F.S.A.I., Vol. 9, Nos. 1–3, pp. 35–46, 2003

Changing the Fuzzy Rules in Crossover Stage (Genetic Algorithm Optimization)

Lucian V. Boiculese*, Hariton N. Costin**

* Faculty of Medicine, "Gr.T.Popa" Univ. of Medicine and Pharmacy, Iasi
** Faculty of Medical Bioengineering, "Gr.T.Popa" Univ. of Medicine and Pharmacy, Iasi

Abstract: The main problem in developing a system consists in optimizing it in order to learn proper rules. The error surface frequently presents local minima that impede the convergence to the best solution. Methods based on gradient descent procedure, like back-propagation, frequently did not manage to cope with local minima traps. In the contrary, techniques based on probabilistic decision usually find the best solution paying the cost of long time training. A fuzzy system was developed and trained with a genetic algorithm procedure. A new crossover operator that changes the system rules was proposed in order to increase the speed in convergence. The method is tested in a classification application.

1. Introduction

Fuzzy systems were developed as a natural evolution of a scientific field that has proven its benefits. These systems deal with imprecision related to belonging to a specific group [1]. The reason of using fuzzy mathematics is motivated by the fact that real things are too difficult to be defined by precise descriptions. Also the human inference deals with vague or fuzzy information. There are many situations in which trying to develop a system using crisp mathematical formulas, yielded unacceptable results or at least, cumbersome to develop.

Designing a fuzzy system requires defining the inputs, the outputs and the rule base that relate the proper output to an certain input. The inputs and outputs are characterized by membership functions that define the system parameters. A Gaussian membership function has two parameters: the mean value and the standard deviation. These parameters must be optimized during the training of the system.

Genetic algorithms (GAs) represent a mathematical search strategy using the statistical methods of selection and combination. This technique is used in a large number of fields like control, modeling, classification, optimization, where the error space contains many local minima that produce difficulties in the optimal solution achievement [2, 3, 4].

Using specific genetic operators (selection, crossover, mutation, and inversion) described by probabilistic decisions, GA proves to be suitable in optimization problems in which methods depending on gradient direction often fail.

Some characteristics of genetic algorithms are:

- Usually they make a codification of the parameter values.
- They work on a set of points (not on a single point).

- It is not necessary to have the differential expression of the transfer function that is involved or other mathematical conditions imposed.
- They use probabilistic transition rules, not deterministic ones.
- Working with a large number of point means that there is information in searching for solutions.

The need not to know the derivative or integral or other information about the function involved in optimization increases the GA robustness and generalization. Also a probabilistic transition (that is why they are called "blind techniques") improves the chances of coping with local minima traps and easily finds new points of possible solution.

The fuzzy system was optimized by a GA technique. The chromosome was made up of the system parameters. Therefore, a chromosome represents an entire system. A new crossover operator, that changes the fuzzy rules between chromosomes, was proposed and tested in a classification field. System implementation and results are subsequently presented.

2. The Fuzzy System

A fuzzy system has several inputs that represents the premise information, several outputs that denote the inferred information and a rule base that is the core of the system. To compute the output for a specific input, the information is passed through a set of stages, which will be subsequently described, according to [1].

2.1. The fuzzifier

The fuzzifier is the first operation used by the most applications with fuzzy systems. It represents the interface between the existent world that usually give us real (crisp) values and the fuzzy system. Suppose that there is a point denoted by x_{in} in the universe of discourse. The fuzzifier assign a membership function that characterizes the fuzzy set A that corresponds to the x_{in} input value. It is a function that transforms a crisp number in a fuzzy set. The simplest method is the singleton fuzzifier that is characterized by the next membership function:

$$\mu_A(x) = \begin{cases} 1 \ \ if \quad x = x_{in} \\ 0 \ \ otherwise \end{cases} \tag{1}$$

$\mu_A(x)$ – is the membership function; A – is the fuzzy set; x – is the n-dimensional input variable; x_{in} – is a specific value of the input.

For Gaussian fuzzifier the following formula is applied:

$$\mu_A(x) = e^{-\left(\frac{x_1 - x_1^A}{a_1}\right)^2} * \ldots. * \ e^{-\left(\frac{x_n - x_n^A}{a_n}\right)^2} \tag{2}$$

$x = (x_1, x_2, ..., x_n)$ – is the input variable;
x_i^A, $i = 1, ..., n$ – is the mean value for the set A;
a_i, $i = 1, ..., n$ – is the standard deviation for the Gaussian function;
$*$ – is the t-norm (the "and" operator) used to compute the membership function.

