F.S.A.I., Vol. 11, No. 1-3, pp. 21-31, 2005

Constructing Fuzzy Rational Approximators with an Insular Distributed Genetic Algorithm

Octav Brudaru*, Octavian Buzatu**

* Dept. of Syst. Eng. & Manag., Technical University "Gh. Asachi" Iasi
** Institute of Theoretical Computer Science, Romanian Academy – Iaşi Branch

Abstract: This paper addresses the problem of approximating a fuzzy dependency known only for a finite number of input-output patterns with a fuzzy rational function whose coefficients are computed by a genetic algorithm. A distributed implementation of the genetic algorithm is presented, which improves the accuracy of the solution and decreases the computing time. The distributed genetic algorithm is based on the so called "island model" and is a client-server application, each client process being a running instance of the sequential genetic algorithm. The server has a supervising role and implements the interface with the user. The results of certain numerical experiments for comparing the distributed genetic algorithm with its sequential version are also presented.

1 Introduction

The problem considered in this paper concerns the design of a fuzzy rational function having fuzzy inputs, fuzzy outputs, and fuzzy coefficients. Starting from a given set of input-output patterns of an unknown fuzzy function, the coefficients of the numerator and nominator polynomials are adapted so that the rational function approximates the unknown fuzzy function. The error function that must be minimized during the learning process is a measure of the distance between the value of the desired outputs and the value of the calculated responses. Standard learning algorithms based on the continuity and differentiability of the error function cannot be used in this case.

In [1] a genetic algorithm for designing fuzzy polynomial approximators is presented. The results of the experimental investigation reported in [2] prove the effectiveness of the method with respect to the accuracy of the approximation and the required computing time. In [3], the polynomial model is extended in sense that a fuzzy rational function is used as approximator. This paper proposes a distributed genetic algorithm used to find the coefficients of the rational function in order to improve the accuracy of the solution and the computing time performance.

2 Problem Formulating

In this section we shortly describe the basic concepts and the tools for operating with fuzzy numbers within the proposed model.

2.1 Fuzzy Arithmetic

Let $x=((x_1,p_1),...,(x_k,p_k))$ denotes a k –sampled real fuzzy number, where x_h is a real number and $p_h \in [0,1]$ is the membership function value for $x_h, h=1,...,k$. Consider two k - sampled fuzzy numbers $x=((x_1,p_1),...,(x_k,p_k))$ and $y=((y_1,q_1),...,(y_k,q_k))$ $x_h, y_h \in R$, $p_h, q_h \in [0,1]$, $h=1,...,k$ where $x_h < x_{h+1}$, $y_h < y_{h+1}$, for $h=1,...,k-1$. Suppose that "$\otimes$" denotes an arithmetic operation, where $\otimes \in \{+,*,/\}$. Then $z = x \otimes y$, where $z=((z_1,r_1),...,(z_k,r_k))$, is given by the algorithm below [6]:

Algorithm 1: Fuzzy arithmetic

step 1. Compute $S_{ij}=x_i \otimes y_j$; $\mu_{ij}=\min(p_i,q_j)$, $i,j=1,...,k$;

step 2. Compute $a_0=\min_{i,j} S_{ij}$; $a_k=\max_{i,j} S_{ij}$; $t=(a_k-a_0)/k$; $a_h=a_0+ht$, $h=1,...,k-1$, and consider the intervals $I_1=[a_0,a_1]$, $I_h=(a_h,a_{h+1}]$, $h=1,...,k-1$;

step 3. For each $h=1,\ldots,k$ take $z_h=S_{i^*j^*}$ and $r_h=\mu_{i^*j^*}$, where $\mu_{i^*j^*}=\max\{\mu_{ij} / S_{ij} \in I_h\}$. If there is h so that $\{\mu_{ij} / S_{ij} \in I_h\}=\varnothing$ then $z_h=a_h$ and $r_h=0$.

