement of the tree node with a randomly generated sub-tree of maximum length 2. Mutation on a terminal can be of 6 different types, which are:

1. Full replacement of the terminal.
2. Variable identification replacement.
3. Splines order replacement.
4. Random displacement of an interior knot.

5. Addition of N interior knots placed randomly. N is fixed to 5.
6. Removal of N interior knots. In the absence of interior knots, no actual operation is executed.

For simplification purposes, the terminal mutation rates will be described as a vector $p_{\text{mut-terminal}}$ =[%1 %2 %3 %4 %5 %6], where %i designates the i^{th} type mutation rate.

4. Bacterial Programming

A slightly different evolutionary technique is called bacterial evolutionary algorithm. This algorithm was introduced by Japanese researchers in the late 90's [14]. The first version of this algorithm was the Pseudo-Bacterial Genetic Algorithm (PBGA) [15] which proposed a modified mutation operator called bacterial mutation, based on the natural phenomenon of microbial evolution. The Bacterial Evolutionary Algorithm (BEA) introduced a new operator called gene transfer operator. While PBGA incorporates bacterial mutation and crossover operator, the BEA substitutes the classical crossover with the gene transfer operation. Both of these new operators were inspired by bacterial evolution. Bacteria can transfer genes to other bacteria. The bacterial mutation performs local optimization whilst the gene transfer allows the bacteria to directly transfer information to the other individuals in the population.

Based on these bacterial operations but using the tree structures similar to the ones in the genetic programming, we propose a new technique (cf. also [6]). We named this technique Bacterial Programming (BP).

4.1. The Evolutionary Process

The evolutionary process of BP involves the following steps:

1. The creation of an initial population, and the determination of the size of the population.
2. Application of bacterial mutation for each bacterium.
3. Application of gene transfer operation to the current population.
4. If the terminal criterion is achieved, the algorithm stops, otherwise it continues from step 2.
5. The terminal criterion is usually the maximum number of generations.

The cycle of evolution is summarized in **Fig. 6**.

4.2. The Encoding Method

Bacterial programming employs the same operators that a bacterial algorithm uses in its search procedure. However, from our point of view this approach is much useful for this type of neural networks because, instead of coding the network parameters in bit strings, it requires a tree structure, composed of *function* and *terminal* nodes. One bacterium is represented by one such expression tree. This tree structure, as well as the characteristics of the nodes, evolves from generation to generation.

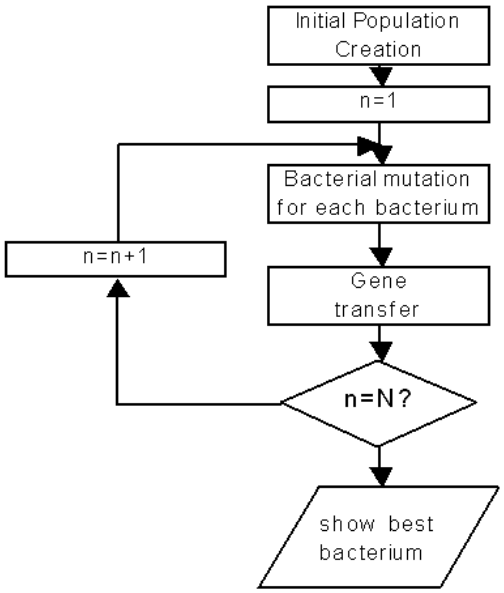


Fig. 6. Flowchart of Bacterial Programming.

4.3. Bacterial Mutation

The bacterial mutation is applied to each bacterium one by one. First, N_{clones} copies (clones) of the bacterium are generated. Then, a certain part of the bacterium is randomly selected and the parameters of this selected part are randomly changed in each clone (mutation). In the new method, because coding is given by an expression tree, there are two types of parts: function and terminal parts. Next, all the clones and the original bacterium are evaluated by an error criterion. The best individual transfers the mutated part into the other individuals. This cycle is repeated for the remaining parts until all of the parts of the bacterium have been mutated and tested. At the end, the best bacterium is kept and the remaining N_{clones} are discharged. By the help of this operation, the bacterium will be at least as good as before, but in most of the cases it will be better. Bacterial mutation is more efficient than classical mutation in GA because of the nature of cloning. Every clone brings a new chance to find a better solution anywhere in the search space, thus wider space can be explored.

Figures 7 and **8** illustrate the mutation procedure. From **Fig. 7**, it can be seen that after a function mutation at a node, the whole sub-tree beneath this node is replaced by a new randomly generated one, using the "grow" method [12]. However, terminal mutation affects only the given node.

