Learning Fuzzy Rule Based Classifier in High Performance Computing Environment

Vinícius da F. Vieira[1], Alexandre G. Evsukoff[1], Beatriz S. L. P. de Lima[1] and Sylvie Galichet[2]

[1]COPPE/Federal University of Rio de Janeiro, Brazil
Rio de Janeiro, Brazil
[2]LISTIC/Polytech'Savoie, University of Savoie,
BP 80439, 74944 Annecy le Vieux, France

Email: vfvieira@gmail.com, alexandre.evsukoff@coc.ufrj.br, bia@coc.ufrj.br, sylvie.galichet@univ-savoie.fr

Abstract *— An approach to estimate the number of rules by spectral analysis of the training dataset has been recently proposed [1]. This work presents an analysis of such a method in high performance computing environment. Two approaches for parallel implementation of the method were studied considering the structure selection genetic algorithm and the spectral decomposition. The results show that both approaches have allowed to reduce considerably the overall processing time.*

Keywords — Fuzzy rule-based classifier; genetic algorithm; spectral clustering; high performance computing.

1 Introduction

Fuzzy rule-based classifiers have been widely studied and many approaches have been reported in literature [2]. One of the main problems in the design of interpretable fuzzy rule-based models is the determination of an adequate number of rules, which is closely related to the complexity of the problem. An approach to estimate the number of rules by spectral analysis of the training dataset has been recently proposed [1]. The method employs also a genetic algorithm for structure selection, which selects the input variables and defines the number of fuzzy membership functions in the domain of each selected input variable. Although this method has obtained interpretable fuzzy rules, with good accuracy, it is very time consuming due to the numerical complexity of both the spectral decomposition and the genetic algorithm.

This work presents an analysis of the method proposed in [1] in high performance computing environment. Two approaches for parallel implementation of the method were studied considering the structure selection by the genetic algorithm and the spectral decomposition. In the first approach, only the genetic algorithm is implemented in parallel while the spectral decomposition is computed in serial. In the second approach both the genetic algorithm and the spectral decomposition are run in parallel in a two levels parallelization scheme.

This paper is organized as follows. Next section introduces the notation and parameters of the fuzzy modeling approach. In section 3 the fuzzy rule induction algorithm [1] is reviewed. In section 4 the parallel implementation is described. In section 5 the results are discussed and the conclusions are drawn in last section.

2 Fuzzy Symbolic Modeling

Consider a dataset composed of N input-output observed samples $T = \{(\mathbf{x}(t), y(t)),\ t = 1 \ldots N\}$, where $\mathbf{x} \in \mathcal{R}^p$ is a vector of input variables and $y \in \mathcal{N}$ is the output variable where a value $y = j$ refers to a class $B_j \in \mathcal{B}$.

The fuzzy symbolic model (FSM) relates a set of input symbols $\mathcal{A} = \{A_i, i = 1 \ldots n\}$ to the set of classes $\mathcal{B} = \{B_j, j = 1 \ldots m\}$ as rules in the form $A_i \rightarrow B_j$ read as:

$$\textit{if}\ \ \mathbf{x}(t)\ \ \textit{is}\ \ A_i\ \ \textit{then}\ \ y(t)\ \ \textit{is}\ \ B_j. \tag{1}$$

As usual, the fuzzy model computes an approximation $\hat{y}(t) = \hat{f}(\mathbf{x}(t))$ in three steps, which are implemented as the operators:

Fuzzification:

$$\hat{\mathbf{u}}(t) = F(\mathbf{x}(t), \boldsymbol{\Xi}). \tag{2}$$

Inference:

$$\hat{\mathbf{v}}(t) = I(\hat{\mathbf{u}}(t), \boldsymbol{\Phi}). \tag{3}$$

Defuzzification:

$$\hat{y}(t) = D(\mathbf{v}(t)). \tag{4}$$

The operators and their parameters, which should be computed by the learning algorithm, are presented in the following [1].