2.2 Defuzzification

Defuzzification is the process to select a representative real number for a fuzzy number. If $x=((x_1,p_1),...,(x_k,p_k))$ is a fuzzy number, then let

$$d(x)=\frac{\sum_{i=1}^{k} x_i p_i}{\sum_{i=1}^{k} p_i}$$

be the defuzzified value returned by the *centre of gravity* operator.

2.3 Fuzzification

A fuzzification operator has the effect of transforming crisp data into fuzzy numbers. It constructs a fuzzy number $F(b)$ using random perturbation of a real value b, by generating a number of samples of the random variable following the Gauss distribution N(b,1).

3 Mathematical Model

Consider two sets A and B of fuzzy numbers and a fuzzy function $f: A \to B$ that is known for a given finite set of input-output examples $y(i)=f(x(i))$,

$x(i) \in A$, $y(i) \in B$, $i = 1,...,S$. The purpose is to approximate the function f using a fuzzy rational function $R(x) = P(x)/Q(x)$, where $P(x) = a(0)x^m + a(1)x^{m-1} + ... + a(m)$ and $Q(x) = b(0)x^n + b(1)x^{n-1} + ... + b(n)$ are two fuzzy polynomials of degree m and n respectively. The coefficients $a(0),...,a(m)$, $b(0),...,b(n)$ and x are k-sampled real fuzzy numbers. These polynomials are evaluated using the fuzzy arithmetic presented above. Consider the approximating error defined by

$$E(a(0),...,a(m);b(0),...,b(n)) = \frac{1}{S}\sum_{i=1}^{S} \left\| R(x(i)) - (y(i)) \right\| \qquad (1)$$

where $\|\cdot\|$ is the Euclidian norm defined on R^{2k}. The problem is to find the fuzzy numbers $a^*(0),...,a^*(m)$ and $b^*(0),...,b^*(n)$ so that (1) is minimized. This requires the finding of suitable values for $2k(m+n+2)$ parameters. Due to the fuzzy arithmetic, the continuity and the differentiability of the error function are not guaranteed. Therefore, the standard learning algorithms based on the continuity and the differentiability of the error function cannot be used in this case. This paper proposes a distributed genetic algorithm able to find suitable values for the coefficients of the fuzzy rational approximator.

4 The Sequential Genetic Algorithm

The main components of the basic genetic algorithm (GA) [7] for finding fuzzy rational approximators of the fuzzy data are presented below.

4.1 Solution's representation

Consider a fuzzy number $x = ((x_1, p_1),...,(x_k, p_k))$. The representation of the membership function value for x_h, $h = 1,...,k$ is simply done with a sequence of 8 binary digits that leads to an integer $w_h \in \{0,...,255\}$. Taking $H = 1/255$, leads to $p_h = Hw_{ih}$ and $p_h \in \{0,\ 1/255,...,254/255,\ 1\}$, and this is enough for the practical applications. The representation of x_h exploits the fuzzy data acquisition and arithmetic. Suppose that $x_1,...,x_k$ belong to k intervals delimited by $k+1$ equally spaced nodes, i.e. $\theta \le x_1 \le \theta + 2\varepsilon \le x_2 \le \theta + 4\varepsilon \le ... \le x_k \le \theta + 2k\varepsilon$ for certain values of the centre θ and of the radius $\varepsilon > 0$. Therefore, for a given h, x_h belongs to the interval $[\theta + (2h-1)\varepsilon - \varepsilon,\ \theta + (2h-1)\varepsilon + \varepsilon]$.

The first locating of each component is done by its rank h. A more refined locating of x_h is made by taking into account that it exists $\gamma_h \in [-1,1]$ so that $x_h = \theta + (2h-1)\varepsilon + \gamma_h\varepsilon$. Denote by $c_1...c_s$ a sequence of binary digits that codify

the direction and distance of the movement induced by a binary search technique for locating the γ -value within the interval [-1,1].

