Model Identification by Bacterial Optimization[1]

Botzheim, J. * – Kóczy, L.T. *,**

*Department of Telecommunication and Media Informatics, Budapest University of Technology and Economics, Budapest, Hungary

**Institute of Information Technology and Electrical Engineering, Széchenyi István University, Győr, Hungary

Abstract: In the field of control systems it is common to use techniques based on model adaptation to carry out control for plants for which mathematical analysis may be intricate. Increasing interest in biologically inspired learning algorithms for control techniques such as artificial neural networks and fuzzy systems is in progress. In this paper a recent kind of evolutionary method called bacterial algorithm is introduced. This method can be used for fuzzy rule extraction and optimization. Bacterial Programming is also proposed in this paper. This approach is the combination of the bacterial algorithm and the genetic programming techniques and can be applied for the optimization of the structure of B-spline neural networks.

1 Introduction

There are several optimization methods 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, multi-modal, and discontinuous character. It has been shown that evolutionary algorithms are efficient tools for solving nonlinear, multiobjective and constrained optimizations. These algorithms have the ability to explore large admissible spaces, without demanding the use of derivatives of the objective functions, such as the gradient-based training methods. Their principles are based on the search for a population of solutions, which tuning is done using mechanisms similar to biological recombination. The original Genetic Algorithm (GA), developed by Holland [5] which is based on the process of evolution of biological organisms. Recently, approaches like Genetic Programming (GP) [6] and Bacterial Algorithm (BA) [7] presented an alternative to the former algorithms. GP uses the same operators as GA, though it requires an expression tree for gene representation as a combination of functions. On the other

[1] Research supported by the National Scientific Research Fund OTKA T034233 and T034212, a Széchenyi University Research Grant 2004, and the National Research and Development Project Grant 2/0015/2002

hand, operations of the bacterial evolutionary algorithm were inspired by the microbial evolution phenomenon. Bacterial Programming (BP) [3] is also introduced, which is a fusion between the principles of the BA and GP. In the applications of fuzzy systems, one of the most important tasks is to find the optimal rule base. The rules can be defined by a human expert or can be given a priori by the linguistic description of the modelled system. If however neither is available, the rule base has to be generated by other methods based on numerical data. The design process of neural networks involves the topology determination which is an extremely complex task, especially if dealing with real-world problems. For both problems the bacterial optimization approach offers suitable solution.

The paper is organised as follows. Section 2 describes the structure of the fuzzy system applied. The bacterial algorithm for trapezoidal fuzzy systems is presented in Section 3. In Section 4 the B-spline neural networks are shown. Section 5 introduces the Bacterial Programming applied to B-spline neural networks.

2 Fuzzy Systems

The theory of fuzzy logic was developed by Zadeh in the early 1960s. His theory was essentially the rediscovering the multi-valued logic created by Lukasiewicz, however, with going much further in some application related aspects. In 1973 he pointed out that the new fuzzy concept could be excellently used for describing very complex problems with a system of fuzzy relations represented by a fuzzy rule base [8]. A fuzzy rule base contains fuzzy rules R_i:

$$R_i\text{: } IF\ (x_1\ is\ A_{i1})\ AND\ (x_2\ is\ A_{i2})\ AND\ \ldots\ AND\ (x_n\ is\ A_{in})\ THEN\ (y\ is\ B_i), \qquad (1)$$

where A_{ij} and B_i are fuzzy sets, x_j and y are fuzzy inputs and output. The meaning of the structure of a rule is the following:

$$IF\ Premise\ THEN\ Conclusion, \qquad (2)$$

where the premise consists of antecedents linked by fuzzy *AND* operators. The Centre of Gravity (COG) defuzzification method is used here because it is general and easy to compute. This method calculates the crisp output by the sums of the centre of gravities of the conclusions. Thus, a fuzzy inference system can compute output y of an input vector $\mathbf{x}$. The main purpose is to make the best solution possible for each input vector, therefore the optimum rule base need to be found.

