Time Series Models Discovery with Similarity-Based Neuro-Fuzzy Networks and Genetic Algorithms: A Parallel Implemention.

Valdés, Julio; Mateescu, G.

Time series model mining with similarity-based neuro-fuzzy networks and genetic algorithms: a parallel implementation. *

Valdés, J., and Mateescu, G.

October 2002

* published in: Proceedings of the RSCTC'02, Special Session on Distributed and Collaborative Data Mining, Pennsylvania. October 2002. NRC 44933.

Time Series Model Mining with Similarity-Based Neuro-Fuzzy Networks and Genetic Algorithms: A Parallel Implementation

Julio J. Valdés[1] and Gabriel Mateescu[2]

[1] National Research Council of Canada
Institute for Information Technology
1200 Montreal Road, Ottawa ON K1A 0R6, Canada
julio.valdes@nrc.ca
[2] National Research Council of Canada
Information Management Services Branch
100 Sussex Drive, Ottawa ON K1A 0R6, Canada
gabriel.mateescu@nrc.ca

Abstract. This paper presents a parallel implementation of a hybrid data mining technique for multivariate heterogeneous time varying processes based on a combination of neuro-fuzzy techniques and genetic algorithms. The purpose is to discover patterns of dependency in general multivariate time-varying systems, and to construct a suitable representation for the function expressing those dependencies. The patterns of dependency are represented by multivariate, non-linear, autoregressive models. Given a set of time series, the models relate future values of one target series with past values of all such series, including itself. The model space is explored with a genetic algorithm, whereas the functional approximation is constructed with a similarity based neuro-fuzzy heterogeneous network. This approach allows rapid prototyping of interesting interdependencies, especially in poorly known complex multivariate processes. This method contains a high degree of parallelism at different levels of granularity, which can be exploited when designing distributed implementations, such as workcrew computation in a master-slave paradigm. In the present paper, a first implementation at the highest granularity level is presented. The implementation was tested for performance and portability in different homogeneous and heterogeneous Beowulf clusters with satisfactory results. An application example with a known time series problem is presented.

1 Introduction

Multivariate time-varying processes are common in a wide variety of important domains like medicine, economics, industry, communications, environmental sciences, etc. Developments in sensor and communication technology enable the simultaneous monitoring and recording of large sets of variables quickly, therefore generating large sets of data. Processes of this kind are usually described by sets of variables, sometimes of heterogeneous nature. Some are numeric, others are non-numeric, for example, describing discrete state transitions. In real world situations, it is practically impossible to record all variables at all time frames, which leads to incomplete information. In practice, the degree of accuracy associated with the observed variables is irregular, resulting in data sets with different kinds and degrees of imprecision. All of these problems severely limit the applicability of most classical methods. Many techniques have been developed for time series prediction from a variety of conceptual approaches ([3], [6]), but the problem of finding models of internal dependencies has received much less attention. However, in real world multivariate processes, the patterns of internal dependencies are usually unknown and their discovery is crucial in order to understand and predict them. In the present approach, the space of possible models of a given kind is explored with genetic algorithms and their quality evaluated by constructing a similarity-based neuro-fuzzy network representing a functional approximation for a prediction operator. This approach to model mining is compute-intensive, but it is well suited for supercomputers and distributed computing systems. In the parallel implementation of this soft-computing approach to model discovery, several hierarchical levels can be identified, all involving intrinsically parallel operations. Therefore, a variety of implementations exploiting different degrees of granularity in the evolutionary algorithm and in the neuro-fuzzy network is possible. Here, following a parsimonious principle, the highest level is chosen for a first parallel implementation: that of population evaluation within a genetic algorithm.

2 Problem Formulation

The pattern of mutual dependencies is an essential element of this methodology. The purpose is to explore multivariate time series data for plausible dependency models expressing the relationship between future values of a previously selected series (the target), with past values of itself and other time series. Some of the variables composing the process may be numeric (ratio or interval scales), and some qualitative (ordinal or nominal scales). Also, they might contain missing values. Many different families of functional models describing the dependency of future values of a target series on the previous values can be considered, and the classical linear models AR, MA, ARMA and ARIMA [3], have been extensively studied. The choice of the functional family will influence the overall result. The methodology proposed here does not require a particular model. Because the generalized nonlinear AR model expressed by relation (1) is a simple model which makes the presentation easier to follow, we use this basic model:

$$S_T(t) = \mathrm{F}\begin{pmatrix} S_1(t-\tau_{1,1}), S_1(t-\tau_{1,2}), \cdots, S_1(t-\tau_{1,p_1}), \\ S_2(t-\tau_{2,1}), S_2(t-\tau_{2,2}), \cdots, S_2(t-\tau_{2,p_2}), \\ \cdots \\ S_n(t-\tau_{n,1}), S_n(t-\tau_{n,2}), \cdots, S_n(t-\tau_{n,p_n}) \end{pmatrix} \tag{1}$$

where $S_T(t)$ is the target signal at time t, S_i is the i-th time series, n is the total number of signals, p_i is the number of time lag terms from signal i influencing $S_T(t)$, $\tau_{i,k}$ is the k-th lag term corresponding to signal i ($k \in [1, p_i]$), and F is the unknown function describing the process.

The goal is the simultaneous determination of: *i)* the number of required lags for each series, *ii)* the sets of particular lags within each series carrying dependency information, and *iii)* the prediction function, in some optimal sense. The size of the space of possible models is immense (even for only a few series and a limited number of time lags), and the lack of assumptions about the prediction function makes the set of candidates unlimited. A natural requirement on function F is the minimization of a suitable prediction error and the idea is to find a reasonably small subset with the best models in the above mentioned sense.

2.1 A SOFT COMPUTING MODEL MINING STRATEGY

A soft computing approach to the model mining problem can be: *i)* exploration of the model space with evolutionary algorithms, and *ii)* representation of the unknown function with a neural network (or a fuzzy system). The use of a neural network allows a flexible, robust and accurate predictor function approximator operator. Feed-forward networks and radial basis functions are typical choices. However, the use of these classical network paradigms might be difficult or even prohibitive, since for each candidate model in the search process, a network of the corresponding type has to be constructed and trained. Issues like chosing the number of neurons in the hidden layer, mixing of numeric and non-numeric information (discussed above), and working with imprecise values add even more complexity. Moreover, in general, these networks require long and unpredictable training times. The proposed method uses a heterogeneous neuron model [9], [10]. It considers a neuron as a general mapping from a heterogeneous multidimensional space composed by cartesian products of the so called extended sets, to another heterogeneous space. These are formed by the union of real, ordinal, nominal, fuzzy sets, or others (e.g. graphs), with the missing value (e.g. for the reals $\hat{\mathcal{R}} = \mathcal{R} \cup \{\chi\}$, where χ is the missing value). Their cartesian product forms the heterogeneous space, which in the present case, is given by $\hat{\mathcal{H}}^n = \hat{\mathcal{R}}^{n_r} \times \hat{\mathcal{O}}^{n_o} \times \hat{\mathcal{N}}^{n_n} \times \hat{\mathcal{F}}^{n_f}$. In the *h-neuron*, the inputs, and the weights, are elements of the n-dimensional heterogeneous input space. Among the many kinds of possible mappings, the one using a similarity function [4] as the aggregation function and the identity mapping as the activation function is used here. Its image is the real interval [0,1] and gives the degree of similarity between the input pattern and neuron weights. See Fig-2.1 (left).

The h-neuron can be used in conjunction with the classical (dot product as aggregation and sigmoid or hyperbolic tangent as activation), forming hybrid network architectures. They have general function approximation properties [1], and are trained with evolutionary algorithms in the case of heterogeneous inputs and missing values due to lack of continuity in the variable's space. The hybrid network used here has a hidden layer of h-neurons and an output layer of classical neurons. In the special case of predicting a single real-valued target time series, the architecture is shown in Fig-2.1 (right).

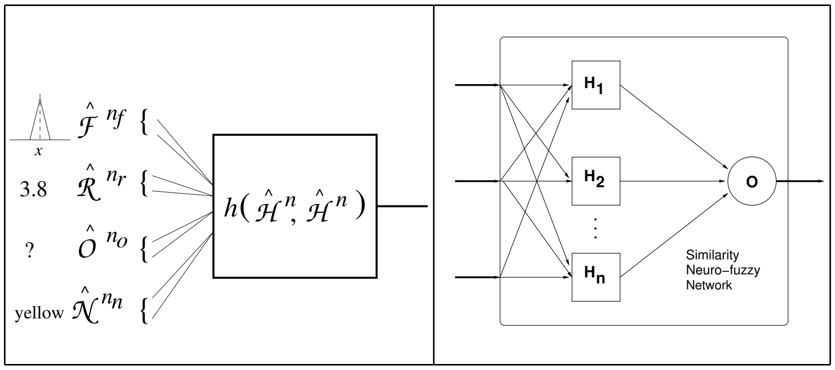
x
3.8
?
yellow
$\hat{\mathcal{F}}^{n_f}$
$\hat{\mathcal{R}}^{n_r}$
$\hat{\mathcal{O}}^{n_o}$
$\hat{\mathcal{N}}^{n_n}$
$h(\hat{\mathcal{H}}^n, \hat{\mathcal{H}}^n)$
H1
H2
Hn
O
Similarity Neuro–fuzzy Network