The codification of γ is done with an algorithm that uses γ and outputs the sequence $c_1...c_s$ of s bits. Another algorithm uses the binary sequence $c_1...c_s$ and returns $\gamma \in (-1,1)$. Each x_h can be represented by its own binary sequence $c_1...c_s$. The values of θ and ε are the same for all $x_1,...,x_k$. The fuzzy number x is represented as the list $L(x) = \left((\theta,\varepsilon),(c_h,w_h),h=1,...,k\right)$ which requires 2 floating-point numbers (for θ and ε), k binary sequences of length 8 and k sequences of s bits, instead of $2k$ floating point representations, where $s \in \{8,16,24,32\}$. Consequently, a chromosome p is the concatenation of the lists corresponding to the coefficients of the polynomials P and Q, namely $p = \left(\left(L\left(a(i)\right), i=0,...,m\right),\left(L\left(b(j)\right), j=0,...,n\right)\right)$.

This fully defines a fuzzy rational function $R(x) = P(x)/Q(x)$. The using of a hybrid representation is motivated by the necessity to cover the search space for the coefficients with different windows. The left margin and the width of the windows can be tuned by θ and ε values, respectively. The search refinement within each window is made by the c -binary sequences. The reason for using this representation is argued in [3].

4.2 Initial Population

The best found technique for generating the initial individuals is to construct the numerator by perturbing the coefficients of the least-squares polynomial constructed for the defuzzified data. The nominator is set by fuzzifying the constant 1 with the fuzzification operator. This technique significantly improves the accuracy of the solutions and the speed of the convergence.

4.3 Mutation Operators

Each individual supports the mutation with the same probability π_m. The proposed GA uses two types of mutation. The first type is called *bit level* mutation and it is applied for both membership values and realizations. The probability $pm(nb, j, t)$ to invert the bit j in a sequence of nb bits at age t ensures that at the beginning of the evolution process, the most representative part of the binary sequences is changed with a greater probability than the ending bits. During the end of the search process, the roles of these parts are interchanged in order to implement a fine-tuning search. In this paper was used the function defined by $pm(nb, j, t) = 2 * f(j) * g(t) - f(j) - g(t) + 1$ where $f(j) = j/nb$, $j = 1,...,nb$ and $g(t) = \ln t/(\ln t + 1)$, $t \geq 1$. The second type of mutation used is the non-uniform mutation [7] which acts on the floating-point part of the representation.

4.4 Crossover Operators

The parents supporting the crossover are selected from the best 40% of the whole population. The individuals in this top form pairs and such a couple is selected with the probability π_c. The first crossover operator acts on the binary sequences from the representations of the two parents. These sequences are fragmented by the same random cutting position. In order to construct its corresponding sequences, an offspring gets the prefix from a parent and the remaining positions from the other.

The crossover needs also to operate on the θ and ε values associated with the representation of each fuzzy coefficient. Consider a pair (θ,ε) in the parent p and the corresponding pair (θ',ε') in the parent p'. The corresponding pairs $(\theta^{(i)},\varepsilon^{(i)})$ in the offspring i, $i=1,2$ are convex combinations of (θ,ε) and (θ',ε').

4.5 Population Management

A fixed size population was considered. During each stage, mutation and crossover lead to new individuals. The individuals are scored with the value of the fitness function given by the objective function (1) that is to be minimized. In order to avoid the cases when a high difference exists between different terms in (1), even if their average is small, it is possible to use $\max_{i=1,\dots,S}\|R(x(i))-y(i)\|$ as fitness function instead of (1). The selection of the individuals for the next generation is done by a tournament technique.

5 The Distributed Implementation

The design of the distributed version of the above basic GA is based on the so-called *island model* [5]. The execution of the distributed genetic algorithm consists in the parallel run of N processes $P_i, i=1,\dots,N$. Each process P_i consists in a sequential run of the basic GA. All processes work with the same values of the control parameters.

5.1 The Migration Operator

Each process contains a migration event generator acting with a given migration probability p_{mig}. The migration operator is used for inter-process communication. In each process, the migration event is produced after the applying of the genetic operators to the current population and before the establishing of the chromosomes which are forming the next generation. Whenever such an event is produced in a process P_i, a second process P_j ($i \neq j$) is selected and P_i sends to P_j a prescribed number, called *migration rate* m_r, of its own individuals. The selection of the

migrating chromosomes can be *elitist* (when the best chromosomes are chosen to be sent) or *randomly*.

