A Parallel Genetic Algorithm for Rule Discovery in Large Databases

Dieferson Luis Alves de Araujo[1] , Heitor S. Lopes[1], Alex A. Freitas[2]

[1] CEFET-PR – Centro Federal de Educação Tecnológica do Paraná
CPGEI - Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial
Av. 7 de setembro, 3165 - 80230-901 Curitiba (PR) - BRAZIL
dief@cpgei.cefetpr.br, hslopes@cpgei.cefetpr.br

[2] PUC-PR – Pontifícia Universidade Católica do Paraná
PPGIA – Programa de Pós-Graduação em Informática Aplicada
Rua Imaculada Conceição, 1155 - 80215-901 Curitiba (PR) - BRAZIL
alex@ppgia.pucpr.br

ABSTRACT

This paper presents GA-PVMINER, a parallel genetic algorithm that uses Parallel Virtual Machine (PVM) to discover rules in a database. The system uses the Michigan's approach, where each individual represents a rule. A rule has the form "*if* condition *then* prediction". GA-PVMINER is based on the concept learning framework, but it performs a generalization of the classification task, which can be called dependence modeling (sometimes also called generalized rule induction). In this task, different discovered rules can predict the value of different goal attributes in the "*prediction*" part of a rule, whereas in classification all discovered rules predict the value of the same goal attribute. The global population of genetic algorithm individuals is divided into several subpopulations, each assigned to a distinct processor. For each subpopulation, all the individuals represent rules with the same goal attribute in the "*prediction*" part of the rule. Different subpopulations evolve rules predicting different goal attributes. The system exploits both data parallelism and function parallelism.

1. INTRODUCTION

One of the most studied data mining tasks is classification, which is often cast as a concept learning task, where the goal is to discover IF-THEN prediction rules.

Concept learning is the task of finding an intentional description, in some symbolic language, for a set of instances of the concept itself. Usually, propositional or first order logic is assumed as language and the concept description takes the form of a disjunction of conjunctive formulas [1].

Multimodal concept learning can be a difficult task, mainly in the presence of noise [2], [3], [4]. For this reason, and also because Genetic Algorithm (GA) is an emergent tool for concept learning [5], [6], [7], this paper proposes a new model for a parallel GA-based data mining. The proposed system is based on the concept learning framework, but it performs a generalization of the classification task, which can be called dependence modeling (sometimes also called generalized rule induction). In this task, different discovered rules can predict the value of different goal attributes in the THEN part of a rule, whereas in classification all discovered rules predict the value of the same goal attribute.

In general, GA-based concept learning is based on either the Pittsburgh or the Michigan approaches. The Pittsburgh approach resembles the traditional GA where every individual in the population is a set of rules, representing a complete solution to the learning problem. Crossover and mutation are then applied in the usual way to create new generations of such populations. The Michigan approach, however, has generally used a different partial solution to the overall learning task. Each individual represents a single rule, and, in general, only through cooperation with the other rules in the population the overall problem is solved. In our system each individual represents a single rule. However, the evaluation of a rule (individual) is done independently from other rules, i.e., there is no need for cooperation between rules in the sense of the traditional Michigan approach.

The paper is organized as follows. Section 2 describes the GA-PVMINER system, introducing the model and the genetic operators that were used. Section 3 describes the results of the application of GA-PVMINER to two public-domain databases. In section 4 results are analyzed. Finally, results are discussed and conclusions presented in section 5.

2. THE GA-PVMINER ALGORITHM

2.1 Rule Format and Chromosome Encoding

In GA-PVMINER each individual represents a single prediction rule. A rule is described as: "IF *C* THEN *P*", where *C* and *P* mean respectively the condition and the prediction of the rule.

Condition is a conjunction of terms, for instance: *term1 and term2 and ...* . It has a maximum number of terms (*max_term*) which is defined by the user. Each term is a triple <*attribute, operator, value*>. The element *operator* in this triple is "=" when the attribute is categorical. If the attribute is continuous, the term is a tuple <*attribute, operator, value_from, value_to*>, where the element *operator* is "*in*" and the elements *value_from* and *value_to* are values belonging to the domain of the element *attribute*. In this paper, however, we mine data sets containing only categorical attributes.