2.2. The fuzzy proposition calculus

The information in a fuzzy system is memorized in a set of rules. An multiple input multiple output system can be decomposed in a multiple input single output system. The number of single output system rules is equal with the number of output variables. Subsequently any rule may be decomposed in a set of canonical rules. The canonical rules uses atomic propositions connected by the "and" operator. The effect of the "or" operator generates new canonical rules. In conclusion any set of rules can be decomposed in a set of canonical rules.

An atomic fuzzy proposition is a single statement for instance "pressure is Low" or "p is L". A canonical rule reads, for example:

IF x is *M* and y is *H* and z is *S* THEN w is *M*

where x, y, z, w are input variables and *M*, *H*, *S* are fuzzy sets (medium, high, small).

A general form of the rule is next presented:

IF fuzzy proposition1 (hypothesis) THEN fuzzy proposition2 (conclusion)

For the fuzzy proposition the "and" operator will be interpreted with a t-norm denoted by $*$. Therefore the membership function for the fuzzy premise proposition in the first rule will be computed as:

$$\mu_{M \cap H \cap S}(x, y, z) = \mu_M(x) * \mu_H(y) * \mu_S(z) \tag{3}$$

The most utilized t-norms are the algebraic product and the "min" operator.

2.3. The implication

The rule expressed in the forward paragraph is interpreted as an implication and is written in the next form: FP1 $\rightarrow$ FP2. Here FP means fuzzy proposition.

In classical propositional calculus the equivalence is $A \rightarrow B$ (were *A* and *B* are nonfuzzy sets). This represents the rule: "IF x is *A* THEN y is *B*" and the truth value is computed as:

$$A \rightarrow B = \overline{A} \vee B \tag{4}$$

That is equivalent with the following formula:

$$A \to B = (A \wedge B) \vee \overline{A} \tag{5}$$

Fuzzy implication procedure may be figured out by applying different types of t-norms for the intersection and different types of s-norm for the union. If one uses a specific t-norm then the corresponding s-norm must be utilized in order to comply with the De Morgan's equalities. Mamdani proposed a new way in computing the implication based on the minimum or the product operator.

For the minimum operator the implication becomes:

$$\mu_Q(x, y) = \min\left[\mu_{fp1}(x), \mu_{fp2}(y)\right] \tag{6}$$

For the algebraic product composition the implication is given by:

$$\mu_Q(x, y) = \mu_{fp1}(x) \cdot \mu_{fp2}(y) \tag{7}$$

2.4. The fuzzy inference

Modus Ponens is the most utilized inference. If one has a proposition *A* (premise) and an implication (that is also a proposition and is a part of the premise) *A*→*B*, then the proposition *B* (conclusion) must be inferred. This is commonly written as:

$$(A \wedge (A \to B)) \to B \tag{8}$$

In the above formula *A* and *A*→*B* are premises and *B* is the conclusion.

In formula (8), the intersection is interpreted as a *t*-norm and therefore next formula is established:

$$t(\mu_{A1}(x), \mu_{A\to B}(x, y)) \tag{9}$$

Formula (9) is defined on a two dimensional space (x, y). The output depends only on y; so, applying the projection on the y space the final result is reached.

As a result with a fuzzy implication *A*→*B* and using the Modus Ponens inference for an input fuzzy variable A_1, the output B_1 membership function will be computed as:

$$\mu_{B_1}(y) = \sup_{x \in U} t\left[\mu_{A_1}(x), \mu_{A\to B}(x, y)\right] \tag{10}$$

2.5. The inference engine

Usually the knowledge base has many rules. For a specific input, more than one rule may be fired and thus the inferred output must be mixed to reflect the correct system answer. The individual rule based inference computes the output for each of rule. Subsequently, the output of the fuzzy system is obtained by combining the whole outputs fuzzy sets. Usually the combination is realized with the s-norm by the concept that rules are interpreted as independent conditional statements.

Formula (11) represents the final output membership function computation for a system with M rules:

$$\mu_{B_F}(y) = \mu_{B_1}(y) \dot{+} \ldots \dot{+} \mu_{B_M}(y) \tag{11}$$

$\mu_{B_F}(y)$ - is the computed output (final) membership function; $\dot{+}$ - is the s-norm used for computation; $\mu_{B_i}(y)$ - is the membership function of the i^{th} rule.