2.1 The Encoding Method

The class of membership function is restricted to trapezoidal, as it is general enough and widely used. They are described by four parameters with the four breakpoints of the trapezium. Moreover the membership functions are identified by the two indices *i* and *j*. So, the membership function $A_{ij}(a_{ij},b_{ij},c_{ij},d_{ij})$ belongs to the i^{th} rule and the j^{th} input variable. $B_i(a_i,b_i,c_i,d_i)$ is the output membership function of the i^{th} rule. The relative importance of the j^{th} fuzzy variable in the i^{th} rule:

$$A_{ij}(x_j)=\begin{cases}\dfrac{x_j-a_{ij}}{b_{ij}-a_{ij}}, & if\ a_{ij}<x_j<b_{ij}\\ 1, & if\ b_{ij}\le x_j<c_{ij}\\ \dfrac{d_{ij}-x_j}{d_{ij}-c_{ij}}, & if\ c_{ij}\le x_j<d_{ij}\\ 0, & \text{otherwise}\end{cases}, \tag{3}$$

where $a_{ij} \le b_{ij} \le c_{ij} \le d_{ij}$ must hold.

So the encoding method of a fuzzy system with two inputs and one output, see in Fig. 1.

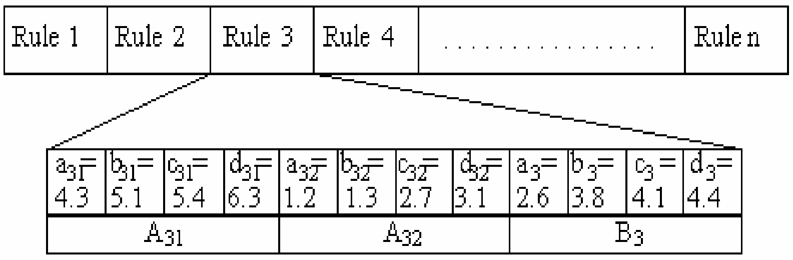

Fig. 1. Fuzzy rules encoded in a chromosome

For example, Rule 3 in Fig. 1 means:

IF x_1 is $A_{31}(4.3,5.1,5.4,6.3)$ and x_2 is $A_{32}(1.2,1.3,2.7,3.1)$ THEN y is $B_3(2.6,3.8,4.1,4.4)$ (4)

where A_{ij} and B_i mean the trapezoidal membership function with the four breakpoints [1].

3 Bacterial Algorithm

A recent kind of evolutionary method is called bacterial algorithm [1,2,7]. This approach includes two operations inspired by the microbial evolution phenomenon. The bacterial mutation operation optimizes the chromosome of a single bacterium, while the gene transfer operation provides the transfer of information between the bacteria in the population.

If we are applying the bacterial algorithm for the optimization of a fuzzy rule base, then one bacterium (individual) corresponds to one fuzzy system. In the evolutionary process, as the population evolves from generation to generation, better and better bacteria (rule base) can be obtained. The steps of the algortihm are described in this section.

3.1 Generating The Initial Population

First the initial (random) bacteria population is created. The population consists of N_{ind} chromosomes (bacteria). This means that all membership functions in the chromosomes must be randomly initialized. The initial number of rules in one chromosome is N_{rule}. So, $N_{ind}(k+1)N_{rule}$ membership functions are created, where k is the number of input variables in the given problem and each membership function has four parameters.

3.2 Bacterial Mutation

The bacterial mutation is applied to each chromosome one by one [1,7]. First, N_{clones} copies (clones) of the rule base are generated. Then a certain part of the chromosome is randomly selected and the parameters of this selected part are randomly changed in each clone (mutation). 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 parts of the chromosome have been mutated and tested. At the end the best rule base is kept and the remaining N_{clones} are discharged.

It is an important question how long is one part suffering mutation and what is the degree of the mutation (expressed as the relative size in terms of the interval). This approach allows both selecting more than one membership function and fine-tuning. The number of mutated membership functions and the mutation degree are external parameters of the bacterial mutation. If selecting more than one membership function is allowed then the local minima in the optimization process can be avoided.

3.3 Gene Transfer

The gene transfer operation allows the recombination of genetic information between two bacteria [1,7].

1. First the population must be devided into two halves. The better bacteria are called superior half, the other bacteria are called inferior half.

2. One bacterium is randomly chosen from the superior half, this will be the source bacterium, and another is randomly chosen from the inferior half, this will be the destination bacterium.