Prediction is a single term that contains a goal attribute, i.e., it is a triple <*goal-attribute, operator, value*>. Goal attributes candidates to be selected for the prediction part of the rule are previously specified by the user in a meta-data file. The element operator is always "=", since we assume that the goal attribute to be predicted is categorical. The element value in the triple is a value belonging to the domain of corresponding goal attribute.

Research supported by a CAPES grant to D.L.A. Araujo.

Notice that different rules can have different goal attributes in their prediction parts. Therefore, as mentioned before, the data mining task being performed by GA-PVMINER can be seen as a generalization of the well-known classification task, where all rules have the same goal attribute in their prediction part.

The above rule format is encoded into a chromosome in a relatively straightforward manner. More precisely, the chromosome encoding is divided into two parts. The first one is the *C* part of rule, consisting of *K* terms, where *K* is a number in the range [1..*max_term*]. The second part of the chromosome is the P part of the rule, consisting of a single term. We use a variable-length chromosome encoding, i.e., different rules (individuals) can have different number of attributes in their condition.

2.2 Population Generation and Distribution

The population is divided into several subpopulations, each of them with *M* individuals. For each subpopulation, all the individuals represent rules with the same goal attribute and the same goal attribute value in the *P* part of the rule.

The number of subpopulations is a user-specified parameter, and it should be greater than or equal to the number of goal attributes indicated by the user. If this constraint is not respected, some goal attribute(s) will not occur in any discovered rule.

For each subpopulation, the goal attribute associated with that subpopulation is chosen randomly by using an uniform probability distribution, out of the set of goal attributes indicated by the user. Note that two or more different subpopulations can be associated with the same goal attribute. However, even if this occurs, the two subpopulations can still evolve rules with different conditions. After the evolution of the GA, the best rule (individual) of each subpopulation is reported to the user.

Our motivation for dividing the population into subpopulations is the following. First, an individual can mate only with another individual of the same population. This is a simple solution to the problem of avoiding the exchange of genetic material between individuals (rules) that are being evolved to predict different goal attributes. Second, this kind of distribution can be easily implemented in a parallel system, where subpopulations evolve in parallel. This topic is discussed in section 2.5.

2.3 Fitness Evaluation

The fitness function measures the goodness of a rule (individual). The fitness function used in this work is based on the J-measure, proposed by Smyth and Goodman [8], which measures the degree of interestingness of a rule. The higher the value of the J-measure for a rule, the more interesting the rule is. The J-measure is defined as follows (recall that a rule has the form "IF *C* THEN *P*"):

$$\begin{cases} Let \quad b = \dfrac{|C \,\&\, P|}{|C|} \\ Let \quad a = \dfrac{|P|}{N} \\ Jmeasure = \dfrac{|C|}{N}\left(b \cdot \log\left(\dfrac{b}{a}\right) + (1-b) \cdot \log\left(\dfrac{(1-b)}{(1-a)}\right)\right) \end{cases} \qquad [1]$$

where: $|C|$ is the number of data instances which are covered by (i.e. satisfy) the *C* part of the rule; $|P|$ is the number of data instances which are covered by the *P* part of the rule; $|C \,\&\, P|$ is the number of data instances which are covered by both the *C* part and the *P* part of the rule; and *N* is the total number of data instances being mined.

Equation [1] can be conceptually divided into two parts, to render it somewhat more understandable. The first part, which we call p_1, is the fraction $|C|/N$ before the outermost brackets, and represents a bias favoring the generality of the rule. The second part, which we call p_2, is the remaining part of the formula. p_2 is derived from information theory, and is essentially a distance measure between our a *posteriori* belief about P and our a *priori* belief about *P*.

We tried to use the J-measure directly as the fitness function of the individuals (rules). However, our early experiments showed two problems with this approach, which led us to make two modifications, as follows.