Fig. 1. Left: A heterogeneous neuron. Right: A hybrid neuro-fuzzy network.

This network works like a *k-best* interpolator algorithm: Each neuron in the hidden layer computes its similarity with the input vector and the k-best responses are retained (k is a pre-set number of h-neurons to select). Using as activation a linear function with a single coefficient equal to the inverse of the sum of the k-similarities coming from the hidden layer, the output is given by (2).

$$output = (1/\Theta) \sum_{i \in \mathcal{K}} h_i W_i, \qquad \Theta = \sum_{i \in \mathcal{K}} h_i \tag{2}$$

where $\mathcal{K}$ is the set of k-best h-neurons of the hidden layer and h_i is the similarity value of the i-best h-neuron w.r.t the input vector. These similarities represent the fuzzy memberships of the input vector to the set classes defined by the neurons in the hidden layer. Thus, (2) represents a fuzzy estimate for the predicted value. Assuming that a similarity function $\mathcal{S}$ has been chosen and that the target is a single time series, this *case-based* neuro-fuzzy network is built and trained as follows: Define a similarity threshold $T \in [0, 1]$ and extract the subset $\mathcal{L}$ of the set of input patterns Ω ($\mathcal{L} \subseteq \Omega$) such that for every input pattern $x \in \Omega$, there exist a $l \in \mathcal{L}$ such that $\mathcal{S}(x, l) \geq T$. Several algorithms for extracting subsets with this property can be constructed in a single cycle through the input pattern set (note that if $T = 1$, the hidden layer becomes the whole training set). The hidden layer is constructed by using the elements of $\mathcal{L}$ as h-neurons. While the output layer is built by using the corresponding target outputs as the weights of the neuron(s). This training procedure is very fast and allows construction and testing of many hybrid neuro-fuzzy networks in a short time. Different sets of individual lags selected from each time series will define different training sets, and therefore, different hybrid neuro-fuzzy networks. This one-to-one correspondence between dependency models and neuro-fuzzy networks, makes the search in the model space equivalent to the search in the space of networks. Thus, given a model describing the dependencies and a set of time series, a hybrid network can be constructed according to the outlined procedure, and tested for its prediction error on a segment of the target series not used for training (building) the network. The Root Mean Squared (RMS) error is a typical goodness of fit measure and is the one used here. For each model the quality indicator is given by the prediction error on the test set of its equivalent similarity-based neuro-fuzzy network (the prediction function). The search for *optimal* models can be made with an evolutionary algorithm minimizing the prediction error measure. Genetic Algorithms and Evolution Strategies are typical for this task and many problem representations are possible. Genetic algorithms were used with a simple model coding given by binary chromosomes of length equal to the sum of the maximal number of lags considered for each of the time series (the *time window depth*). Within each chromosome segment corresponding to a given series, the non-zero values will indicate which time lags should be included in the model, as shown in Fig-2.1.

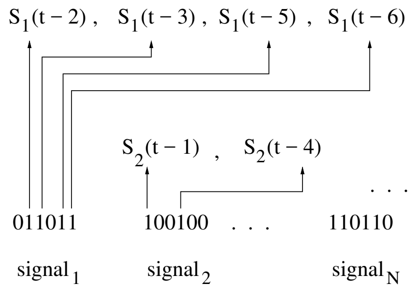

Fig. 2. Chromosome decodification.

The system architecture is illustared in Fig-2.1. The series are divided into training and test sets. A model is obtained from a binary chromosome by decodification. With the model and the series, a hybrid neuro-fuzzy network is built and trained, representing a prediction function. It is applied to the test set and a prediction error is obtained, which is used by the genetic algorithm internal operators. Models with smaller errors are the fittest.