In the second case the *emigrants* are randomly extracted from the first 50% chromosomes from population which have the best fitness values. After the sending of emigrants, process P_i continues its own work. The receiving process P_j continues its activity and at the end of its current stage, before deciding the structure of its next generation, verifies the presence of the *immigrants* and includes them into its own population. The added individuals compete with the existing population for the next generation. The process P_i selects the destination process randomly. Each process can send/receive chromosomes to/from any another process.

5.2 The Stopping Condition

The convergence detection is implemented by a fan-in mechanism hosted by a master process P_0. At the end of the current evolution stage, each working process tests if the average fitness value is stabilized along a prescribed number of successive stages. If this happened, the process is terminated and the best fitness value from the final population together with its corresponding chromosome are sent to master P_0.

The working process is also stopped when the number of generations reaches a predefined limit, even if the convergence was not detected. After all client processes stopped, P_0 outputs the best solution and the whole activity is stopped. Such stop conditions become valid due to the activity of the migration operator that tends to acts as performance equalizer. Due to the exchange of information, all processes get closer but not equal solutions in terms of the fitness function values.

6 Performance of the Distributed GA

The implementation of the distributed genetic algorithm was done on a cluster structure based on the NT-MPICH environment implementing MPI-1 standard. For this implementation, we used a cluster of four IBM PC computers, connected by a local network Ethernet 100Base TX. All computers have the same hardware configuration: Athlon XP 2100+ with 512 MB DDRAM, Ethernet 100 MBs network cards. The operating system used was Windows XP.

In the following group of experiments, we compare the above distributed GA with the sequential version analyzed in [3]. It was considered a known witness rational function, $R(x) = P(x)/Q(x)$ with the polynomials P and Q (having the degree 3) presented in Table 1. The size of the fuzzy coefficients was 3. The gray locations contain the realizations of the fuzzy coefficients and the white locations contain the membership function values.

Table 1

The witness rational function

Pol.	Coef of X^3			Coef of X^2			Coef of X^1			Coef of X^0		
P	8.9	9	9.1	5.9	6	6.1	6.9	7	7.1	9.9	10	10.1
	0.1	0.9	0.2	0.15	0.9	0.1	0.1	0.9	0.15	0.1	0.9	0.15
Q	-0.9	-1	-1.1	2.9	3	3.1	0.9	1	1.1	-1.9	-2	2.1
	0.1	0.9	0.2	0.15	0.9	0.1	0.1	0.9	0.15	0.1	0.9	0.15

Legend	
	realization value
	membership value

Starting from this witness rational function, it was produced a set of 30 input-output pairs $(x(i), y(i))$ with $y(i) = R(x(i))$, $i = 1,...,30$. These dependencies are shown in Table 2.

The ability of the distributed GA to produce solutions which are similar to the witness function was estimated. We chose 5 sets of control parameters for the distributed genetic algorithm and done 5 experiments. The probabilities of the mutation and crossover operators were chosen from the ones which produced the best solution in the sequential version [3] ($\pi_m = \pi_c = 0.4$). We used a *pop_size* of 30 individuals, because for larger populations the running time would greatly increase. The values of the control parameters for the distributed algorithm are presented in Table 3.

Each experiment aggregates the results of 10 successive runs. Further, for the current experiment, $bestval_i[j]$ denotes the best fitness value of the final population in process P_i, at the j-th run, $i = 1,...,4$ and $j = 1,...,10$. GEN_j and $TIME_j$ denotes the number of generations, respectively the execution time (in seconds) necessary to obtain the value $\min_{i=1,...,4} bestval_i[j]$.