In bacterial programming, mutation is applied once to the selected part, meaning that neither the selected nodes nor the sub-tree in case of function mutation, will be chosen once again for mutation, in the same generation. The terminal mutations in the clones can be any of the same six different types as in genetic programming.

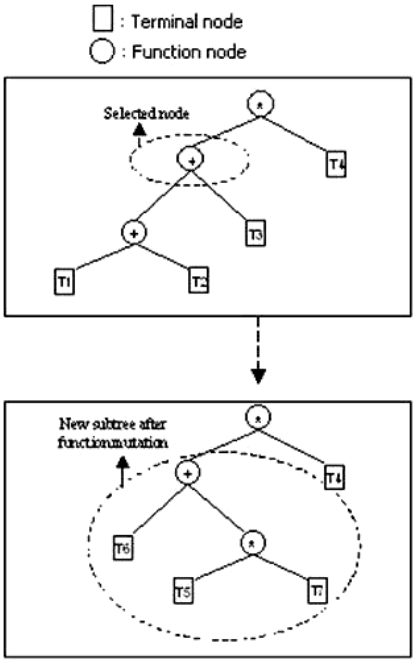


Fig. 7. Mutation on a function part: the individual's selected node sub-tree is changed randomly.

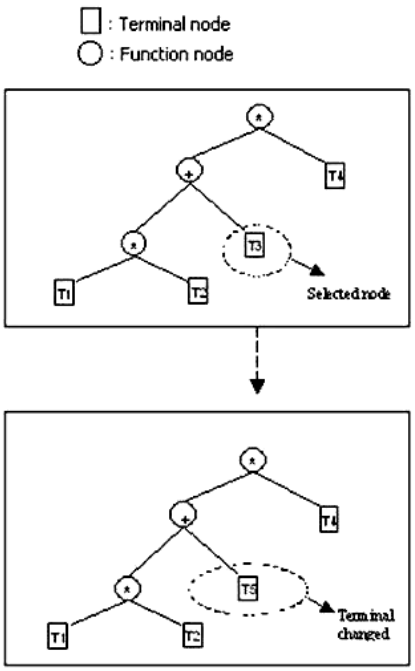


Fig. 8. Mutation on a terminal part: only the selected node changed randomly given the terminal mutation rates.

4.4. Gene Transfer

The aim of the gene transfer operation is to exchange genetic information between two bacteria. This procedure is somewhat similar to the crossover operation used in genetic programming.

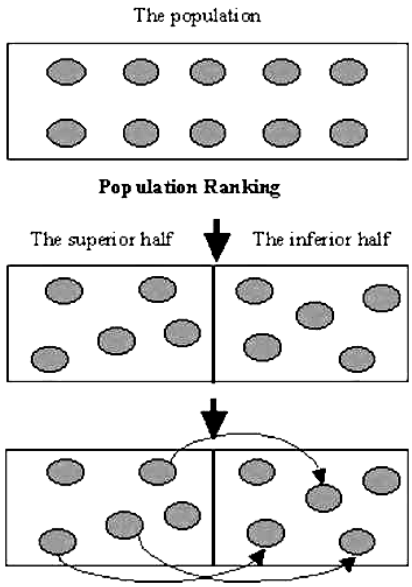


Fig. 9. The gene transfer procedure.

First, the population must be divided into two halves, where the "better" bacteria are called the superior half, whereas the other is referred to as inferior half. Neither fitness assignment nor any selection method is performed to candidates, as is often the case in GP. Next, one bacterium called the "source bacterium" is randomly chosen from the superior half, while the other is randomly selected from the inferior half. This will be termed the "destination bacterium". A part from the source bacterium is chosen and this part will overwrite a part of the destination bacterium.

Gene transfer is repeated for N_{inf} times, where N_{inf} is the number of "infections", per generation. Since the destination bacterium represents the worst part of the population, by accepting phenotypes from the better part, this operation leads to better solutions. Gene transfer shows some advantages over the crossover in GA. A superior bacterium gives information to an inferior bacterium, while the inferior one does not give back any information to the superior one. Thus, here we will definitely not lose the superior individual, which can happen in the GA, because of the exchange of information between two parents. The gene transfer operation is illustrated in **Fig. 9**.

4.5. Bacterium Evaluation

The bacteria can be evaluated with different criteria, such as *RMS* (Root-Mean-Square) in the training set, or *Cross-Validation*; the most usual criteria are however, *information criteria*, which balance the accuracy obtained against the model complexity. Some of the most employed are:

- Final Prediction Error;
- Akaike Information Criterion;

- Minimum Description Length;
- Structural Risk Minimization;
- Bayesian Information Criterion.