At the end of the evolutionary process, the best model(s) are obtained and if the test errors are acceptable, they represent meaningful dependencies within the multivariate process. Evolutionary algorithms can't guarantee the global optimum, thus, the models found can be seen only as plausible descriptors of important relationships present in the data set. Other neural networks based on the same model may have better approximation capabilities. In this sense, the proposed scheme should be seen as giving a *coarse* prediction operator. The advantage is the speed with which hundreds of thousands of models can be explored and tested (not possible with other neural networks). Once the best models are found, more powerful function approximators can be obtained with other types of neural networks, fuzzy systems, or other techniques. This method depends

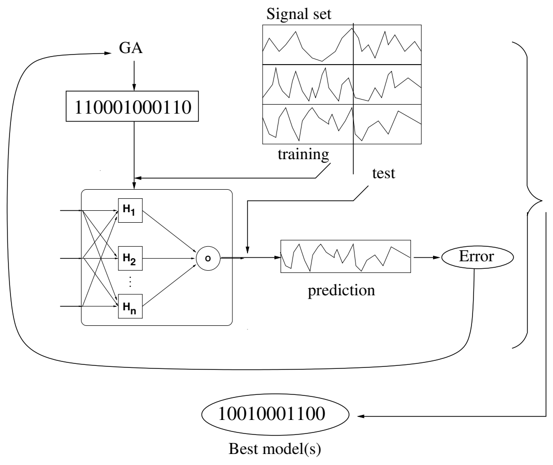

Fig. 3. System architecture.

on different parameters which must be defined in advance (the similarity function, the similarity threshold, etc). In order to account for an optimal selection of these parameters, meta-evolutionary paradigms like the one outlined in Fig-4 are relevant. The outermost genetic structure explores the space of problem parameters.

3 Parallel Implementation

The hybrid nature of the soft-computing method described in the previous section allows several different approaches for constructing parallel and distributed computer implementations. Hierarchically, several levels can be distinguished: the evolutionary algorithm (the genetic algorithm in this case) operates on a least squared type functional containing a neuro-fuzzy network. In turn, inside the network there are neuron layers, which themselves involve the work of individual neurons (h-neurons and classical). Finally, inside each neuron, a similarity function is evaluated on the input and the weight vector in a componentwise operation (e.g. a correlation, a distance metric, or other function). All of these are typical cases of the workcrew computation paradigm at different levels of *granularity*. Clearly, a completely parallel algorithm could be ultimately constructed by parallelizing all levels traversing the entire hierarchy. This approach however, will impose a big amount of communication overhead between the physical computation elements, especially in the case of Beowulf clusters (the most affordable supercomputer platform). Following the principle of parsimony, the implementation was done at the highest granularity level in the genetic algorithm, namely at the population evaluation level (clearly, other operations can be parallelized within the other steps of the genetic algorithm like selection, etc.). The classical

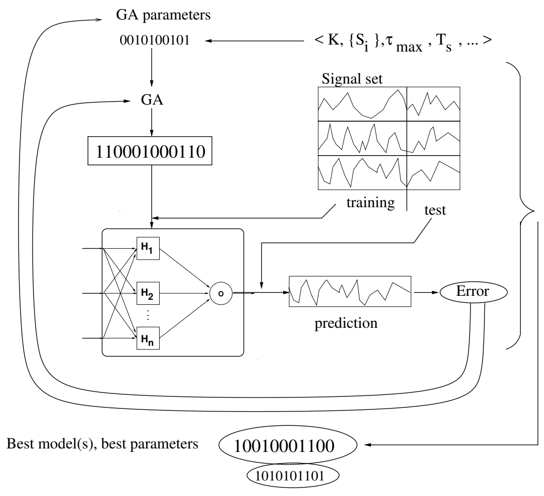

Fig. 4. Meta-genetic algorithm architecture. The outermost genetic structure explores the space of problem parameters and the inner process finds the best model(s) for a given set of them.

type of genetic algorithm chosen (with binary chromosomes, simple crossover and mutation, etc) makes the population initialization and evaluation steps a natural first choice. The evaluation of a population involves the parallel evaluation of all its individuals (models) which can be done in parallel. At this level, there is a high degree of parallelism. A master-slave computation structure is employed, where the master initializes and controls the overall process, collecting the partial results, and the slaves construct the neuro-fuzzy network based on the decoded chromosome, and evaluate it on the time series. In our implementation, the workload is managed dynamically, so that the load is well balanced in a heterogeneous cluster environment. The communication overhead was reduced by replicating the data set on all the machines in the cluster. Thus, the master program is relieved from sending a copy of the entire data set to each slave program each time a model has to be evaluated. Messages sent to the slaves are binary chromosomes, while messages received back by the master contain only a single floating point number with the RMS error associated with the chromosome (model).