First, we realized that the J-measure has a high value (indicating that the underlying rule is a high-quality one) when there is a large difference between our *a posteriori* belief about *P* and our *a priori* belief about *P*. The problem is that this difference is measured in a symmetric way, i.e., the J-measure will have a high value if either our *a posteriori* belief about *P* is much larger than our a priori belief about *P* or vice-versa. In the former case the rule has a high predictive accuracy. However, in the latter case, the rule has a low predictive accuracy. To avoid the discovery of this kind of rule, we have used a "one-sided" variant of the J-measure, defined as [9]:

$$J1measure = \frac{|C|}{N}\left(b \cdot \log\left(\frac{b}{a}\right)\right) \qquad [2]$$

Second, we noted that individuals of the first generations had a fitness value of zero, due to the fact that they were not covering any training examples. In order to solve this problem, we have extended the fitness function with a measure of the "correlation" between the attribute values in the rule antecedent and the goal attribute value in the rule consequent. The basic idea of this measure is to compute the ratio of the number of "potentially useful attributes" over the total number of attributes in the rule antecedent. A given attribute *A* is said to be potentially useful if there is at least one training example having both the *A*'s value specified in the rule antecedent and the goal attribute value specified in the rule consequent.

In order to combine this extension with the equation [2] in the fitness function, it was necessary to normalize the above J1-measure. The ratio of potentially useful attributes over the total number of attributes is already normalized, returning a value between 0 and 1. Finally, the fitness function used in our system is:

$$fitness = \frac{w_1 \cdot (J1) + w_2 \cdot \left(\dfrac{n_{pu}}{n_T}\right)}{w_1 + w_2} \qquad [3]$$

where: *J1* is as defined in equation [2] (normalized to return a value between 0 and 1); n_{pu} is the number of potentially useful attributes in the rule antecedent; n_T is the total number of attributes in the rule antecedent; w_1, w_2 are user-defined weights.

For all experiments described in this paper, w_1 and w_2 were set to 0.6 and 0.4 respectively.

2.4 Genetics Operators

2.4.1 Crossover

The crossover operator is based on that used in GA-MINER [10]. Given two parents, the algorithm first checks whether or not they have a common attribute occurring in their *C* part. There are two possibilities, as follows:

(a) If the algorithm finds one or more attributes which occur in the C part of both parents, it randomly chooses one term out of all the terms which have a common attribute in both parents. Then it swaps the value element of the term in the first parent with its counterpart in the second parent.
Example:
parent1: (age in 21..30) and (gender = 'male')
parent2: (age in 18..20) and (salary in 130..500) and (gender = 'female')
after the crossover based on the attribute gender:
child1: (age in 21..30) and (gender = 'female')
child2: (age in 18..20) and (salary in 130..500) and (gender = 'male')

(b) If the algorithm finds no attribute occurring in the *C* part of both parents, then it randomly chooses a term of the first parent and inserts it into the second parent with a probability equal to (*max_term* - *K*), where *K* is the number of the terms in the *C* part of the second parent. Therefore, the probability of adding one more term to the genome of the second parent is inversely proportional to the current length (number of terms) of the second parent. The motivation for this idea is to favor the discovery of somewhat shorter (hopefully, more comprehensible) rules. A similar computation is performed to decide whether or not to insert a randomly chosen term of the second parent into the first parent.
Example assuming *max_term* = 5:
parent1: (age in 21..30) and (gender = 'male')
parent2: (salary in 130..500) and (marital_status = 'single')

Since there is no common attribute in both parent1 and parent2, suppose the term containing "gender" is chosen in the parent1. This term will be inserted into parent2 to produce child1 with probability (5 – 2)/5, i.e. 60% (3/5). Child1 would then be:
child1: (salary in 130..500) and (marital_status = 'single') and (gender = 'male')

A similar computation would be performed to decide the insertion a randomly chosen term of parent2 into child2.
Finally, recall that an individual can mate only with other individuals of the same subpopulation, as mentioned above.

2.4.2 Mutation

Given an individual, the algorithm randomly chooses a term to undergo mutation. There are two types of mutation, namely attribute mutation and value mutation. The type of mutation to be applied is chosen at random, with a 50% probability for each type. These two types of mutation work as follows:

(a) Attribute mutation:
The algorithm replaces the current attribute in the term undergoing mutation with another randomly chosen valid attribute. An attribute is said to be valid if it does not occur yet in the other terms of the rule (individual). Whenever an attribute mutation occurs, the elements operator and value of the term undergoing mutation are randomly generated (in a manner similar to the generation of these elements in the initial population).

(b) Value mutation:
The algorithm replaces the current value element of the term with another randomly generated value belonging to the domain of the corresponding attribute in the term.