2.1 Fuzzification

The compound symbol A_i in rule (1) represents a region on the input variables domain. The membership function $\mu_{A_i}(\mathbf{x}(t))$ is computed by the conjunction of p elementary symbols $\mu_{A_k^i}(x_k(t)), k = 1 \ldots p$ as:

$$u_i(t) = \mu_{A_i}(\mathbf{x}(t)) = \prod_{k=1 \ldots p} \mu_{A_k^i}(x_k(t)) \tag{5}$$

The component A_k^i in (5) is associated to an elementary fuzzy set defined in the one dimensional domain of the variable x_k. The fuzzy membership functions are supposed to be equally spaced Gaussian functions, and only the number of fuzzy sets in each input variable fuzzy partition is necessary to define fuzzy membership functions [1].

The table $\Xi = \{\xi_{ik}, i = 1 \ldots n, k = 1 \ldots p\}$ is the output of the rule induction algorithm, presented in Section 3.2. An element $\xi_{ik} \in \Xi$ defines which symbol in the partition of the variable x_k should be associated to the component A_k^i in the rule i. A value $\xi_{ik} = j$ indicates that the fuzzy set A_{kj} should be used as the component A_k^i in the rule A_i (cf. (5)).

2.2 Inference

Rule weights have been shown to allow more flexibility to the fuzzy model [3][4]. The inference operator is defined by the rule weights matrix $\mathbf{\Phi} \in [0,1]^{n \times m}$, of which each component $\varphi_{ij} = \mu_\Phi(A_i, B_j)$ represents the confidence of the rule $A_i \to B_j$. The fuzzy inference is computed by the sum-product composition operator:

$$\hat{\mathbf{v}}(t) = \hat{\mathbf{u}}(t).\mathbf{\Phi} \tag{6}$$

The rule weights matrix are computed as the solution of a bounded quadratic optimization problem, as discussed in section 3.3.

2.3 Defuzzification

In classification problems, defuzzification computes the class output index. Generally the maximum rule is used, such that the class index is computed as the component with the greatest membership value:

$$\hat{y}(t) = j : v_j(t) = \max(\hat{\mathbf{v}}(t)) \tag{7}$$

The design of the FSM is presented next.

3 Induction of Fuzzy Rules

The rule induction algorithm is based on the spectral analysis of the training dataset to determine the number of fuzzy rules. The rule-base weights are then computed by solving a bounded quadratic optimization problem.

3.1 Spectral Analysis

Consider a graph associated to the training dataset, of which the set of nodes represent the input variable samples and the set of edges is defined by the $N \times N$ adjacency (or affinity) matrix $\mathbf{A}$. The graph is undirected and weighted, hence the affinity matrix is symmetric, real valued and its elements represent the similarity between two input variable samples. In this work, the similarity metric is computed by the Gaussian function, such that each element of affinity matrix is computed as:

$$a_{ij} = \begin{cases} \exp\left(-\dfrac{\|\mathbf{x}(i) - \mathbf{x}(j)\|^2}{2\sigma^2}\right), & \mathit{if}\ i \neq j \\ 0 & \mathit{otherwise} \end{cases} \tag{8}$$

where σ is a dispersion parameter that controls the spread of the similarity function, fixed in this work at $\sigma = 1$. The $N \times N$ matrix $\mathbf{D}$ is a diagonal matrix whose elements are the degree of the nodes of the graph, computed by the sum of similarities of neighbors of each node:

$$d_{ii} = \sum_{j=1..N} a_{ij} \tag{9}$$

The spectral analysis of the graph is based on the normalized Laplacian matrix, computed as [5][6]:

$$\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \tag{10}$$

The rule induction algorithm is based on the spectral decomposition of the Laplacian matrix as:

$$\mathbf{L} = \mathbf{Z}\mathbf{\Lambda}\mathbf{Z}^{\mathrm{T}} \tag{11}$$

The number of positive eigenvalues of the Laplacian is used as an estimation of the number of rules [1].