The Parallel Virtual Machine PVM [5] message passing system (version 3.4) has been used, with GaLib 2.4 [12] as the general genetic algorithm library. The same source code corresponding to the previously described parallel implementation was compiled with the g++ compiler in two different distributed environments, both being Beowulf clusters running Red Hat Linux 7.2 and connected with an EtherFast-100 ethernet switch (Linksys):

– a two-node cluster (ht-cluster), with a Pentium III processor (1000 MHz, 512 MB RAM), and an AMD Athlon processor (750 Mhz, 256 MB RAM).

– a 4 CPU homogeneous cluster (HG-cluster), with two dual Xeon processor (2 GHz, 1 GB RAM) DELL Workstations.

Both clusters were benchmarked with an off-the-shelf Poisson solver giving the following results: (i) ht-cluster: 68.59 MFlops for Pentium III, 111.58 MGlops for Athlon, 180.17 MFlops total; (ii) HG-cluster: 218.5 MFlops/CPU, 874 MFlops total.

3.1 EXAMPLE

The parallel implementation was tested using the Sunspot prediction one dimensional problem [7]. This univariate process describes the American relative sunspot numbers (mean number of sunspots for the corresponding months in the period 1/1945 – 12/1994), from AAVSO - Solar Division [12]. It contains 600 observations, and in this case, the first 400 were used as training and the remaining 200 for testing. A maximum time lag of 30 years was pre-set, defining a search space size of 2^{30} models.

No preprocessing was applied to the time series. This is not the usual way to analyze time series data, but by eliminating additional effects, the properties of the proposed procedure in terms of approximation capacity and robustness are better exposed. The similarity function used was $\mathcal{S} = (1/(1+d))$, where d is a normalized euclidean distance. The number of responsive h-neurons in the hidden layer was set to $k = 7$, and the similarity threshold for the h-neurons was $T = 1$. No attempt to optimize these parameters was made, but meta-algorithms can be used for this purpose.

The experiments have been conducted with the following set of genetic algorithm parameters: number of generations = 2, population size = 100, mutation probability = 0.01, crossover probability = 0.9. Single point crossover and single bit mutation were used as genetic operators with roulette selection and elitism being allowed.

The performance of the two clusters w.r.t the parallel algorithm is illustrated in table 1.

As an illustration of the effectiveness of the method, a run with 2000 generations and 50 individuals per population was made. The best model found contained 10 time lags, namely: (t-1), (t-2), (t-4), (t-10), (t-12), (t-14), (t-16), (t-20), (t-28), (t-29). Its RMS prediction error in the test set was 20.45, and the real and predicted values are shown in Fig 5.

4 Conclusions

Time series model mining using evolutionary algorithms and similarity-based neuro-fuzzy networks with h-neurons is flexible, robust and fast. Its parallel implementation runs well on inexpensive Beowulf clusters, making intensive data mining in time series affordable. This method is appropriate for the exploratory stages in the study of multivariate time varying processes for quickly finding

Table 1. ht-cluster and HG-cluster performance

No. slaves	No. CPUs	Time(secs) Time(secs)	Ratio (2 CPU/ 1 CPU)	Time(secs) Time(secs)	Ratio (4 CPU/ 2 CPU)
1	2	117	0.991	60	1
	4	116		60	
2	2	120	0.642	60	0.45
	4	77		27	
3	2	116	0.689	61	0.377
	4	80		23	
4	2	120	0.658	60	0.467
	4	79		28	
5	2	116	0.672	60	0.45
	4	78		27	

