World Automation Congress © 2010 TSI Press.

HIERARCHICAL FUZZY SYSTEM CONSTRUCTION APPLYING GENETIC AND BACTERIAL PROGRAMMING ALGORITHMS WITH EXPRESSION TREE BUILDING RESTRICTIONS

KRISZTIÁN BALÁZS[1], JÁNOS BOTZHEIM[2] AND LÁSZLÓ T. KÓCZY[1,3]

[1]*Department of Telecommunications and Media Informatics,*
Budapest University of Technology and Economics, Budapest, Hungary
(e-mail: balazs@tmit.bme.hu, koczy@tmit.bme.hu)

[2]*Department of Automation, Széchenyi István University, Győr, Hungary*
(e-mail: botzheim@sze.hu)

[3]*Institute of Informatics, Electrical and Mechanical Engineering, Faculty of Engineering Sciences,*
Széchenyi István University, Győr, Hungary
(e-mail: koczy@sze.hu)

ABSTRACT— In this paper various restrictions are proposed in the construction of hierarchical fuzzy rule bases by using Genetic and Bacterial Programming algorithms in order to model black box systems defined by input-output pairs, i.e. to solve supervised machine learning problems.
The properties (learning speed, accuracy) of the established systems are observed based on simulation results and they are compared to each other.

Key Words: hierarchical fuzzy systems, genetic programming, bacterial programming

1. INTRODUCTION

Our previous work [1] proposed methods for constructing hierarchical fuzzy rule bases in order to model black box systems defined by sets of input-output pairs, i.e. to solve supervised machine learning problems. The resulting hierarchical rule bases mean knowledge bases constructed by using Genetic and Bacterial Programming algorithms. This paper surveys these hierarchical fuzzy rule base constructing methods and proposes various expression tree building restrictions, which can be applied during the optimization process to obtain better results.

There are two key elements in machine learning systems: the learning architecture and the learning algorithm applied during the learning process. Hence, in order to establish a learning system having favorable properties (low knowledge complexity, high accuracy, fast learning process, etc.) these elements must be chosen so that they outperform other alternatives of knowledge bases and learning algorithms.

For interpretability purposes, i.e. after the learning process to have a quasi-interpretable knowledge base, this paper considers only fuzzy rule bases as knowledge bases.

The classical approaches of fuzzy control deal with dense rule bases where the universe of discourse is fully covered by the antecedent fuzzy sets of the rule base in each dimension, thus for every input there is at least one activated rule. The main problem is the high computational complexity of these approaches.

If a fuzzy model contains k variables and maximum T linguistic terms in each dimension, the order of the number of necessary rules is $O(T^k)$. This expression can be decreased either by decreasing T, or k, or both. The first method leads to sparse rule bases and rule interpolation, and was first introduced by Kóczy and Hirota (see e.g. [2], [3]). The second, more effectively, aims to reduce the dimension of the sub-rule bases by using meta-levels or hierarchical fuzzy rule bases.

The scope of this paper focuses on the second mentioned way of complexity reduction, namely, the application of hierarchical fuzzy rule bases.

Nature inspired some evolutionary optimization algorithms suitable for global optimization e.g. Genetic Algorithm (GA) developed by Holland [4] 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. Bacterial Evolutionary Algorithm (BEA) [5] inspired by the microbial evolution phenomenon is similar, but based on results in the literature appearing more effective technique than GA (e.g. [6], [7]). Approaches like Genetic Programming (GProg) [8] and Bacterial Programming (BProg) [9], present an alternative to the former algorithms. These techniques use the same operators as GA and BEA, respectively, though they require expression trees for gene representation, which could model hierarchical fuzzy rule bases, for example.

Evolutionary algorithms are useful tools for identification problems; they can help with finding a good compromise between the accuracy and the complexity of the model. In case of non-hierarchical rule bases, genetic and bacterial approaches were successfully applied for fuzzy rule based learning by various authors (see e.g. [7]). This is the reason why Genetic and Bacterial Programming is applied in our work as the learning algorithms that optimize the knowledge base.

Since so far no method has been invented to obtain the main properties (efficiency, accuracy, etc.) of such complex systems exactly, they can be figured out mostly by simulation. Therefore, in order to discover the capabilities of the established hierarchical fuzzy rule based learning systems, simulation runs were carried out.