2.6. The defuzzifier

The final block, the defuzzifier, realizes the interface between the fuzzy set and the real value. The aim of this final operation is to indicate the real point in the output domain V that best corresponds to the inferred fuzzy set. One of the most used formulas is based on the center of gravity:

$$y_1 = \frac{\int_V y \cdot \mu_{BF}(y)\, dy}{\int_V \mu_{BF}(y)\, dy} \tag{12}$$

Because it is computationally intensive it is seldom used in real problems. The center of average defuzzifier tries to overcome this disadvantage with a simplified formula:

$$y = \frac{\sum_{i=1}^{M} y_i \cdot w_i}{\sum_{i=1}^{M} w_i} \tag{13}$$

y – is the output real value; w_i – is the activation of the premise of the i-th rule; y_i – is the center of the i-th output membership function.

2.7. The tested system

Using formula (10) for the inference, formula (11) for the fuzzy engine and applying the center of gravity for the defuzzifier block (12), the output becomes:

$$y_1 = \frac{\int_V y \cdot \mathrm{Y}_{i=1}^{M} \mu_{Bi}(y)}{\int_V \mathrm{Y}_{i=1}^{M} \mu_{Bi}(y)} = \frac{\int_V y \cdot \mathrm{Y}_{i=1}^{M} \sup_{x\in U} t\left[\mu_{A_1}(x)\, ,\, \mu_{A\to B}(x\, ,\, y)\right]}{\int_V \mathrm{Y}_{i=1}^{M} \sup_{x\in U} t\left[\mu_{A_1}(x)\, ,\, \mu_{A\to B}(x\, ,\, y)\right]} \tag{14}$$

M - represents the rules number, A_1 is the fuzzy input, U is the input universe of discourse, V is the output universe of discourse and $\underset{i=1}{\overset{M}{\mathrm{Y}}}$ is the s-norm.

Intensive computation will be needed in a software implementation of the formula (14). For a simplified system, one may start with the defuzzifier from formula (13). The w_i value represents the output activation of the i rule. It is the maximum value of the output membership function inferred from the input. Considering that for implication the Mamdani algebraic product is used and the inference type is Modus Ponens, then from formula (10) we obtained:

$$\mu^l_{Bin}(y) = \sup_{x\in U} t[\mu_{Ain}(x), \mu_{A\to B}(x,y)] = \sup_{x\in U} t[\mu_{Ain}(x), \mu^l_A(x)\cdot\mu^l_B(y)], \quad (15)$$

where: Ain is the actual fuzzy input, A is the fuzzy premise, B is the fuzzy output, l is the rule number, $\mu(x)$ is the membership function.

Using the singleton fuzzifier method expressed in (1) and algebraic product for the t-norm, formula (15) becomes:

$$\begin{aligned}\mu^l_{Bin}(y) &= \sup_{x\in U}\left[\mu_{Ain}(x)\cdot\mu^l_A(x)\cdot\mu^l_B(y)\right] = \sup_{x=xin}\left[1\cdot\mu^l_A(x)\cdot\mu^l_B(y)\right] = \\ &= \mu^l_A(x_{in})\cdot\mu^l_B(y)\end{aligned} \quad (16)$$

Allowing the output membership function to be considered like a singleton fuzzifier with the single point in $\bar{y}_l$ (for rule number *l*), one may finally compute the output activation for the rule:

$$\mu^l_{Bin}(y) = \mu^l_A(x_{in})\cdot\mu^l_B(y) = \mu^l_A(x_{in})\cdot\mu^l_B(\bar{y}_l) = \mu^l_A(x_{in})\cdot 1 = \mu^l_A(x_{in}) \quad (17)$$

Using formula (3), the following expression computes the fuzzy output for a number of M rules:

$$y_{out} = \frac{\sum_{l=1}^{M} \bar{y}_l \cdot \mu^l_A(x_{in})}{\sum_{i=1}^{M} \mu^l_A(x_{in})} \quad (18)$$

Here, $l = \{1, ..., M\}$ – is the rule number, $\bar{y}_l$ – is the center of gravity of the output membership function of the $l-$th rule, x_{in} – is the real input system value, $\mu^l_A(x_{in})$ – is the membership function of the input of the $l-$th rule.