3.2 The Rule Induction Algorithm

The combination of one-dimensional fuzzy sets is easier to understand than a multi-dimensional fuzzy set. Thus, the idea of the rule induction algorithm is to run a clustering algorithm on the dataset and then to associate a rule to each cluster. For each cluster, the nearest combination to the cluster center, among all possible combinations of fuzzy sets is chosen as the rule to be included in the model.

In the spectral clustering algorithm, the standard k-means algorithm is applied over the matrix $\hat{\mathbf{Z}}$, computed as the first K columns of the eigenvector matrix $\mathbf{Z}$. Recall that the columns of $\mathbf{Z}$ are ordered according to the corresponding eigenvalues, such that the first columns correspond to the largest eigenvalues.

The rule induction algorithm is run within an outer loop for structure selection, which defines both the variables that should be used by the model and the number of fuzzy sets in each variable's domain. At each iteration, the structure selection algorithm provides the candidate structure represented by the vector $\boldsymbol{\gamma} = [\gamma_1, \ldots, \gamma_p]$, where each component $2 \le \gamma_k \le \gamma_{\max}$ is the number of fuzzy sets in the fuzzy partition of variable x_k and a value $\gamma_k = 1$ indicates that the variable x_k should not be considered by the model. The rule induction algorithm is sketched out in Algorithm 1.

Algorithm 1: Rule induction

```
input:  $\boldsymbol{\gamma} = [\gamma_1, \ldots, \gamma_p]$ ; $T = \{(\mathbf{x}(t), \mathbf{y}(t)), t = 1...N\}$
output: $\Xi = \{\xi_{ik}, i = 1 \ldots n, k = 1 \ldots p\}$ ;
01 begin
02    compute the matrix $\mathbf{A}$    //Equation (8)
03    compute the matrix $\mathbf{D}$    //Equation (9)
04    compute $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
05    compute $\mathbf{L} = \mathbf{Z}\mathbf{\Lambda}\mathbf{Z}^{\mathrm{T}}$
06    $n \leftarrow K : \lambda_i \ge \delta, i = 1...K$   //estimate $n$ rules
07    $\mathbf{W} \leftarrow \mathit{clustering}(n)$   //assign rules to clusters
08    for $i = 1 \ldots n$
09       for $k = 1 \ldots p$
10          if ( $\gamma_k > 1$ )
11             $\xi_{ik} = j^* : \alpha_{kj^*} = \arg\min(\alpha_{kj} - w_{ik})$ ;
12 end
```

The set of rules issued by the rule induction algorithm defines the premises of rules (1). The same set of rules is used for all classes, with the classification being assigned by the rule weights, which are computed as described next.

3.3 Rule Weights Optimization

The rule weight matrix can be written as $\mathbf{\Phi} = [\boldsymbol{\varphi}_1 | \ldots | \boldsymbol{\varphi}_m]$, where the vectors $\boldsymbol{\varphi}_j$ are the rule confidence weights that relates the rule premise symbol A_i to the rule consequent (class) B_j. Optimal rule weights $\boldsymbol{\varphi}_j$ can be computed in the least squares sense, by solving the following bounded quadratic programming problem for each class B_j [1][7]:

$$\underset{\boldsymbol{\varphi}_j}{\textit{minimize}} \ \tfrac{1}{2}\boldsymbol{\varphi}_j^{\mathrm{T}}\mathbf{K}\boldsymbol{\varphi}_j - \mathbf{C}^{\mathrm{T}}\boldsymbol{\varphi}_j \qquad (12)$$
$$\textit{subject to} \ 0 \le \varphi_{ij} \le 1, \ i = 1...n$$

where $\mathbf{K} = \mathbf{U}^{\mathrm{T}}\mathbf{U}$ is a strictly positive definite matrix and $\mathbf{C}^{\mathrm{T}} = \mathbf{V}_j^{\mathrm{T}}\mathbf{U}$. The matrix $\mathbf{U} = [\mathbf{u}(1) | \ldots | \mathbf{u}(N)]^{\mathrm{T}}$ is the matrix of which each line is the fuzzy vector computed for each data sample, whose components are computed as (5). The vector $\mathbf{V}_j = [v_j(1) | \ldots | v_j(N)]^{\mathrm{T}}$ contains the memberships values of class B_j for each sample in the training set, such that $v_j(t) = \mu_{B_j}(y(t))$. The bounds attempt to constrain the weights within the $[0,1]$ interval, such that they can be interpreted as fuzzy rule confidence. The bound constrains also avoid high values for the rule weight parameters, providing the smoothness of the solution.