This last criterion is used in this work and is described as:

$$BIC = m\ln(RMS) + n\ln(m) \quad \ldots\ldots\ldots (11)$$

where m denotes the number of training samples and n the model complexity (number of basis functions).

4.6. Parameters

An evolutionary computational algorithm needs to be fine-tuned. There are different techniques for the steps described earlier (and also different parameters) that should be selected for the particular application. A preliminary experimental study was conducted and the main conclusions are summarized here:

- As it is often the case, the larger is the size of the population, and the number of generations employed, the better are the results obtained.
- The parameter of bacterial mutation is the number of clones (N_{clones}). The larger is the N_{clones}, the more effective is the bacterial mutation.
- From all the terminal mutations referred above, the one related with knots addition appears to be the most important.

5. Simulation Results

In this section three problems are used in order to illustrate the power of the new method. In [4] the genetic programming is compared with some other methods applied for B-spline design task, such as the ASMOD [18], the MARS [7], and the LOLIMOT [16] algorithms. This paper focuses only for the comparison of the genetic programming with the newly proposed bacterial programming technique.

5.1. The pH Problem

The aim of this example is to approximate the inverse of a titration-like curve. This type of non-linearity relates the *pH* (a measure of the activity of the hydrogen ions in a solution) with the concentration (x) of chemical substances.

5.2. The Inverse Coordinate Transformation Problem (ICT)

This example illustrates an inverse kinematic transformation between 2 Cartesian coordinates and one of the angles of a two-links manipulator.

For a more complete description of the examples refer to [17].

5.3. A Six Dimensional Generic Function

This example is widely used as a target function and the output is given by the following expression (see e.g. [2]):

$$y = x_1 + x_2^{0.5} + x_3 x_4 + 2e^{2(x_5 - x_6)},$$

where

$x_1 \in [1,5]$, $x_2 \in [1,5]$, $x_3 \in [0,4]$,
$x_4 \in [0,0.6]$, $x_5 \in [0,1]$, $x_6 \in [0,1.2]$.

5.4. BP Parameter Values Selection

To start, results were generated in order to decide the best values for the BP parameters. The parameters for the BP are the number of individuals (N_{ind}), the number of clones (N_{clones}), the number of infections (N_{inf}), and the number of generations (N_{gen}). The training patterns used were taken from the pH problem known from the literature (cf. [17]).

Therefore, 10 sessions were executed and results for the mean values for the BIC, MSE (***M**ean **S**quare of the absolute **E**rror*), MSRE (***M**ean **S**quare of the **R**elative **E**rror*), and PMRE (***P**ercentage of **M**ean **R**elative **E**rror*) values were obtained.

The MSRE and PMRE were defined as follows:

$$MSRE = \frac{1}{N\_pat}\sum_{i=1}^{N\_pat}\frac{(t_i - y_i)^2}{y_i^2};$$

$$PMRE = \frac{100}{N\_pat}\sum_{i=1}^{N\_pat}\left|\frac{t_i - y_i}{y_i}\right|. \quad \ldots\ldots (12)$$

where t_i refers to the desired output and y_i is the model output for the i^{th} pattern, respectively, and N_pat is the number of patterns. These relative errors can be useful when the output has a wide range because they consider the error of each pattern relatively to the model's output. Nevertheless, the BIC and MSE values are more important in the simulation results because the individuals are evaluated with the BIC criterion, and BIC includes somehow the MSE instead of the relative errors.

Both the number of individuals, and generations are 20. Firstly, the number of clones was adjustable to 5, 10 and 15 clones, using a number of infections of 5.

Secondly, the number of infections was adjustable, set to 5, 10 and 15 infections, using a number of 8 clones.

The results presented in **Table 1** show that, in general, the more the number of clones the better is the accuracy of the output, as expected. However, the more clones we apply the more computation we need. So, we need to find some optimal balance and avoid using too many clones.

When one analyses the values in **Table 2**, it can be assumed that the results are not as different from each other as in the case of the number of clones' issue. More infections may lead the population into local optima because of the premature convergence. A low value for N_{inf} may provide better result in general, and it needs less computation.

Table 1. Mean values for BIC, MSE, MSRE, PMRE and model complexity adjusting parameter N_{clones}.

N_{clones}	5	10	15
BIC	-1695.9	-1834.3	-1913
MSE	3.6×10^{-8}	6.9×10^{-9}	2.3×10^{-9}
MSRE	5.8×10^{-2}	2.9×10^{-3}	1.2×10^{-3}
PMRE	1.2	2.7×10^{-1}	1.7×10^{-1}
Complexity	57.8	73	81

Table 2. Mean values for BIC, MSE, MSRE, PMRE and model complexity adjusting parameter N_{inf}.