Table 2

The dependencies generated using the witness rational function

i	x(i)			y(i)			i	x(i)			y(i)		
1	0.9	1	1.1	6.27	23.12	32	16	5.9	6	6.1	-27.05	-21.27	-17.61
	0.1	0.9	0.1	0.15	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2
2	1.23	1.33	1.43	8	17.2	22.37	17	6.23	6.33	6.43	-24.91	-19.96	-16.69
	0.1	0.9	0.1	0.15	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2
3	1.56	1.66	1.76	10.64	14.34	23.66	18	6.56	6.66	6.76	-23.23	-18.91	-15.92
	0.1	0.9	0.1	0.15	0.1	0.9		0.1	0.9	0.1	0.1	0.9	0.2
4	1.9	2	2.1	14.81	19.23	30	19	6.9	7	7.1	-21.86	-18.01	-15.27
	0.1	0.9	0.1	0.15	0.1	0.9		0.1	0.9	0.1	0.1	0.9	0.2
5	2.23	2.33	2.43	25.39	35.36	43.54	20	7.23	7.33	7.43	-20.76	-17.28	-14.73
	0.1	0.9	0.1	0.15	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2

6	2.56	2.66	2.76	20.47	45.51	78.44	21	7.56	7.66	7.76	-19.85	-16.66	-14.26
	0.1	0.9	0.1	0.1	0.15	0.9		0.1	0.9	0.1	0.1	0.9	0.2
7	2.9	3	3.1	136.67	238.05	328	22	7.9	8	8.1	-19.06	-16.11	-13.85
	0.1	0.9	0.1	0.15	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2
8	3.23	3.33	3.43	-185.54	100.87	244.08	23	8.23	8.33	8.43	-18.4	-15.64	-13.5
	0.1	0.9	0.1	0.9	0	0.15		0.1	0.9	0.1	0.1	0.9	0.2
9	3.56	3.66	3.76	-200.41	-101.06	-77.6	24	8.56	8.66	8.76	-17.83	-15.24	-13.19
	0.1	0.9	0.1	0.1	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2
10	3.9	4	4.1	-85.35	-52.25	-50.71	25	8.9	9	9.1	-17.34	-14.86	-12.9
	0.1	0.9	0.1	0.1	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2
11	4.23	4.33	4.43	-69.11	-42.63	-39.08	26	9.23	9.33	9.43	-16.9	-14.54	-12.65
	0.1	0.9	0.1	0.1	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2
12	4.56	4.66	4.76	-50	-33.36	-32.46	27	9.56	9.66	9.76	-16.51	-14.26	-12.43
	0.1	0.9	0.1	0.1	0	0.9		0.1	0.9	0.1	0.1	0.9	0.2
13	4.9	5	5.1	-39.92	-28.09	-22.18	28	9.9	10	10.1	-16.15	-13.99	-12.22
	0.1	0.9	0.1	0.1	0.9	0.2		0.1	0.9	0.1	0.1	0.9	0.2
14	5.23	5.33	5.43	-33.99	-25.14	-20.26	29	10.23	10.33	10.43	-15.83	-13.75	-12.04
	0.1	0.9	0.1	0.1	0.9	0.2		0.1	0.9	0.1	0.1	0.9	0.2
15	5.56	5.66	5.76	-30	-22.98	-18.8	30	10.56	10.66	10.76	-15.54	-13.54	-11.87
	0.1	0.9	0.1	0.1	0.9	0.2		0.1	0.9	0.1	0.1	0.9	0.2

Table 3

The values of the control parameters for the distributed algorithm

Exp. no.	p_{mig}	m_r	emigrants selection type
1	0.1	2	elitist
2	0.4	2	elitist
3	0.1	1	elitist
4	0.3	1	elitist
5	0.5	5	random

For the whole distributed algorithm, we computed the following performance indicators:

- the average of the best fitness values

$$M_{pop} = \frac{\sum_{j=1}^{10} \min_{i=1,\ldots,4} bestval_i[j]}{10}$$

$$GENM_{pop} = \frac{\sum_{j=1}^{10} GEN_j}{10} \qquad (2)$$

$$TIMEM_{pop} = \frac{\sum_{j=1}^{10} TIME_j}{10}$$

- the best of the best fitness values (for all 10 executions) corresponding to the best chromosome found after each execution:

$$B_{pop} = \min_{j=1,...,10} \min_{i=1,...,4} bestval_i[j]$$
$$GENB_{pop} \text{ - the number of generation necessary to obtain value } B_{pop} \quad (3)$$
$$TIMEB_{pop} \text{ - the execution time (in seconds) necessary to obtain } B_{pop}$$

All these values are shown in Table 4. Due to the use of a tournament selection, the $GENM_{pop}$ and $GENB_{pop}$ values are always 2000. The stopping condition of each client process is satisfied only when the number of generations reaches a predefined upper limit (in this case 2000), because the tournament selection doesn't allow the stagnation of the average fitness around certain values.

The quality of the solutions given by the distributed algorithm is more than 10 times better compared to the sequential version [3], regarding both medium and best fitness values. Thus the distributed algorithm produced a major improvement in both M_{pop} and B_{pop} values. However, this improvement requires a higher computing time, due to the inter-process communication costs. The distributed algorithm simultaneously explores more subspaces of the solution space, dealing with many subpopulations at a time.

This feature improves both the average and the best fitness values. Moreover, whenever a good individual is found in one worker process, it has a high probability to be transmitted to the others worker processes, so the good individuals are always promoted.

Table 4

The values of performance indicators for the 5 experiments

Exp. no.	M_{pop}	$TIMEM_{pop}$	B_{pop}	$TIMEB_{pop}$
1	5.15	1225	3.19	1199
2	5.57	1215	3.88	1267
3	6.54	1160	4.80	1144
4	5.90	1163	3.86	1140
5	4.65	1282	2.66	1287

The approximation accuracy for the best fuzzy rational function obtained in each of the 5 experiments described above is illustrated in Fig. 1. The values $(d(x(i)), d(y(i))), i = 1,...,30$ and $(d(x(i)), d(R_j(x(i)))), i = 1,...,30, j = 1,...,5$ are represented, where $R_j, j = 1,...,5$ are the fuzzy rational functions corresponding to the B_{pop} values presented in Table 4 and d is the centre of gravity defuzzification operator.

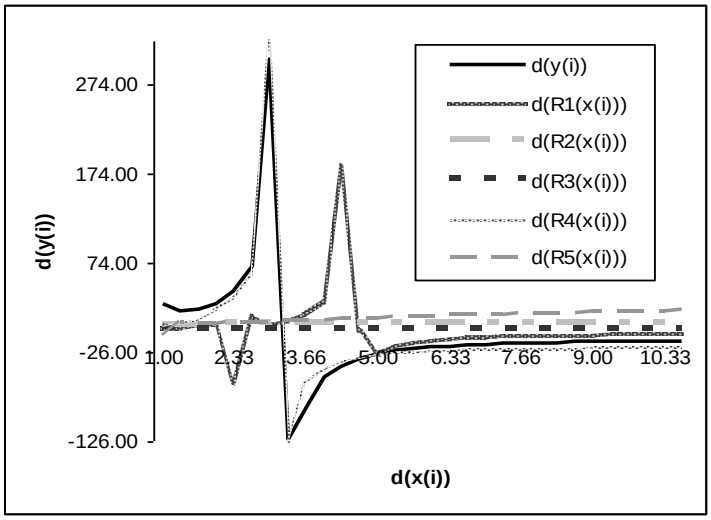

Fig. 1 – The approximation accuracy for the best fuzzy rational functions obtained in the 5 experiments presented above

The variation of the best and medium fitness values for a version of the distributed GA involving four processes is shown in Fig. 2.