The next section gives a brief overview of the algorithms and techniques used. The third section describes a method for constructing and optimizing hierarchical fuzzy rule bases by Genetic and Bacterial Programming approaches and proposes some expression tree building restrictions. The simulation results and the observed behavior will be discussed in the fourth section. Finally, in the last section we summarize our work and draw some conclusions.

2. OVERVIEW OF THE TECHNIQUES AND ALGORITHMS

In order to carry out this investigation, it is necessary to have familiarity with some theoretical fields including the concept of hierarchical fuzzy rule bases, structure constructing evolutionary optimization techniques (GProg, BProg) and machine learning.

The following subsections aim to give a brief overview of some important points of these theoretical aspects, which will be referred to later repeatedly in the paper.

2.1 Hierarchical fuzzy rule based systems

The basic idea of using hierarchical fuzzy rule bases is the following. Often the multi-dimensional input space $X=X_1\times X_2\times\ldots\times X_m$ can be decomposed, so that some of its components, e.g. $Z_0=X_1\times X_2\times\ldots\times X_p$ determine a subspace of X ($p < m$), so that in Z_0 a partition $\Pi=\{D_1, D_2,\ldots, D_n\}$ can be determined: $\bigcup_{i=1}^{n} D_i = Z_0$.

In each element of Π, i.e. D_i, a sub-rule base R_i can be constructed with local validity. In the worst case, each sub-rule base refers to exactly $X/Z_0=X_{p+1}\times\ldots\times X_m$. The complexity of the whole rule base $O(T^m)$ is not decreased, as the size of R_0 is $O(T^p)$, and each R_i, $i > 0$, is of order $O(T^{m-p})$, $O(T^p)\times O(T^{m-p})= O(T^m)$. A way to decrease the complexity would be finding in each D_i a proper subset of $\{X_{p+1}\times\ldots\times X_m\}$, so that each R_i contains only less than m-p input variables. In some concrete applications in each D_i a proper subset of $\{X_{p+1},\ldots, X_m\}$ can be found so that each R_i contains only less than m-p input variables, and the rule base has the following structure:

R_0:	**If** z_0 is D_1 **then** use R_1
	…
	If z_0 is D_n **then** use R_n
R_1:	**If** z_1 is A_{11} **then** y is B_{11}
	…
	If z_1 is A_{1r1} **then** y is B_{1r1}
…	
R_n:	**If** z_n is A_{n1} **then** y is B_{n1}
	…
	If z_n is A_{nrn} **then** y is B_{nrn}

where $z_i \in Z_i$, $Z_0\times Z_i$ being a proper subspace of X for $i = 1,..,n$.

If the number of variables in Z_i is $k_i < m-p$ and $\max_{i=1}^{n} k_i = K < m-p$, then the resulting complexity will be $O(T^{p+K}) < O(T^m)$, so the structured rule base leads to a reduction of the complexity.

The task of finding such a partition is often difficult, if not impossible, (often such a partition does not exist at all), however there are cases when, locally, some variables unambiguously dominate the behavior of the system, and consequently the omission of the other variables allows an acceptably accurate approximation.

2.2 Genetic Programming

There are 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 (GA) [4] an individual is represented by a sequence of numbers, for example a sequence of bits. This sequence is called chromosome. GA uses three operators: selection, crossover, and mutation. A more recent approach is Genetic Programming (GProg) [8], which uses the same operators as GA, though the individuals are represented by the so called expression tree. This tree is composed of non-terminal (inner or function) nodes and terminal nodes (leaves).

The evolutionary process involves the following steps:

1. creation of an initial population
2. evaluation of the candidates
3. application of genetic operators:
   - **selection**: pairs of trees are selected based on their fitness for reproduction
   - **crossover**: nodes in the trees are selected at random and the sub-trees belonging to the selected nodes are exchanged producing 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 the inner properties of the node
   - **replacement**: substituting the created offsprings to the population
4. test of the termination criteria (e.g. reaching the time limit) and if the terminal criterion is achieved, the algorithm stops, otherwise it continues from step 2

2.3 Bacterial Programming

Compared to GA, a slightly different evolutionary technique is called Bacterial Evolutionary Algorithm (BEA). This algorithm was introduced by Japanese researchers in the late 90's [5]. Compared to GA, BEA contains two new operators: bacterial mutation and gene transfer. Both of these operators were inspired by bacterial evolution. The bacterial mutation performs local optimization, whereas 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, a new technique was proposed, named Bacterial Programming (BProg) [9].