N_{inf}	5	10	15
BIC	-1823.9	-1792.8	-1934.3
MSE	8.4×10^{-9}	4.2×10^{-8}	2.2×10^{-9}
MSRE	7.1×10^{-3}	7.4×10^{-2}	3.4×10^{-3}
PMRE	4.1×10^{-1}	9.1×10^{-1}	3.3×10^{-1}
Complexity	75.9	76.2	87.4

Table 3. Parameters definition for both algorithms.

Parameters	GP	BP
N_{inf}	–	5
N_{clones}	–	8
N_{ind}	160	20
N_{gen}	20	20
Crossover rate	50% of population	–
Mutation rate	0.8	–

5.5. Comparison Between BP and GP

In this section, the aim is to show results obtained from the GP and BP when applied to academic problems used as benchmark.

As in the former section, 10 sessions were executed and the mean values for the BIC, MSE, MSRE and PMRE were obtained. The number of patterns used is 101, 110, and 200 for the pH, ICT and the six dimensional generic function problem, respectively.

In order to obtain the same computational complexity by the two algorithms, the values for the parameters used are as shown in **Table 3**.

The terminal type mutation rate is: [5%, 10%, 5%, 10%, 60%, 10%], for both algorithms.

From the values used for the parameters, it can be seen that the size of the population in the BP is much smaller. However, setting a number of 8 clones gives similar computational complexity because in the bacterial mutation we use 8 clones for each bacterium. One advantage of the BP approach is that we do not need to handle big population; the evolution of only 20 bacteria is enough to compare the methods.

Table 4. Mean values for MSE, MSRE, PMRE and model complexity obtained for the pH problem.

	GP	BP
BIC	-1784.6	-1786.7
MSE	1.1×10^{-8}	1.3×10^{-8}
MSRE	1.3×10^{-2}	2.6×10^{-2}
PMRE	6.1×10^{-1}	6.9×10^{-1}
Complexity	82.6	72.4

Table 5. Model structure for the lowest BIC value found after all sessions for the pH problem.

	GP	BP
Sub-models	(1)	(1)
Complexity	33	40
BIC	-1903.1	-1874.3
MSE	1.4×10^{-9}	1.8×10^{-9}
MSRE	2.7×10^{-3}	7.5×10^{-5}
PMRE	5.3×10^{-1}	1.0×10^{-1}
$\lvert W\rvert$	3.3	3.6

Table 6. Mean values for MSE, MSRE, PMRE and model complexity obtained for the ICT problem.

	GP	BP
BIC	-1344.4	-1539.7
MSE	2.5×10^{-7}	8.6×10^{-8}
MSRE	1.6×10^{5}	2.1×10^{3}
PMRE	1331.6	125.8
Complexity	33.4	31.7

5.5.1. The pH problem

In **Table 4** the mean values obtained for the pH problem can be seen. In this case, the results of GP and BP seem to be similar. However, the complexity is lower in the BP case. In **Table 5** the model structure for the best individual is shown in the case of BP and GP also. From this result it can be diagnosed that both algorithms show similar final values. The reason for this is that the pH problem is only a one dimensional problem, thus the structure of B-spline neural network is not so complex, and so both methods can solve the design task easily.

5.5.2. The Inverse Coordinate Transformation Problem

Results for the ICT problem can be seen in **Tables 6** and **7**. BP gives better results not only in the mean case but also if the best individual is considered (the individual with the lowest BIC). Though GP shows lower values for the best individuals regarding the relative errors, however, the evolution processes are driven by the BIC criterion (which contains the MSE in some way) both for GP and BP, thus the BIC and MSE criteria are more important. According to these criteria our method gives better results.

Table 7. Model structure for the lowest BIC value found after all sessions for the ICT problem.

	GP	BP
Sub-models	(1x2) (1) (2)	(2x1) (1)
Complexity	106	106
BIC	-1533.9	-2048.3
MSE	1.3×10^{-8}	8.8×10^{-11}
MSRE	1.6×10^{-7}	22.45
PMRE	1.3×10^{-2}	79.7
$\|W\|$	686.7	2.6×10^{7}

Table 8. Mean values for MSE, MSRE, PMRE and model complexity obtained for the six dimensional generic function problem.

	GP	BP
BIC	-380.1	-552.9
MSE	2.0×10^{-1}	2.7×10^{-2}
MSRE	2.1×10^{-3}	5.2×10^{-4}
PMRE	2.1	0.98
Complexity	38.8	44.6

Table 9. Model structure for the lowest BIC value found after all sessions for the six dimensional generic function problem.