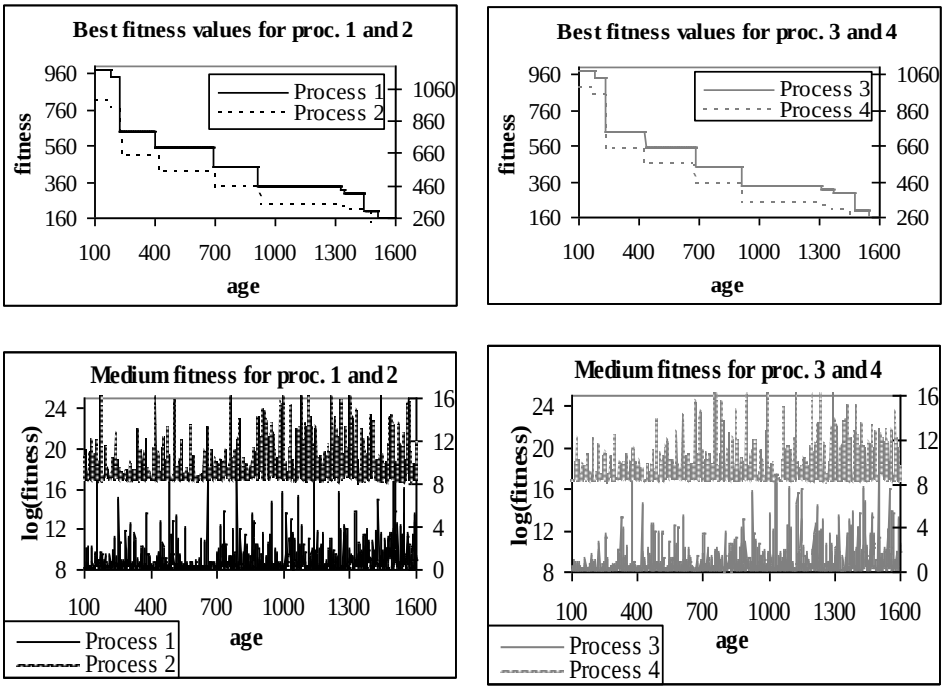

Fig. 2 – Typical variation of the best and medium fitness values (four processes)

7 Conclusions

We proposed a distributed genetic algorithm for computing fuzzy rational approximators. The numerical investigation shows that it produces good quality solutions and requires a reasonable computing time. It is proved that the distributed GA produces much better solutions that its sequential version.

Another direction of research concerns the case of large sets of training data in which the input data is organized in explicit or implicit clusters [8]. In such a case, a set of good cluster oriented polynomial or rational approximators could be combined for producing a global fuzzy approximator.

References

[1] O. BRUDARU, S. GRADINARU: *Evolutionary Approach for Constructing Fuzzy Polynomials Approximating Fuzzy Data*, VII Congress of SIGEF "New Logic for the New Economy", 2001

[2] O. BRUDARU, O. BUZATU: *Genetic Algorithm for Finding Polynomial Approximators of Fuzzy Dependencies*, Proceedings of the 4th International Conference on Recent Advances in Soft Computing – RASC2002, pp. 176 – 181, 2002

[3] O. BRUDARU, O. BUZATU: *Genetic algorithm for finding fuzzy rational approximators for fuzzy data*, National Symposium on Intelligent Systems and Applications – SSIA'2003, 2003

[4] O. BRUDARU, V. ALAIBA, O. BUZATU: *Distributed Genetic Algorithm for Finding Fuzzy Polynomial Approximators*, National Symposium on Intelligent Systems and Applications - SSIA'2003, 2003

[5] E. CANTU-PAZ: *Implementing Fast and Flexible Parallel Genetic Algorithms*, Practical Handbook of Genetic Algorithms, vol. **3**, ed. CRC Press, 1999

[6] T. HIROTA: *A Digital Representation of Fuzzy Numbers and its Application*, Proceedings of IIZUKA'90, pp. 527-529, 1990

[7] Z. MICHALEWICZ: *Genetic Algorithms + Data Structures = Evolution Program*, 2nd ed., ed. Springer Verlag, 1994

[8] M. SATO, Y. SATO, L.C. JAIN: *Fuzzy Clustering Models and Application*, ed. Physica Verlag, 1997