The evolutionary process of BProg involves the following steps:

1. creation of an initial population
2. application of bacterial operators:
   - **bacterial mutation**: this is performed by either replacing a node selected at random by a sub-tree generated randomly or by changing the inner properties of the node
   - **gene transfer**: nodes in the trees are selected at random in the selected superior individual and the sub-trees belonging to the selected nodes are copied to the selected inferior individual
3. test of the termination criteria (e.g. reaching the time limit) and if the terminal criterion is achieved, the algorithm stops, otherwise it continues from step 2

2.4 Supervised machine learning

Supervised machine learning [10] means a process where parameters of a *model* are being adjusted so that its behavior becomes similar to the behavior of the *system*, which is to be modeled. Since the behavior can be characterized by input-output pairs, the aim of the learning process can be formulated so that the model should give similar outputs for the input as the original system does.

The model can be, for example, a simple function (e.g. a polynomial function), where the parameters are the coefficients, or it can be a neural network, where the parameters are the weights, or it can be a fuzzy rule base together with an inference engine [11]. In our work we applied a hierarchical fuzzy rule base containing trapezoidal membership functions.

If a function $\phi(\mathbf{x})$ denotes the system and $f(\mathbf{x},\mathbf{p})$ denotes the model, where $\mathbf{x}\in X$ is the input vector and $\mathbf{p}$ is the adjustable parameter vector, the previous requirement can be expressed as follows:

$$\forall \mathbf{x} \in \mathbf{X} : \phi(\mathbf{x}) \approx f(\mathbf{x}, \mathbf{p})$$

In a supervised case learning happens using a set of training samples (input-output pairs). If the number of samples is m, the input in the i^{th} sample is $\mathbf{x}_i$, the desired output is $d_i = \phi(\mathbf{x}_i)$ and the output of the model is $y_i = f(\mathbf{x}_i,\mathbf{p})$, the following formula can be used:

$$\forall i \in [1, m] : d_i \approx y_i$$

The error (ε) shows how similar the modeling system to the system to model is. It is the function of the parameter vector, so it can be denoted by $\varepsilon(\mathbf{p})$. Let we use a widely applied definition for the error, the Mean of Squared Error (MSE):

$$\varepsilon(\mathbf{p}) = \frac{\sum_{i=1}^{m} (d_i - y_i)^2}{m}$$

Obviously, the task is to minimize this $\varepsilon(\mathbf{p})$ function. It can be done by numerical optimization algorithms. This way, machine learning problems can be traced back to optimization problems.

3. THE ENCODING METHOD

This section describes the method how the individuals represent the applicant hierarchical fuzzy rule bases, i.e. how the rule bases are encoded, furthermore some restrictions we applied during the expression tree building.

The non-terminal (inner) nodes hold the antecedent parts of one or more rules. In case of each non-terminal node the number of considered input dimensions (number of *decision variables*) is a random value, whose maximum can be parameterized. The consequent part of a rule is a child node of the particular node, which can be either a non-terminal or a terminal node. Thus, the number of rules a non-terminal node holds is determined by the number of the children of the particular node. The number of children of the nodes is a random value, whose maximum, i.e. the maximum number of rules belonging to a node, can be parameterized.

Each terminal node (leaf) holds one single fuzzy set, as a conclusion (output) set.

Since there can be non-terminal nodes as well as terminal nodes on the same levels, meta-rules and rules can appear together.