	GP	BP
Sub-models	(5) (4) (2) (3x4) (5x3x6) (3x1)	(6x5) (5x2) (6x1) (3x6) (3x4) (1)
Complexity	98	156
BIC	-593.2	-702
MSE	3.8×10^{-3}	4.8×10^{-4}
MSRE	7.3×10^{-5}	1.2×10^{-5}
PMRE	6.6×10^{-1}	2.4×10^{-1}
$\|W\|$	423.8	128.4

5.5.3. A Six Dimensional Generic Function

The main advantage of the bacterial approach can be deduced from the results of the six dimensional problem. The results show that the bacterial method gives good performance for problems with larger input dimension. From **Tables 8** and **9** one can see that the BP is better comparing the mean values and the best individuals.

In **Figs. 10** and **11** the desired output (target) and the error can be seen for each pattern considering the best individual for GP and BP, respectively. Comparing these pictures we can see that our technique gives the smaller error.

5.6. Statistical Approach

In the previous simulations we could see that the Bacterial Programming is more efficient than the Genetic Programming. The reason for that is the different nature of the operations in the BP approach. Bacteria can explore bigger part of the search space because of the effective clones in the bacterial mutation.

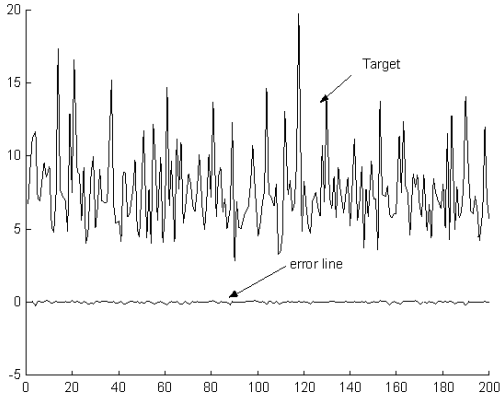


Fig. 10. Target output and error for the six dimensional generic function problem, using GP.

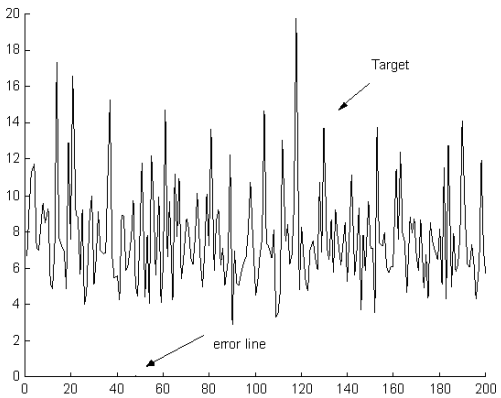


Fig. 11. Target output and error for the six dimensional generic function problem, using BP.

In this subsection some hypotheses tests are presented as shown in the table below. A significance level for rejection of $\alpha = 5\%$ was used, giving a 95% rate of confidence on the null hypotheses.

To assess the equality of solutions, the most-popular two sample method was applied, the Mann-Whitney test [13]. This test supplies the p-value which represents the probability of obtaining equal value samples drawn from two different algorithms. Also, to ascertain whether both algorithms have equal medians, the location method known as the Median test was conducted. Given the number of runs and the cumulative sum of ranks from samples from one of the algorithms, the Median test poses a judgment on the probability of having different, smaller or bigger medians between two populations.

The results shown in **Table 10** indicate how similar the performance is for both algorithms when using the one dimensional pH problem, since the p-value is high. When

Table 10. Statistical inference obtained for the BP and GP using both the Mann-Whitney and the Median test methods.

Problem	Mann-Whitney test (p-value)	Median test
pH	0.7624	Different medians
ICT	0.0156	Different medians
Six dimensional generic function	0.00194	Lower Median for BP than for GP

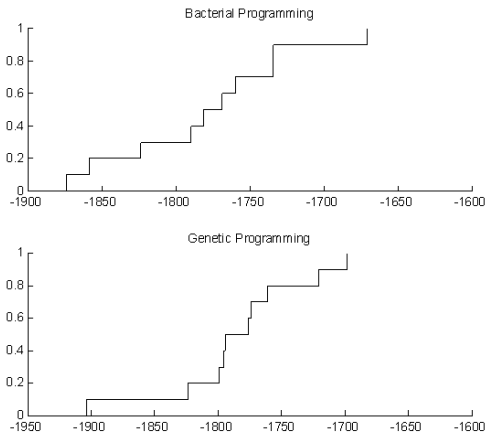


Fig. 12. Empirical probability distribution function for the pH problem.