The quadratic optimization problem defined by (12) is widely known and the solution can be computed by efficient numerical algorithms [8].

For a fixed model structure, the more the solution values $\varphi_{ij} \approx 1$, the more certain is the classifier and also the greater is the norm of the weight vector, defined by:

$$\left\|\boldsymbol{\varphi}_j\right\|^2 = \boldsymbol{\varphi}_j^{\mathrm{T}}.\boldsymbol{\varphi}_j, \ j = 1...m \qquad (13)$$

The sum $\sum_{j=1..m} \left\|\boldsymbol{\varphi}_j\right\|^2$ is thus a measure of the amount of certain rules and also can be interpreted as a measure of the classifier quality.

3.4 Structure Selection

The structure selection is implemented by a genetic algorithm (GA). Each individual in the population represents an estimation of the structure vector $\boldsymbol{\gamma} = [\gamma_1, \ldots, \gamma_p]$, which defines the variables that must be included in the model and also the number of fuzzy sets in the domain of each variable.

The GA fitness function is computed as a trade-off between the model's accuracy and capacity according to the required complexity of the problem [1]:

$$R(\boldsymbol{\gamma}) = \frac{1}{2N} \sum_{j=1...m} \left\|\mathbf{U}(\boldsymbol{\gamma})\boldsymbol{\varphi}_j(\boldsymbol{\gamma}) - \mathbf{V}_j\right\|^2 + \frac{1}{2N} \sum_{j=1...m} \left\|\boldsymbol{\varphi}_j(\boldsymbol{\gamma})\right\|^2 \qquad (14)$$

where the firs term represent the model accuracy and the second accounts for model quality according to (13). The factor $\frac{1}{2N}$ is used to rescale the fitness function to the $[0.1]$ interval.

The GA performs simultaneously the selection of input variables to be considered in the model and the number of fuzzy sets in the domain of each (selected) variable. During the GA execution, it is frequently necessary to evaluate an individual that has been previously evaluated. The number of fuzzy rules is computed from the spectral analysis, which is a function only of the selected input variables. It is thus possible that two different individuals select the same input variables such that they will result in the same number of rules. In order to avoid the unnecessary reevaluation of the spectral analysis for individuals related to the same subset of variables, the number of rules computed by the solution of (11) is stored and related to the corresponding subset of variables. Consequently, the eigendecomposition (11) for a new individual is computed only if the corresponding subset of variables has not been evaluated before. The fitness functions for all visited individuals are also stored such that for a new individual, identical to another already evaluated, the fitness function (14) is not reevaluated.

The GA parameters for all tests described in section 5 were the same. The population was fixed at 32 individuals. An uniform crossover operator with 85% crossover rate was used. The mutation rate was set in 5%. The maximum number of iterations was fixed at 150. A value $\gamma_{\max} = 8$ was used to limit the number of fuzzy sets per variable.

All the implementation was made by free source software available on the web. The GA was implemented in Python programming language. The most computationally expensive tasks (spectral analysis and the rule weights optimization) were implemented in C using standard libraries.

4 Parallel Implementation

The structure selection GA is a very intensive computational task, since each candidate evaluation requires the solution of equation (11) and the GA itself requires the evaluation of a number of individuals during several iterations. The modern multi-core hardware platforms allow the reduction of the global processing time in complex application through parallel implementation.