Considering this encoding method, there are two obvious points where the restrictions of the rule base creation can take place. These are the parameterizations of the maximum number of decision variables and the maximum number of children. Decreasing these values decreases the degrees of freedom of the rule base, which means lower expression power, but a smaller state space to search during the optimization process. This may result two opposite effects: lower accuracy and higher convergence speed. Whether the favorable or the unfavorable effect is the stronger one, and whether it is or is not worth to apply such restrictions during the construction of the expression trees; these questions are investigated by carrying out simulation runs and will be discussed in the next section. In the simulations the applied building restrictions (**BR**s) are as follows:

1. **BR$_{1,2}$**: only 1 decision variable and at most 2 children
2. **BR $_{1,4}$**: only 1 decision variable and at most 4 children
3. **BR $_{3,2}$**: at most 3 decision variable and at most 2 children
4. **BR $_{3,4}$**: at most 3 decision variable and at most 4 children

4. SIMULATION RESULTS

Simulation runs were carried out in order to compare the established learning systems. The learning process was carried out for both GProg and BProg, and for all the expression tree building restrictions defined in the previous section.

In our work a six-dimensional learning problem was applied that was also used by Nawa and Furuhashi to evaluate the performance of Bacterial Evolutionary Algorithm [5]. This problem is to approximate the following function:

$$f_{6\dim} = x_1 + \sqrt{x_2} + x_3 x_4 + 2e^{2(x_5 - x_6)},$$
$$x_1, x_2 \in [1,5], \quad x_3 \in [0,4], \quad x_4 \in [0,0.6], \quad x_5 \in [0,1], \quad x_6 \in [0,1.2]$$

In the simulations the parameters had the following values, because after a number of test runs these values seemed to be suitable. The number of individuals in a generation was 8 in both GProg and BProg. In case of GProg the selection rate was 0.3 and the mutation rate was 0.1, in case of BProg the number of clones was 5 and 4 gene transfers were carried out in each generation. GProg applied elitist strategy. The optimization processes stopped after reaching the time limit, which was 1000 seconds. The numbers of training samples were 200 in the learning processes.

In case of both algorithms for each building restrictions 10 runs were carried out. Then we took the mean of the obtained values.

During the runs the fitness values were calculated based on the MSE values (measured on the training samples) as follows:

$$F = \frac{10}{MSE + 1} = \frac{10m}{\sum_{i=1}^{m}(d_i - y_i)^2 + m}$$

The fitness values of the best individuals in terms of time during the simulation runs are presented in Figure 1. The dashed lines denote the results given by GProg, whereas the solid lines present the fitness values given by BProg.

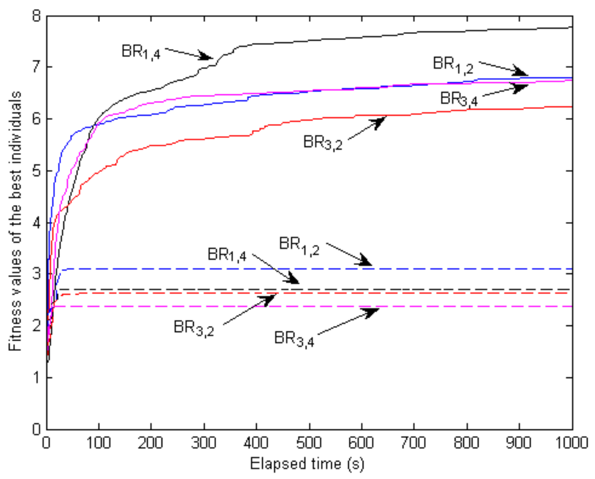

Figure 1. Results for the six-dimensional learning problem

Comparing the applied optimization techniques the following facts can be observed. BProg always outperformed GProg considering arbitrary restrictions, moreover, unlike GProg that stuck in local optima and whose convergence speed became zero in 100 seconds, BProg had a positive convergence speed even after 1000 seconds and continued advancing to better fitness values.

From the point of view of the expression tree building restrictions the simulation runs highlighted the following properties.

In case of GProg applying the strictest building restriction, i.e. $\mathbf{BR_{1,2}}$ (only 1 decision variable and at most 2 children) led to the best performance, whereas in case of BProg $\mathbf{BR_{1,4}}$ (only 1 decision variable and at most 4 children) appeared to be the best choice.

Applying restrictions in the maximum number of decision variables, i.e. using $\mathbf{BR_{1,4}}$ instead of $\mathbf{BR_{3,4}}$ or $\mathbf{BR_{1,2}}$ instead of $\mathbf{BR_{3,2}}$, resulted higher fitness values regardless of the optimization method.