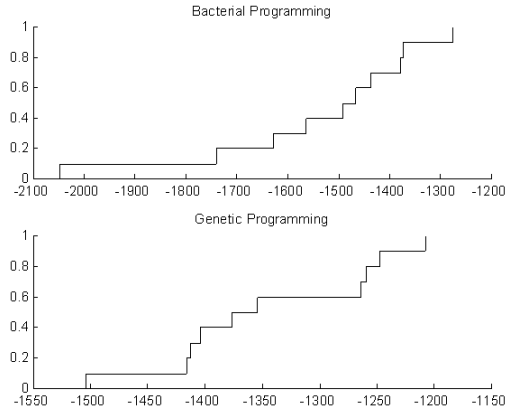


Fig. 13. Empirical probability distribution function for the ICT problem.

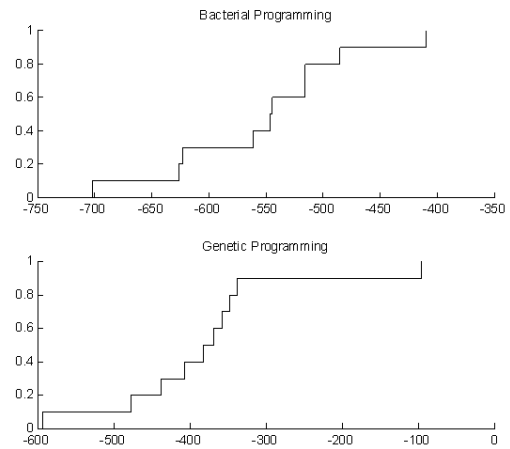


Fig. 14. Empirical probability distribution function for the generic six dimensional problem.

using multi-variable problems it looks as if the probability of obtaining similar results was depreciated, since the p-value is lower than the 5% rejection boundary. Moreover, it happens that the six dimensional problem presents not only a different but also lower median value, which means that in most of the times a better evaluation criterion will be obtained.

Figures 12-14 relate to the empirical probability distribution functions for each of the problems, using both algorithms. From the figures the same conclusions can be done as in the previous subsection. For the one dimensional problem the two approaches give similar results. For the two dimensional ICT problem, the best individuals' BIC values are between approx. -2050 and -1750 with BP, and between approx. -1500 and -1420 with GP, which means that the GP method provides much poorer result than our technique. In **Fig. 14** the six dimensional problem is shown. While the best individuals have only BIC values with GP between about -600 and -500, they have the same values between -700 and -630 with the BP.

6. Conclusions

In B-spline neural networks design one important task is to find the best topology. Different algorithms were introduced previously, which try to solve this task. In this paper, the bacterial programming is introduced, which applies the bacterial operators instead of the original genetic operators. The bacterial mutation optimizes the local portions of the individual. The gene transfer operator substitutes the traditional crossover operator and allows the transfer of information between different individuals. So, while the bacterial mutation is working on one individual, and tries to optimize this bacterium, the gene transfer is applied to the whole bacterium population, avoiding the local minima solutions. If more clones are being applied in the bacterial mutation, then better results are received. In the gene transfer operation, it is fundamental not to use a too high infection value in order to avoid that the population can be trapped in a local minimum.

The operations of the bacterial programming seem to be more effective than the operations of the classical ge-

netic programming. In the bacterial mutation we have more chance to find a better solution because of the large number of clones. In the gene transfer we do not lose good individuals, because the information flow is directed from the superior sub-population to the inferior one. Another advantage of the bacterial approach is that no selection process is needed. Fewer individuals in the population are sufficient than in case of the genetic algorithms. If the dimensionality of the problem is increasing then we need to increase the number of individuals and/or the number of generations in order to reach as good results as for the lower dimensional problem. If we consider a problem with a given number of input dimensions, then the computational demand of the method grows if the values of the parameters are larger, however, the accuracy of the model is better in this case. If we intend to reach higher accuracy then the values of the numerical parameters have to be increased, thus the computational demand is higher in this case.

The advantage of our technique is that it is efficient in higher dimensional problems, too. Bacterial Programming turns out to be an effective tool for optimization.

Acknowledgements

This paper was supported by the Széchenyi University Main Research Direction Grant 2005, a National Scientific Research Fund Grant OTKA T048832, and the Hungarian-Portuguese Bilateral Intergovernmental S&T Cooperation Grant P-8/2005.

References:

[1] C. Bishop, "Improving the generalization properties of radial basis function neural networks," Neural Computation, Vol.3, pp. 579-588, 1991.

[2] J. Botzheim, B. Hámori, L. T. Kóczy, and A. E. Ruano, "Bacterial algorithm applied for fuzzy rule extraction," in Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems, IPMU 2002, pp. 1021-1026, Annecy, France, 2002.