Depending on the application and the available hardware resources, the parallelism can be exploited in different ways, through the partitioning of the problem with respect to data and/or tasks. In data parallelism, the same task is allocated to different processors, where each one of them works with a different portion of the data. This approach applies to problems whose solution involves a great mass of data. In task parallelism, different tasks are allocated to different processors. This approach applies to problems where there is interdependency among the tasks.

When designing a parallel program, the communication among the processors involved in the execution must be considered carefully. There are two main communication paradigms in parallel applications: memory sharing and message passing. The memory sharing parallel programming model is based in the same principle as the serial programming; essentially, a single memory addressing space is used during the program execution flow. The message passing programming is conceptually independent of the hardware platform, operating system and programming language. As the memory addressing spaces are different in each process the concept of message is used. A message is a set of information that can be sent from process to another through a communication channel.

One of the most known and complete libraries for message passing is the Message Passing Interface (MPI). The MPI is a standardized and portable message passing library designed for distributed memory environments, parallel machines and heterogeneous network. In MPI programs, all the parallelism is explicit, i.e., the programmer must code the synchronization points and the message passing calling the functions. Each process, receives a copy of the entire parallel program code.

In this work two parallel strategies are evaluated: in the first one only the GA is implemented in parallel; in the second one, both the GA and the spectral analysis for each individual are processed in parallel in a two levels strategy.

4.1 Parallel Genetic Algorithm

Several features of GAs allow its parallelization in a very natural way and many approaches have been reported in literature [9]. The main approaches are divided in three types: fine-grained single population GA, coarse-grained multiple population GA and single global population master-slave GA.

Fine-grained parallel genetic algorithms use just one population, but there is a special structure that limits the interaction among the individuals. An individual can only compete and operate with individuals that belong to his neighborhood, which is assigned by a predefined topology.

Coarse-grained parallelization model, also known as island model, uses several populations that occasionally change individuals through migration operator. In island GAs, each sub-population evolves as in traditional GAs, excepting for the migration operator among different islands.

The master-slave parallel genetic algorithms, also known as parallel global population GAs, use a single population and the parallelization is done in the evaluation of the individuals. As in a traditional serial GA, each individual competes and operates with any other individuals in the population. In master-slave genetic algorithms, a master process is responsible by the application of the genetic operators to the population (selection, crossing, mutation) while the slave processors are responsible by the individuals' evaluation [10]. The communication among the individuals occurs when the master process sends the individuals to the slave processes and when the slave processes return their fitness values to the master.

In this work, a master-slave parallel implementation was used for the structure selection GA. The adoption of such GA parallel model is useful, since its implementation is independent of the machine architecture and of the communication model among the processes.

4.2 Two Levels Approach: Parallel GA with Parallel SA

The complexity of a full eigenproblem like (11) is $O(N^3)$ in processing time and $O(N^2)$ in storage requirements. In Algorithm 1, the solution of (11) requires the solution of an eigenproblem of the size of the number of records of the dataset N. The two-level strategy considers thus the parallelization of the spectral analysis (SA) for each individual, within the parallel GA.

The schematic representation of the two-level strategy is shown in Fig. 1. In the higher level there is the GA parallelism, where the master performs the operations over the population and sends the individuals to the slaves. In the lower level is the SA parallelism, where each slave shares the eigenproblem solution with other processors.

The two levels strategy is possible due to the concept of communicators groups in the MPI library that allows the definition of modules that encapsulate the communication operations within a group. The communicator identifies the group of processes and the context in which a given operation must be applied. The MPI library provides a standard communicator, called MPI COMM WORLD, which identifies every processes involved in the computation through global identifiers.

The execution of the GA is performed by the processors that belong to Group 0 (cf. Fig. 1.). Each slave processor in Group 0, also belongs to another group, which is responsible for computing the SA. Thus, new sub-communicators must be created, allowing the communication within each group of slave processors.