In case of GProg restricting the maximum number of children, i.e. applying $\mathbf{BR_{3,2}}$ instead of $\mathbf{BR_{3,4}}$ or $\mathbf{BR_{1,2}}$ instead of $\mathbf{BR_{1,4}}$, resulted in better performance, however in case of BProg after applying this type of restriction the results became worse.

It is interesting that applying $\mathbf{BR_{3,4}}$ or $\mathbf{BR_{1,2}}$ in case of BProg led to nearly the same results, whose reason is that restricting the maximum number of children decreased the performance, whereas restricting the maximum number of decision variables increased it.

To sum up the simulation results: allowing only one decision variable in nodes of the expression tree increased the performance generally, whereas the effect of decreasing the maximum number of children depended on the applied optimization technique. Therefore, the stricter the restriction, the better the performance of GProg regardless of the type of the restriction, however in case of BProg the applied restriction must be chosen carefully, because only the restriction of the maximum number of decision variables led to better performance.

5. CONCLUSIONS

Our work proposed various similar ways for constructing hierarchical fuzzy rule bases in order to solve supervised machine learning problems. The rule bases were constructed by using Genetic and Bacterial Programming techniques with various expression tree building restrictions.

Simulation runs were carried out to compare the obtained learning systems. In case of both optimization algorithms allowing only one decision variable in the nodes of the expression tree increased the performance; however the effect of allowing fewer children depended on the applied optimization technique.

Further research may aim to improve both the encoded learning architecture and the applied optimization methods (e.g. using gradient steps during the optimization process).

The framework developed in this paper will also be exploited in the construction of expert systems for real life applications, such as problems in building energetic.

ACKNOWLEDGMENTS

This paper was supported by the National Scientific Research Fund Grant OTKA K75711, a Széchenyi István University Main Research Direction Grant and the Social Renewal Operation Programme TÁMOP-4.2.2 08/1-2008-0021. János Botzheim was also supported by the Baross Gábor Programme of the Hungarian National Office for Research and Technology (ENERGOPT Project: BAROSS-ND07-INRG5-07-2008-0059).

REFERENCES

1. K. Balázs, J. Botzheim, L. T. Kóczy, “Hierarchical Fuzzy System Modeling by Genetic and Bacterial Programming Approaches”, *World Congress on Computational Intelligence, WCCI 2010*, Barcelona, Spain, 2010, accepted for publication.
2. L. T. Kóczy and K. Hirota, “Approximate reasoning by linear rule interpolation and general approximation”, *International Journal of Approximate Reasoning*, 9, 1993, pp. 197-225.
3. L. T. Kóczy and K. Hirota, “Interpolative reasoning with insufficient evidence in sparse fuzzy rule bases”, *Information Sciences*, 71, 1993, pp. 169-201.
4. J. H. Holland, “Adaption in Natural and Artificial Systems”, *The MIT Press*, Cambridge, Massachusetts, 1992.
5. N. E. Nawa and T. Furuhashi, “Fuzzy system parameters discovery by bacterial evolutionary algorithm”, *IEEE Transactions on Fuzzy Systems*, 7(5), 1999, pp. 608-616.
6. K. Balázs, J. Botzheim, L. T. Kóczy, “Comparison of Various Evolutionary and Memetic Algorithms”, *Proceedings of the International Symposium on Integrated Uncertainty Management and Applications, IUM 2010*, Ishikawa, Japan, 2010, accepted for publication.
7. K. Balázs, J. Botzheim, L. T. Kóczy, “Comparative Analysis of Interpolative and Non-interpolative Fuzzy Rule Based Machine Learning Systems Applying Various Numerical Optimization Methods”, *World Congress on Computational Intelligence, WCCI 2010*, Barcelona, Spain, 2010, accepted for publication.
8. J. R. Koza, “Genetic Programming: On the Programming of Computers by Means of Natural Selection”, *The MIT Press*, San Francisco, 1992.
9. J. Botzheim, C. Cabrita, L. T. Kóczy, and A. E. Ruano, “Genetic and bacterial programming for B-spline neural networks design”, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 11(2):220–231, February 2007.
10. E. Alpaydin, “Introduction to Machine Learning”, *The MIT Press*, San Francisco, 2004, 445 p.
11. D. Driankov, H. Hellendoorn, M. Reinfrank, “An Introduction to Fuzzy Control”, *Springer*, New York, 316 p.