Considering a number of n inputs in the system, the algebraic product for the t-norm used in fuzzy premise calculus and Gaussian membership functions the system output will be:

$$y_{out} = \frac{\sum_{l=1}^{M} \bar{y}_l \cdot \prod_{i=1}^{n} \mu_{Ai}^{l}(x_{in}^{i})}{\sum_{l=1}^{M} \prod_{i=1}^{n} \mu_{Ai}^{l}(x_{in}^{i})} = \frac{\sum_{l=1}^{M} \bar{y}_l \cdot \prod_{i=1}^{n} \exp\left[-\left(\frac{x_{in}^{i} - x_i^l}{\sigma_i^l}\right)\right]}{\sum_{l=1}^{M} \prod_{i=1}^{n} \exp\left[-\left(\frac{x_{in}^{i} - x_i^l}{\sigma_i^l}\right)\right]} \qquad (19)$$

x_{in}^{i} is the *i*-th input of the system, x_i^l is the *i*-th system input of the $l-$th rule
σ_i^l is the standard deviation for the $i-$th input of the $l-$th rule.

The system based on the above formula was implemented, trained and tested with different databases. In this paper, the experimental results were based on the Iris plant classification database.

3. Genetic algorithm with genes value in the real domain

3.1. Genetic algorithm in fuzzy system optimization

In our experiments, the GA was applied to optimize the system parameters. The range of definition for the input and output variables and also the number of rules for each system output, were settled by the user before the optimization begun.

The set of parameters (from (19)) that must be adjusted in order to obtain optimum performance is:

The center of gravity abscises of the output membership functions, denoted by $\bar{y}_l$. This parameter is specific for each rule so there are M parameters for the system.

The mean value of the Gaussian input membership function, denoted by x_i^l. This parameter depends on the rule number and also on the input number. There are $M \times N$ parameters that must be trained.

The standard deviation of the input Gaussian membership function (σ_i^l). Like the mean value case, there are $M \times N$ parameters.

Practically the chromosome must contain the system parameters that need to be adjusted [5,6]. All the parameters have real value and the real chromosome codification was applied. Thus each gene in the chromosome represents a fuzzy system parameter.

To realize the stage of reproduction in GA training, a fitness function that express the adaptability of each population individual must be defined. In related problems of control or modeling, the error or cost functions are minimized in order to optimize the systems. Roughly speaking, the fitness or utility or performance function is the opposite of the error or cost function. So the fitness function must be maximized to obtain a better system.

Reproduction of the offspring is based on the fitness function, and the information is passed onto the new generation by the selected individuals of the last population. Thus, the fitness function has a great impact on convergence to the best possible solution.

One of the features of the GA is the balance between the exploitation of the actual individuals and the exploration of the other new individuals that may bring necessary information. Reproduction is involved in the exploitation of the actual population individuals.

In the reproduction stage there is the possibility that at a specific moment one individual may be better adapted than the others. In this way the fitness function will be increased and thus a large number of offspring (with the same characteristics) will dominate the new population. In this way other individuals will be rejected from mating and perhaps good information might be lost. On the other hand, if the fitness function is coarse in expressing the chromosome adaptation, all the individuals will have the same percent of mating and thus the aim of optimization will never be reached. It is also possible that the convergence is slowed significantly because of the unsuitable fitness function.

We have implemented two procedures in expressing the fitness function defined by the following formulas:

$$Fitness\ 1 = e^{-\frac{\sum_{p=1}^{n} d\left(Y_p, Y(x_p)\right)}{n}} \ ; \quad Fitness\ 2 = e^{-\sum_{p=1}^{n} d\left(Y_p, Y(x_p)\right)} \tag{20}$$

here $p = 1, ..., n$ represents the number of patterns, Y_p is the output of pattern p, x_p is the p input pattern, $Y_{(x_p)}$ is the fuzzy system output for the x_p input and d is the Euclidian distance.

If the error is too large, then $Fitness\ 2$ is 0. Thus it is impossible to realize the chromosomes selection. $Fitness\ 1$ depends on the error average and therefore even if the error is large this function rarely goes to 0.

Each chromosome has a proper fitness function value. The number of offspring generated by a chromosome depends on the individual adaptability that is proportional to the fitness value.

Denote by F the sum of the performances of the all individuals in the population at a particular instant. The probability of selection is defined as the ratio between the individual fitness function and the sum of the performances:

$$p_i = \frac{f(x_i)}{F} \tag{21}$$

where x_i is the $i-$th chromosome and $f(x_i)$ is the fitness value for the individual x_i.