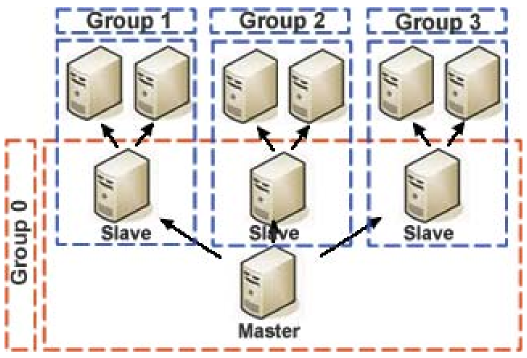

Fig. 1: Two-level Parallel Architecture

There are numbers of software libraries that implements parallel eigenproblem solution using different numerical methods. In this work, SLEPc (Scalable Library for Eingenvalue Problem Computation [11]) was adopted. SLEPc is a software package for solution of large scale problems built over PETSc (Portable, Extensible Toolkit for

ISBN: 978-989-95079-6-8

Scientific Computation), which is a set of libraries of data structure and routines for parallel scientific applications.

The Krylov-Schur solver, which implements the Arnoldi method for eigenvalues problem solution, was adopted in this work. It is probably the most robust solver in SLEPc [11].

5 Results and Discussion

5.1 *Performance Results*

Four benchmark datasets from the UCI Machine Learning Repository [12] were considered for the performance evaluation. Some synthetic datasets, with 18 variable and different numbers of records, were also considered. These datasets were considered in order to exploit a greater problem size.

Each benchmark evaluation was performed in 10-fold cross validation analysis. The evaluation of each benchmark was repeated 10 times to ensure the GA generalization. The average results for each dataset are shown in Table 1, where Acc. is the average accuracy (computed for the 10-fold cross validation) and Time is the average global time (in seconds) of the serial implementation. For the synthetic datasets just the time results were considered and the time values correspond to the execution time for just one GA generation.

All tests were run an 8-core Linux cluster composed of two machines, each one equipped with two dual-core Intel Xeon processors (1.6GHz) and 4GB DRAM. The serial processing time was computed using only one processor.

Table 1: Performance Results.

Dataset	p	m	N	rules	Acc.	Time
Diabetes	8	2	768	12.5	78.67	1795.72
Ionosphere	32	2	344	26.7	91.47	4752.24
Balance	4	3	625	8.3	81.41	145.69
Cancer	9	2	683	8.4	98.13	1665.62
Synthetic 1	18	-	1000	-	-	99.68
Synthetic 2	18	-	2000	-	-	580.18
Synthetic 3	18	-	3000	-	-	1952.7
Synthetic 4	18	-	4000	-	-	4339.75

As seen in Table 1, the modeling approach has obtained good accuracy with small number of rules results. The accuracy results obtained for the same datasets by other algorithms can be found in [1]. The focus of the analysis in this work is on the processing time and parallelization efficiency.

5.2 *Parallel GA Results*

The processing time (in seconds) of the parallel GA (with serial SA) implementation are shown in Table 2. The parallel processing time as a percentage of the serial processing time is also shown in Table 2. It can be seen that the parallel GA reduces the global execution time considerably.

Table 2: Execution time for parallel GA (with serial SA).

	2 Proc.		4 Proc.		8 Proc.	
	Time	%	Time	%	Time	%
Diabetes	1194.8	66.5	738.28	41.11	550.75	30.67
Ionosphere	2517.57	52.97	1354.1	28.49	853.56	17.96
Balance	107.29	73.64	88.35	60.64	69.51	47.71
Cancer	1070.35	64.26	696.54	41.81	476.76	28.62
Synthetic1	54.79	54.96	47.66	47.81	54.16	54.33
Synthetic2	317.14	54.66	210.73	36.32	230.89	39.79
Synthetic3	1050.2	53.78	700.89	35.89	620.05	31.75
Synthetic4	2380.84	54.86	1765.83	40.68	1486.28	34.24

The parallel efficiency of the parallel GA algorithm is shown in Fig. 2. The parallel efficiency was computed in the standard way as [13]:

$$\varepsilon_P = 100.\frac{1}{P}\frac{T_S}{T_P} \qquad (15)$$

where P is the number of processors, T_S the serial processing time (in seconds) and T_P is the parallel processing time using P processors.