[3] M. Brown and C. Harris, "Neurofuzzy Adaptive Modelling and Control," Prentice-Hall, 1994.

[4] C. Cabrita, A. E. Ruano, and C. M. Fonseca, "Single and multi-objective genetic programming design for B-spline neural networks and neuro-fuzzy systems," IFAC Workshop on Advanced Fuzzy/Neural Control (AFNC'01), pp. 93-98, Valencia, Spain, 2001.

[5] C. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Kóczy, "Genetic programming and bacterial algorithm for neural networks and fuzzy systems design," IFAC International Conference on Intelligent Control Systems and Signal Processing (ICONS 2003), pp. 500-505, Faro, Portugal, 2003.

[6] C. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Kóczy, "Design of B-spline Neural Networks using a Bacterial Programming Approach," International Joint Conference on Neural Networks, pp. 2313-2318, Budapest, Hungary, 2004.

[7] J. H. Friedman, "Multivariate Adaptive Regression Splines," The Annals of Statistics, Vol.19, No.1, pp. 1-141, 1991.

[8] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley, Reading, Massachusetts, 1989.

[9] J. H. Holland, "Adaptation in natural and artificial systems," University of Michigan Press, Ann Arbor, USA, 1975.

[10] L. T. Kóczy and A. Zorat, "Fuzzy systems and approximation," Fuzzy Sets and Systems, Vol.85, pp. 203-222, 1995.

[11] J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection," MIT Press, 1992.

[12] J. R. Koza, "Genetic Programming II, Automatic Discovery of Reusable Programs," 2nd ed., MIT, 1998.

[13] H. B. Mann and D. R.Whitney, "On a test of whether one of two random variables is stochastically larger than the other," Annals of Mathematical Statistics, Vol.18, pp. 50-60, 1947.

[14] N. E. Nawa and T. Furuhashi, "Fuzzy System Parameters Discovery by Bacterial Evolutionary Algorithm," IEEE Tr. Fuzzy Systems 7, pp. 608-616, 1999.

[15] N. E. Nawa, T. Hashiyama, T. Furuhashi, and Y. Uchikawa, "Fuzzy logic controllers generated by pseudo-bacterial genetic algorithm," in Proc. IEEE 1997 Int. Conf. Neural Networks (ICNN'97), Houston, pp. 2408-2413, 1997.

[16] O. Nelles, "Nonlinear Systems Identification with Local Linear Neuro-Fuzzy Models," Ph.D. Thesis, TU Darmstadt, Germany, 2000.

[17] A. E. Ruano, C. Cabrita, J. V. Oliveira, and L. T. Kóczy, "Supervised training algorithms for B-spline neural networks and neuro-fuzzy systems," International Journal of Systems Science, Vol.33, No.8, pp. 689-711, 2002.

[18] E. Weyer and T. Kavli, "The ASMOD Algorithm. Some New Theoretical and Experimental Results," SINTEF Report STF31 A95024, Oslo, 1995.

[19] K. F. C. Yiu, S. Wang, K. L. Teo, and A. C. Tsoi, "Nonlinear System Modeling via Knot-Optimizing B-Spline Networks," IEEE Trans. Neural Networks, Vol.12, pp. 1013-1022, 2001.

Name:
János Botzheim

Affiliation:
Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics

Address:
Magyar Tudósok Krt. 2, Budapest, H-1117, Hungary

Brief Biographical History:

2001- M.Sc. in Technical Informatics from the Budapest University of Technology and Economics

Ph.D. student in the Department of Telecommunications and Media Informatics at the Budapest University of Technology and Economics

Main Works:

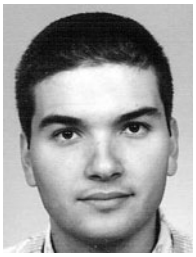
• J. Botzheim, B. Hámori, and L. T. Kóczy, "Extracting trapezoidal membership functions of a fuzzy rule system by bacterial algorithm," In B. Reusch (Ed.), Computational Intelligence, Theory and Applications, Lecture Notes in Computer Science, Vol.2206, pp. 218-227, Springer-Verlag, Berlin-Heidelberg, 2001.

• J. Botzheim, C. Cabrita, L. T. Kóczy, and A. E. Ruano, "Estimating fuzzy membership functions parameters by the Levenberg-Marquardt algorithm," IEEE Int. Conf. on Fuzzy Systems, pp. 1667-1672, Budapest, Hungary, 2004.