For each individual, the number of offspring may be computed by multiplying the selection probability by the total number of population chromosomes,

$$n_i = p_i \cdot n = \frac{f(x_i) \cdot n}{F} = \frac{f(x_i)}{\overline{F}} \tag{22}$$

where n_i is the number of generated offspring from the $i-$th chromosome, n is the number of chromosomes from the population, $\overline{F}$ is the mean value of the fitness function over the population.

In some genetic algorithms the selected chromosomes are used for recombination, while others apply the recombination, than mutation and inversion to all the individuals. So it is possible to utilize all individuals in the population to create a new generation. Finally, from the last two populations the best individuals will be selected, proportional to the fitness function (using (22)). In this way the best solution will be selected for each generation. This approach is named the elitist method. Therefore, the fitness function evolution during the training will be nondecreasing.

In order to maintain the diversity of the chromosomes, the new offspring multiplication was limited to 5% (also adjustable) of the individual number from population.

Crossover was applied in two ways. The classical crossover with two cut points was compared with the new proposed crossover.

The non-uniform mutation was applied for both crossover types. Because the chromosome bits has real value the next two formula were used for mutation. For increasing the value formula (23) was applied, till for decreasing the value formula (24) was used. The way of changing the chromosome bit that means the addition or subtraction is arbitrarily chosen. A random number was generated and compared with 0.5. If the generated value was less than 0.5 than subtraction was applied otherwise addition was performed on the chromosome bit value.

$$x' = x + (x_{\max} - x) \cdot \left[1 - r^{\left(1-\frac{t}{T}\right)}\right] \tag{23}$$

$$x' = x - (x - x_{\min}) \cdot \left[1 - r^{\left(1-\frac{t}{T}\right)}\right] \tag{24}$$

Here x is the gene value, r is the changing amplitude, t is the generation number, T is the maximum generation number of applying mutation.

One may observe that the changing on the chromosome bit depends on the generation number by the t parameter. As t increases, the changing in the chromosome bit tends to 0. This is correct, by the idea that fewer adjustments on the chromosome bits may be fruitful at the end of the optimization closed to the solution. In addition, the backpropagation method uses a decreasing learning rate.

Thus the speed of convergence is usually increased achieving better results in a less training time.

3.2. Changing the rules by the crossover operator

A new crossover was proposed, based on the idea that the rules that are already optimized must be preserved in the populations. This operator consists in randomly chosen the desired rules and all the parameters that are involved with them are consequently exchanged. Also, the variants of convex crossover, uniform and two cut points may be exploit with changing rules. The fuzzy system may be framed with a neural network [5-7]. Changing the rules means to exchange all the neurons from the system frame that are involved with the rules parameters. The probability that affect this operator is adjustable in integer numbers between 0 and 100% in our simulations.

4. Experimental results

The system was tested in plant classification field. The Iris plant database was used for testing. The data set contains three classes of 50 instances each so a total of 150 patterns. Each class refers to a type of iris plant. One class is linearly separable from the other two. The remaining two classes are not linearly separable.

The input variables represent: 1. sepal length in cm; 2. sepal width in cm; 3. petal length in cm; 4. petal width in cm.

The output is the class of belonging: - Iris Setosa, - Iris Versicolour, - Iris Virginica.

The system was trained for each crossover type a number of 300 predefined steps. To compare the results, the crossover probability was settled to 50%, 60%, 70%, 80% and 90%. For each probability, a number of 30 experiments were realized and the mean values were graphically represented. For each test the initialization of the population was the same in order to have the same start set point. After initialization the random generation number was started again but this time depending on the computer time in order to obtain different random numbers. Therefore, after the population initialization, each algorithm has its specific evolution.

For mutation the probability was settled to 5% (experimental found), r=0.7 and T=280 (it was applied on the first 280 optimization steps). Inversion was used in the first 30 steps with a probability of 1%. The percent of offspring generation was limited to 5%.

Fig. 1 represents the average value over 30 simulations of the fitness function for a number of 300 steps for each crossover probability. One may conclude that for this database crossover probability of 50% is suited for optimization. Each point on Fig. 2 represents the average value over 30 simulations of the fitness function over 300 steps for the new crossover operator for different probabilities.