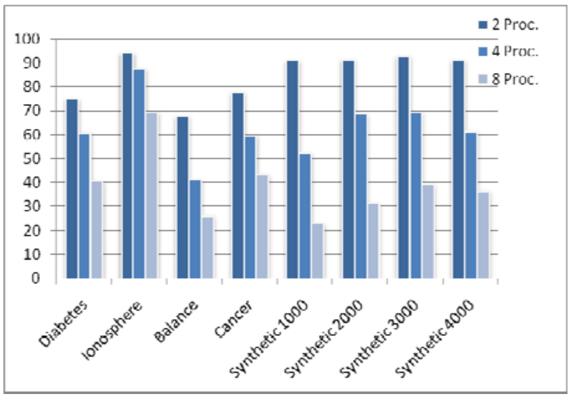

Fig. 2: Parallel Efficiency for Parallel GA With Serial SA.

The parallel efficiency decreases as the number of processors increases, due communication overhead and the load imbalance introduced by the reevaluation avoiding scheme, since the number of evaluated individuals is not the same for all the slave processes. A processor with fewer individual's evaluations will become idle waiting the other processors finish their work to synchronize in a new generation. The resulting load imbalance reflects in a great damage to the overall parallel efficiency.

The results show that this approach is not good for large datasets, since each GA individual evaluation allocates an instance of the Laplacian matrix in memory, making a bad use of the memory resources available.

5.3 *Parallel GA with Parallel SA Results*

The execution times (in seconds) of the two levels approach for the studied datasets are shown in Table 3 as so as the percentage of the serial processing time.

Table 3: Execution times for the two levels approach.

	2 Proc.		4 Proc.		8 Proc.	
	Time	%	Time	%	Time	%
Diabetes	2185.32	121.6	1303.46	72.58	833.62	46.42
Ionosphere	5704.34	120.0	3401.24	71.57	2949.77	62.07
Balance	185.67	127.4	128.28	88.05	116.71	80.10
Cancer	1628.91	97.79	994.00	59.67	745.43	44.75
Synthetic1	76.01	76.25	40.03	40.15	35.41	35.52
Synthetic2	368.81	63.56	195.59	33.71	160.38	7.64
Synthetic3	1182.08	60.53	615.06	31.49	416.85	21.34
Synthetic4	2537.81	58.47	1300.7	29.97	805.05	18.55

ISBN: 978-989-95079-6-8

The parallel efficiency of the two-level approach for the same datasets are shown in Fig. 3.

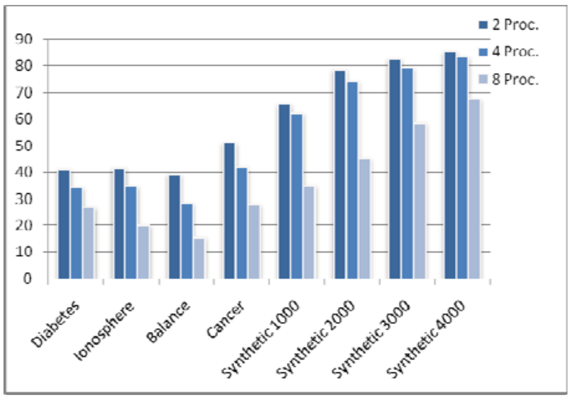

Fig. 3: Parallel Efficiency for the Two-Level Approach.

The results in Table 3 and Figure 3 show that for the four benchmark datasets studied, the global execution time is greater than the results presented for the parallel GA with serial SA. Consequently, the parallel efficiency is lower when compared to the results presented by those approach. This indicates that for small size datasets, the parallelization of the eigenproblem solution is not efficient. For larger datasets, the two levels approach shows more interesting results since the synthetic datasets present better results for the two levels approach when compared to the parallel GA with serial SA. This comparison indicates that, for datasets with a great number of records, the SA parallelization can be a good idea since it reduces the load imbalance introduced by the reevaluation avoiding strategy and makes a better use of the memory available for the method execution, since the Laplacian matrix used in the spectral analysis is sliced for the involved processes.