3. A “good” part from the source bacterium is chosen and this part can overwrite a not-so-good part of the destination bacterium or simple be added. A good part can be a fuzzy rule with a high degree of activation value [1].

4. Steps 1, 2, and 3 are repeated for N_{inf} times, where N_{inf} is the number of “infections” per generation.

3.4 Stop Condition

If the population satisfies a stop condition or the maximum generation number (N_{gen}) is reached then the algorithm ends, otherwise it returns to the bacterial mutation step.

4 B-spline Neural Networks

Belonging to the class of networks termed grid or lattice-based associative memories networks (AMN), B-spline Neural Networks are composed of three layers which are, a normalized input space layer, a basis functions layer and a linear weight layer. These will be described in the next subsections.

4.1 Normalized Input Layer

It is usually a grid on which the basis functions are defined. To define a grid in the input space, we need to set the values of the vectors of *knots*, one for each input axis. Each dimension has usually a different number of knots, generally placed at different positions. The interior knots, for the ith axis, are $\lambda_{i,j}, j = 1,\cdots,r_i$, and they are arranged in such a way that:

$$x_i^{\min} < \lambda_{i,1} \le \lambda_{i,2} \le \cdots \le \lambda_{i,r_i} < x_i^{\max} , \qquad (5)$$

where $x_i^{\min}$ and $x_i^{\max}$ are the minimum and maximum values of the i^{th} input, respectively. The j^{th} interval of the i^{th} input is denoted as $I_{i,j}$ and is defined as:

$$I_{i,j} = \begin{cases} [\lambda_{i,j-1} \quad \lambda_{i,j}[\ for \ j = 1, \cdots, r_i \\ [\lambda_{i,j-1} \quad \lambda_{i,j}] \ if \ j = r_i + 1 \end{cases}, \tag{6}$$

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

4.2 The Basis Function Layer

The output of the hidden layer is determined by a set of p basis functions defined on the n-dimensional grid. 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-splines neural networks, the *order* of the spline implicitly sets the size of the basis functions support and its shape. The univariate B-spline basis function of order k has a support, which is k intervals wide. 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:

$$N_k^j(x) = \left(\frac{x - \lambda_{j-k}}{\lambda_{j-1} - \lambda_{j-k}} \right) N_{k-1}^{j-1}(x) + \left(\frac{\lambda_j - x}{\lambda_j - \lambda_{j-k+1}} \right) N_{k-1}^{j}(x)$$
$$N_1^j(x) = \begin{cases} 1 & if \quad x \in I_j \\ 0 & otherwise \end{cases}, \tag{7}$$

To define multivariate basis functions, the *tensor product* of the univariate basis functions is performed. 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:

$$N_{\mathbf{k}}^j(\mathbf{x}) = \prod_{i=1}^{n} N_{\mathbf{k}_i,i}^j(\mathbf{x}_i), \tag{8}$$

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 $p = \prod_{i=1}^{n} (r_i + k_i)$. Because this number depends exponentially on the input dimension, B-splines are only applicable for problems where the input dimension is small (typically ≤ 5).

4.3 The Weight Layer

Since the output of an AMN is a linear combination of the outputs of the basis functions, linear coefficients must be defined, usually termed adjustable weights. As the mapping is linear, finding the weights is just a linear optimization problem. The output is given by:

$$y = \sum_{i=1}^{p} \mathbf{a}_i \mathbf{w}_i = \mathbf{a}^T \mathbf{w} , \tag{9}$$

where $\mathbf{a}_i = N_{\mathbf{k}}^i(x), \quad i = 1,\ldots,p$.

4.4 Sub-Modules

A way to overcome the "curse of dimensionality", is to cover all inputs by a linear sum of small sub-modules, each one with a lower input dimensionality, instead of using only one model covering all the inputs. The output of such a network is:

$$y(\mathbf{x}) = \sum_{u=1}^{n_u} S_u\left(\mathbf{x}_{\underline{u}}\right) , \tag{10}$$

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

4.5 B-Spline Neural Networks Design

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

1. The determination of the number of its sub-modules;
2. For each sub-model, the set of its inputs;
3. The order of the splines (for B-Spline NNs) 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 former points constitute a very complex combinatorial problem. There are different constructive algorithms to help on this task, such as the *ASMOD* (*Adaptive Spline Modelling of Observed Data*) algorithm, the *MARS (Multivariate Adaptive Regression Splines*) algorithm, the *LOLIMOT* algorithm and the genetic programming technique [4]. This paper proposes a new method for B-spline neural networks design, namely the *Bacterial Programming* approach.