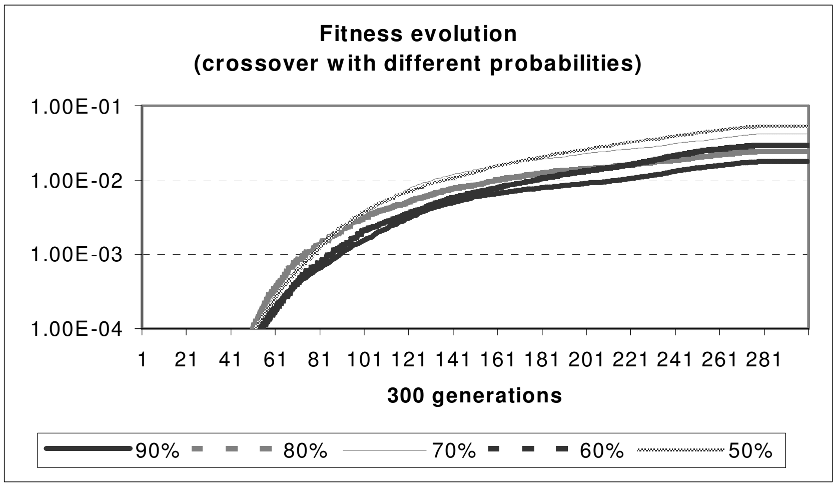

Fig. 1 – Fitness evolution for classical crossover with two cut points.

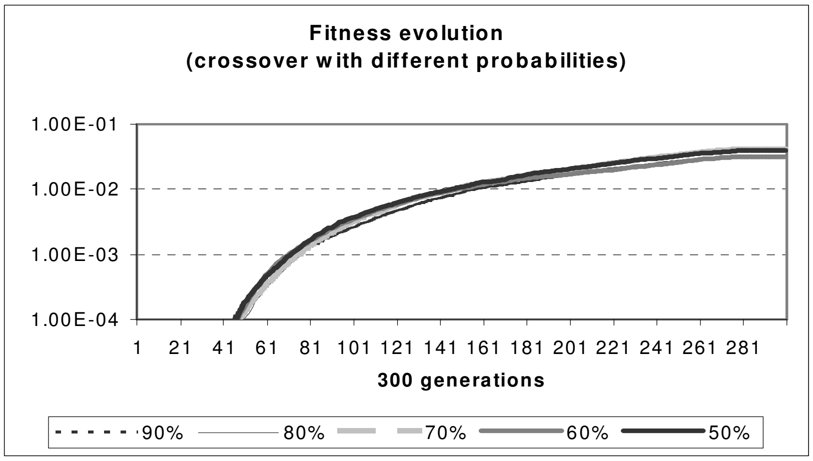

Fig. 2 – Fitness evolution for changing the rules crossover.

The differences were not large, but with the proposed crossover better solutions were obtained.

First of all the lines that define the fitness evolution for the crossover with rules changing, are grouped and it seems that the crossover probability is not so important. This is strange because usually the applying probability has a great effect on the offspring composition and therefore on the overall evolution. Perhaps this is specific only for this set of patterns.

5. Conclusions

Genetic algorithms prove to be suited for the fuzzy system optimization. It is important to mention that the application was in the classification domain but there are no restrictions in using this technique in prediction or decision. Indeed it is possible to use this system in situations were there are examples (patterns) that define the rules that must be learned.

The proposed crossover operator that changes the rules between chromosomes proves its advantage and better results were obtained. It seems that the crossover probability did not have a large effect on the fitness evolution. This may be strange and can be explained by the algorithm characteristic or the peculiarity of the training database.

References

[1] LI XIN WANG: *Adaptive Fuzzy System and Control - Design and Stability Analysis*, Prentice Hall, 1994.

[2] D. DUMITRESCU: *Algoritmi Genetici si Strategii Evolutive*, Ch. 2,3,5, Casa de editura Albastra, Cluj-Napoca, 2000.

[3] Z. MICHALEWICZ, D.B. FOEGEL: *How to Solve It - Modern Heuristics*, Springer-Verlag, Berlin, 2000.

[4] M. MITCHELL: *An Introduction to Genetic Algorithms*, MIT Press, London, 1998.

[5] F.L. LUO, R. UMBEHAUEN: *Applied Neural Networks for Signal Processing*, Cambridge University Press, 1998.

[6] J. C. PRINCIPE, N.R. EULIANO, W.C. LEFEBRE: *Neural and Adaptive Systems,* John Wiley & Sons, New York, 2000.

[7] A.J.F. VAN ROOIJ, L.C. JAIN, R.P. JOHNSON: *Neural Networks Trainings using Genetic Algorithms*, Series in Machine Perception Artificial Intelligence, Vol. **26**, Word Scientific Publising, 1996.