6 Conclusions

The design of fuzzy symbolic models [1] is focused on the interpretability and the resulting model's structure is selected by a wrapper genetic algorithm that defines the variables that should be included into the model and the number of fuzzy sets in the partition of each variable. The number of rules is determined by a spectral analysis method and rules' weights are optimized.

This work has evaluated two approaches for the parallel implementations of the fuzzy symbolic models. Both approaches have reduced the global execution time of the algorithm when compared to the serial times, which was the main drawback of the previous implementations.

The parallel efficiency results indicate that, even with the load imbalance introduced by the reevaluation avoiding strategy, the GA parallelization is an interesting approach for smaller datasets and the two-level approach is a good approach for larger datasets. Further studies may be done in order to increase the load balance among the processors and evaluate the implementation behavior for larger datasets.

Acknowledgement

The authors are grateful to the Laboratory of Computational Physiology (FISIOCOMP) at Federal University of Juiz de Fora (UFJF), where all the parallel experiments were run. This work has been supported by the Brazilian Research Council CNPq and the Brazilian Agency CAPES under the agreement CAPES/Cofecub.

References

[1] A.G. Evsukoff, S. Galichet, B. S. L. P. de Lima and N. F. F. Ebecken, "Design of interpretable fuzzy rule-based classifiers using spectral analysis with structure and parameters optimization". Fuzzy Sets and Systems (2008), doi: 10.1016/j.fss.2008.08.010.

[2] E. Hüllermeier, "Fuzzy methods in machine learning and data mining: status and prospects", Fuzzy Sets and Systems, vol. 156, no. 3, pp. 387-406, 2005.

[3] H. Ishibuchi and T. Nakashima, "Effect of rule weights in fuzzy rule-based classification systems" IEEE Trans. Fuzzy Systems, vol. 9, no. 4, pp. 506–515, 2001.

[4] H. Ishibuchi and T. Yamamoto, "Rule weight specification in fuzzy rule-based classification systems", IEEE Trans. on Fuzzy Systems, vol. 13, no. 4, pp. 428-435, 2005.

[5] M. Filippone, F. Camastra, F. Masulli and S. Rovetta, "A survey of kernel and spectral methods for clustering". Pattern Recognition, vol. 41, pp. 176-190, 2008.

[6] F. R. K. Chung, Spectral Graph Theory, CBMS Regional Conf. Series in Mathematics., no. 92. American Mathematic Society, 1997.

[7] A. G. Evsukoff, "Learning fuzzy rule-based classifier with rule weights optimization and structure selection by a genetic algorithm", Proceedings of the 2007 IEEE International Conference on Fuzzy Systems, London, 23-26 July, 2007.

[8] B. Schölkopf and A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, 2001.

[9] E. Cantú-Paz, "A Survey of Parallel Genetic Algorithms", Calculateurs Paralleles, vol. 10, 1998.

[10] C. R. Xavier, V. F. Vieira, D. M. S. Martins, R. W. Santos, "Comparing Two Parallel Genetic Algorithms for the Inverse Problem Associated to the Cardiac Bidomain Equations", IEEE 18th International Symposium on Computer Architecture and High Performance Computing, Ouro Preto, 17-20 October, 2006.

[11] V. Hernandez , J. E. Roman, V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems", ACM Transactions on Mathematical Software, v.31 n.3, p.351-362, September 2005.

[12] A. Asuncion, and D. J. Newman, "UCI Machine Learning Repository" [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science, 2007.

[13] V. Kumar, A. Grama, A. Gupta and G. Karypis, Introduction to Parallel Computing - Design and Analysis of Algorithms, The Benjamin/Cummings Publishing Company, Inc., CA, 1994