Membership in Academic Societies:

• Hungarian Fuzzy Association

• John von Neumann Computer Society



Name:
Cristiano Cabrita

Affiliation:
Centre for Intelligent Systems, University of Algarve

Address:
Campus de Gambelas, 8000 Faro, Portugal

Brief Biographical History:
1998- First Degree in Systems and Computing Engineering from the University of Algarve, Portugal
2001- M.Sc. in Systems and Computing Engineering from the University of Algarve, Portugal
1999- Assistant Professor at the Escola Superior de Tecnologia, University of Algarve
Ph.D. student in Systems and Computing Engineering at the University of Algarve

Main Works:
• C. Cabrita, A. E. Ruano, and C. M. Fonseca, "Single and multi-objective genetic programming design for B-spline neural networks and neuro-fuzzy systems," IFAC Workshop on Advanced Fuzzy/Neural Control (AFNC'01), pp. 93-98, Valencia, Spain, 2001.

Name:
László T. Kóczy

Affiliation:
Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics

Address:
Magyar Tudósok Krt. 2, Budapest, H-1117, Hungary

Brief Biographical History:
1975, 1976, 1977 M.Sc., M. Phil., Ph.D. in Electrical Engineering from Technical University of Budapest
1998- Doctor of the Hungarian Academy of Science (the highest earnable postdoctoral degree in Hungary)
Professor (Telecommunication and Information Technology) at Budapest University of Technology and Economics, and Professor and Dean at the Széchenyi István University, Győr

Main Works:
• Y. Yam and L. T. Kóczy, "Representing membership functions as points in high-dimensional spaces for fuzzy interpolation and extrapolation," IEEE Tr. on Fuzzy Systems, Vol.8, No.6, pp. 761-772, 2000.
• L. T. Kóczy, K. Hirota, and L. Muresan, "Interpolation in hierarchical fuzzy rule bases," Int. Journal of Fuzzy Systems, Vol.1, No.2, pp. 77-84, 1999.
• L. T. Kóczy, "Fuzzy if...then rule models and their transformation into one another," IEEE Tr. SMC A, Vol.26, No.5, pp. 621-637, 1996.
• J. M. Han, L. T. Kóczy, and T. Poston, "Fuzzy Hough transform," Pattern Rec. Letters, Vol.15, pp. 649-658, 1994.
• L. T. Kóczy and K. Hirota, "Approximate reasoning by linear rule interpolation and general approximation," International Journal of Approximate Reasoning, Vol.9, pp. 197-225, 1993.

Membership in Academic Societies:
• 2001-2003 President of the International Fuzzy Systems Association
• 1990- Hungarian Fuzzy Society Founding President, Life Honorary President
• 2004- IEEE Computational Intelligence Society AdCom Member
• 2005- President of CIS Hungarian Chapter

Name:
Antonio E. Ruano

Affiliation:
Centre for Intelligent Systems, University of Algarve

Address:
Campus de Gambelas, 8000 Faro, Portugal

Brief Biographical History:
1982- First Degree in Electronic and Telecommunications Engineering from the University of Aveiro, Portugal
1989- M.Sc. in Electrotechnic Engineering from the University of Coimbra, Portugal
1992- Ph.D. degree in Electronic Engineering from the University of Wales
1992- Joined the Department of Systems Engineering and Informatics of the Faculty of Sciences & Technology of the University of Algarve
1996- Assistant Professor of Automatic Control
Coordinator of the Centre of Intelligent Systems of the University of Algarve

Main Works:
• A. E. Ruano, C. Cabrita, J. V. Oliveira, and L. T. Kóczy, "Supervised training algorithms for B-spline neural networks and neuro-fuzzy systems," International Journal of Systems Science, Vol.33, No.8, pp. 689-711, 2002.
• C. Teixeira, A. E. Ruano, M. G. Ruano, W. C. Pereira, and C. A. Negreira, "Non-invasive Temperature Prediction of In-vitro Therapeutic Ultrasound Signals using Neural Networks," Medical & Biological Engineering & Computing (Springer), 44, 1-2, pp. 111-116, 2006.
• A. E. Ruano, E. M. Crispim, E. Z. E. Conceição, and M. M. J. R. Lúcio, "Prediction of Building's Temperature using Neural Network Models," Energy and Buildings (Elsevier Science), 38, pp. 682-694, 2006.
• A. E. Ruano, P. M. Ferreira, and C. M. Fonseca, "An Overview of Nonlinear Identification and Control with Neural Networks," in A. E. Ruano (Ed.), Intelligent Control Systems using Computational Intelligence Techniques, IEE Control Series.

Membership in Academic Societies:
• Member of the Editorial Board of International Journal of Systems Science
• Associate Editor for Automatica