5 Bacterial Programming

In Bacterial Algorithm described in Section 3 the individuals are represented by a sequence of numbers called chromosome. A common used approach in the field of evolutionary algorithms is Genetic Programming, which uses the same operators as GA, though the individuals are represented by an expression tree [6]. This tree is composed of the function (inner) nodes and the terminal (outer) nodes, representing the functions and their input arguments, respectively. Bacterial Programming combines the BA and the GP techniques, it uses its coding based on an expression tree, but the bacterial operators are applied in the evolutionary process [3].

When using Bacterial Programming as the “optimizer” of a B-spline Neural Network, sub-models must be *added* (+), sub-models of higher dimensionality must be *created* from smaller sub-modules (*), and sub-models of higher dimensionality must be *split* into lower dimensional sub-models (/). The node terminals do not represent only each *input variable*, but also the *spline order*, the *number of interior knots*, and *their location*. Fig. 2. shows the composition of the nodes in such a tree. For the tree given here, one would expect the model’s output to be given as a function of the addition of 3 submodels, two of each would be bidimensional, so that:

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

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

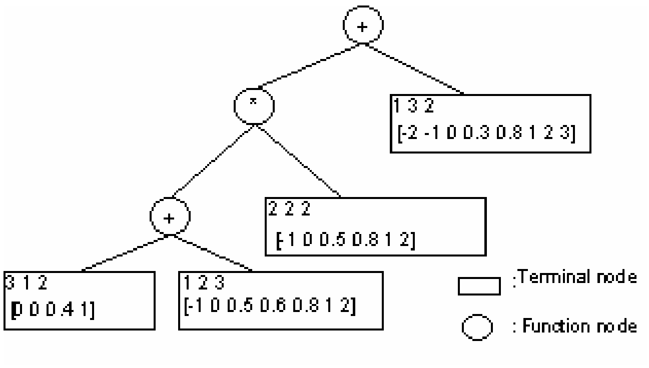

Fig. 2. A sample expression tree for B-spline networks

Obviously these functions and terminals must be well defined, so that they can receive any value returned by other set of functions or terminals in the lower subtrees, as their arguments.

Whenever models present a complexity higher than the number of samples within the training set, it is desirable to perform a convenient change in their structures so that these candidates may still participate in the evolution and not be completely discarded. Thus, 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 sub-tree that corresponds to the least complex ramification.

Steps 1 and 2 are performed until the candidate is designated valid. By this we mean that branches, or subtrees, may be discarded.

5.1 The Evolutionary Process

The evolutionary process of BP is the same as in the case of BA. So, it is involving the following steps.

1. The creation of an initial 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.

The terminal criterion is defined as the maximum number of generations allowed.

5.2 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 strings, it requires a tree structure, composed of *function* and *terminal* nodes. This tree structure, as well as the characteristics of the nodes, evolves from generation to generation. Next, the bacterial operators will be described in the case of BP.

5.3 Bacterial Mutation

The basic idea of the bacterial mutation operation is the same as in BA (See Section 3.2). However, because coding is given by an expression tree, there are two types of parts: function and terminal parts. When a certain part of the chromosome is randomly selected then this part can be either function or terminal part, thus there are two types of bacterial mutation.

The following figures illustrate the mutation procedure. From Fig. 3., it can be seen that after a function mutation on a node, all the subtree beneath is replaced by a new randomly generated one, using the "grow" method [6]. However, terminal mutation affects only one of the nodes. In the B-spline structure optimization task, the mutation of a terminal can be one 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 operation is executed.

In bacterial programming, mutation is applied once to the part selected, meaning that neither the nodes selected nor the subtree will be chosen again for mutation, in the same generation.