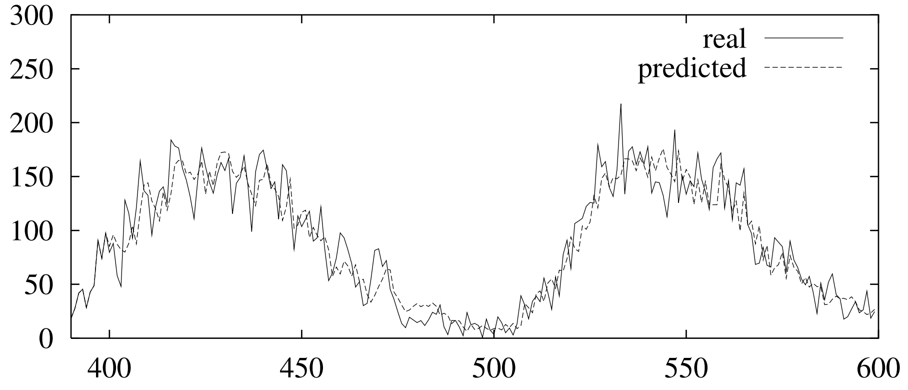

Fig. 5. Comparison of the real and predicted values for sunspot data (test set).

plausible dependency models and hidden interactions between time dependent heterogeneous sets of variables, possibly with missing data. The dependency structure is approximated or narrowed down to a manageable set of plausible models. These models can be used by other methods such as neural networks or non-soft-computing approaches for constructing more accurate prediction operators.

Many parallel implementations of this methodology are possible. Among many elements to be considered are: deeper granularity in the parallelization, use of other chromosome schemes for model representation, variations in the type of genetic algorithm used (steady state, mixed populations, different crossover, mutation and selection operators, etc.), use of other kinds of evolutionary algorithms (evolution strategies, ant colony methods, etc.), variations in the neuro-fuzzy paradigm (kind of h-neuron used, its parameters, the network architecture, etc), the sizes of the training and test set, the maximum exploration time-depth window. These considerations make meta-evolutionary algorithms an attractive approach, introducing a higher hierarchical level of granularity. Furthermore, the intrinsic parallelism of the algorithms allows for efficient parallel implementations. The results presented are promising but should be considered preliminary. Further experiments, research and comparisons with other approaches are required.

5 Acknowledgments

The authors are very grateful to Alan Barton and Robert Orchard from the Integrated Reasoning Group, Institute for Information Technology, National Research Council of Canada. A. Barton made insightful comments and suggestions, while R. Orchard gave his big understanding and support for this project.

References

1. Belanche, Ll. : Heterogeneous neural networks: Theory and applications. PhD Thesis, Department of Languages and Informatic Systems, Polytechnic University of Catalonia, Barcelona, Spain, July, (2000)
2. Birx, D., Pipenberg, S. : Chaotic oscillators and complex mapping feedforward networks for signal detection in noisy environment. Int. Joint Conf. On Neural Networks (1992)
3. Box, G., Jenkins, G. : Time Series Analysis, Forecasting and Control. Holden-Day. (1976)
4. Chandon, J.L., Pinson, S. : Analyse Typologique. Thorie et Applications. Masson, Paris, (1981)
5. Gueist, A., et.al. : PVM. Parallel Virtual Machine. Users Guide and Tutorial for Networked Parallel Computing. MIT Press 02142, (1994)
6. Lapedes, A., Farber, R. : Nonlinear signal processing using neural networks: prediction and system modeling. Tech. Rep. LA-UR-87-2662, Los Alamos National Laboratory, NM, (1987)
7. Masters, T. : Neural, Novel & Hybrid Algorithms for Time Series Prediction. John Wiley & Sons, (1995)
8. Specht, D. : Probabilistic Neural Networks, Neural Networks **3**. (1990), 109–118
9. Valdés, J.J., García, R. : A model for heterogeneous neurons and its use in configuring neural networks for classification problems. Proc. IWANN'97, Int. Conf. On Artificial and Natural Neural Networks. Lecture Notes in Computer Science **1240**, Springer Verlag, (1997), 237–246
10. Valdés, J.J., Belanche, Ll., Alquézar, R. : Fuzzy heterogeneous neurons for imprecise classification problems. Int. Jour. Of Intelligent Systems, **15** (3), (2000), 265–276.
11. Valdés, J.J. : Similarity-based Neuro-Fuzzy Networks and Genetic Algorithms in Time Series Models Discovery. NRC/ERB-1093, 9 pp. NRC 44919. (2002)
12. Wall, T. : GaLib: A C++ Library of Genetic Algorith Components. Mechanical Engineering Dept. MIT (http://lancet.mit.edu/ga/), (1996)
13. Zadeh, L.: The role of soft computing and fuzzy logic in the conception, design and deployment of intelligent systems. Proc. Sixth Int IEEE Int. Conf. On Fuzzy Systems, Barcelona, July 1-5, (1997)