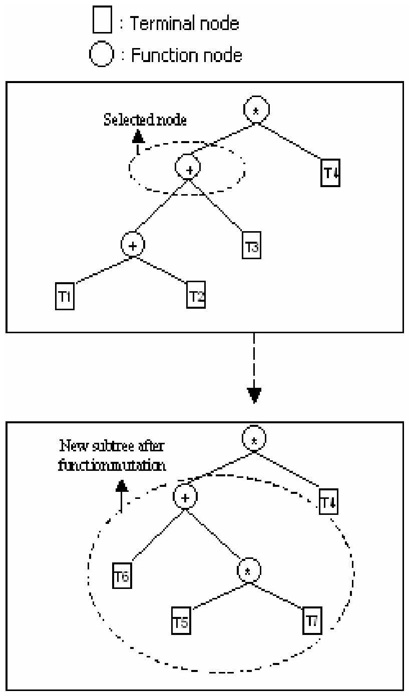

Fig. 3. Mutation on a function part: the individual's selected node subtree is changed randomly

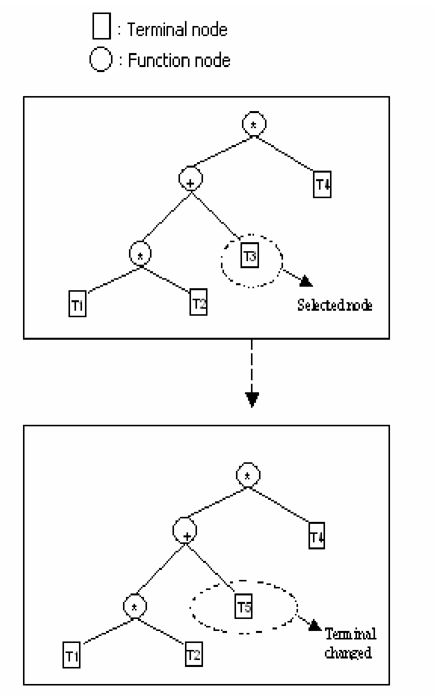

Fig. 4. Mutation on a terminal part: only the selected node changed randomly

5.4 Gene Transfer

The aim of the gene transfer operation is to exchange of genetic information between two bacteria. This procedure is somewhat similar to the crossover operation used in genetic programming. The basic idea of the gene transfer operation is the same as in BA (See Section 3.3). In BP, a node in the source bacterium is selected, and the corresponding subtree will overwrite a random selected subtree of the destination bacterium.

5.5 Bacterium Evaluation

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

In our case, the criterion is defined as:

$$BIC = m\ln\left(RMS\right) + n\ln\left(m\right), \tag{11}$$

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

5.6 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:

1. As it is often the case, the larger the size of the population (N_{ind}), and the number of generations (N_{gen}) employed, the better were the results obtained;
2. The parameter of bacterial mutation is the number of clones (N_{clones}). The larger N_{clones}, the more effective is the bacterial mutation.

Conclusion

Bacterial optimization techniques were presented in this paper. These methods apply the bacterial operators instead of the original genetic operators. So, while the bacterial mutation is working on one individual, and tries to optimise this bacterium, the gene transfer is applied to the whole bacteria population, avoiding the local minima solutions. The bacterial approach can be applied for fuzzy rule base optimization and for the design of neural networks too. Bacterial Programming offers suitable performance for the design phase of the B-spline networks, which is originally a highly complex computational task.

Acknowledgement

This paper is partly based on research done in collaboration with Mr. Cristiano Cabrita and Professor Antonio Ruano (University of Algarve, Faro, Portugal).

References

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

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

[3] Cabrita, C., Botzheim, J., Ruano, A. E., Kóczy, L. T: Design of B-spline neural networks using a bacterial programming approach, International Joint Conference on Neural Networks, Budapest, Hungary, 2004, pp. 2313-2318.

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

[5] Holland, J. H.: Adaptation in Nature and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, MIT Press, Cambridge, 1992.

[6] Koza, J. R.: Genetic Programming, On the programming of computers by means of natural selection, 6th ed., MIT. 1998.

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

[8] Zadeh, L. A.: Outline of a new approach to the analysis of complex systems and decision processes, IEEE Tr. Systems, Man and Cybernetics **3** (1973), pp. 28-44.