2.4.3 Removal

Given an individual, the algorithm removes one of its terms in the *C* part of rule with probability proportional to the number of these terms. The probability of removal varies from 0, when the *C* part of the rule has only one term, to 0.9, when it already has *max_term* terms. The aim of this operation is to improve rule comprehensibility by shortening the *C* part of the rule.
Example:
Individual before removal: (age in 21..30) and (gender = 'male')
Individual after removal: (gender = 'male')

2.5 Exploiting Parallelism

As mentioned above, in GA-PVMINER the global population is divided into several subpopulations, so that different subpopulations evolve rules (individuals) predicting different goal attributes. Each subpopulation is assigned to a distinct processor, so that different subpopulations evolve in parallel. In addition, the data being mined is also partitioned across the available processors. This approach has two related advantages. First, it allows the exploitation of data parallelism (see below). Second, it avoids the problem of replicating the data being mined across all processors, which would reduce scalability for large data sets.

Each generation of the genetic algorithm consists of two phases, namely fitness evaluation and application of genetic operators. Both these phases exploit parallelism, as follows.

The fitness evaluation phase exploits data parallelism. Fitness evaluation is performed by having the individuals passing through all the processors in a kind of round-robin scheme. To implement this scheme, the processors's physical interconnections are mapped into a logical ring of processors. At first each processor computes a partial measure of fitness for its local individuals by accessing only its local data set. After this partial fitness computation, each processor transfers its local population of individuals (as well as the partial value of their fitness measure) to its "right neighbor" in the ring of processors. Each neighbor then computes the partial fitness measure of the incoming individuals on its local data set; combines this partial fitness measure with the previous one of the incoming individuals, to produce a new partial fitness measure; and forward the incoming individuals (with the updated partial fitness measure) to its right neighbor. This process is repeated until all individuals have passed through all the processors and returned to their original processors, with their final fitness value duly computed. This scheme is illustrated in Figure 1. Note that what is being passed through the processors is only individuals (with their partial fitness values), not the data being mined. This minimizes interprocessor communication overhead.

The phase of application of genetic operators exploits function parallelism. Each processor applies genetic operators to the individuals of its local subpopulation. This phase requires no interprocessor communication.

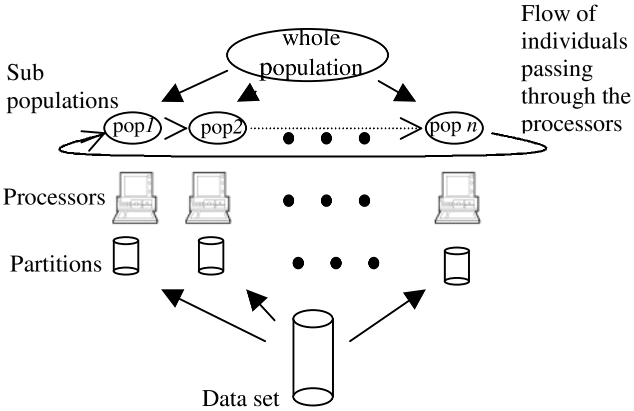

Fig. 1: Model of GA-PVMINER

3. RESULTS

We have evaluated GA-PVMINER on two public domain data sets available from the UCI repository of machine learning databases (*http://www.ics.uci.edu/AI/ML/Machine-Learning.html*), namely the nursery school and adult data sets.

The nursery school data set was derived from a hierarchical decision model originally developed to rank applications for nursery schools [11]. The final decision depended on three subproblems: occupation of parents and child's nursery, family structure and financial standing, and social and health picture of the family. This data set has 12960 instances and 9 attributes, all of them categorical. In our experiments we have specified 3 goal attributes for the nursery data set, namely Recommendation, Social and Finance (recall that the GA can discover rules predicting any of the goal attributes). Notice that Recommendation would be the only goal attribute in a classification task.

The adult data set contains 48844 data instances and 15 attributes, categorical and continuous. We have specified two goal attributes for the adult data set namely “Workclass” and “Class”. The latter would be the only goal attribute used in a classification task.

In all the experiments the genetic algorithm had 200 individuals in the population, and was run for 100 generations. These parameters values were sufficient to find some good individuals, but it has to be said that we have made no serious attempt to optimize these values. Tables 1 and 2 show the results of the experiments, which are discussed in the next section.

For the experiments on the exploitation of parallelism we have used a parallel virtual machine (PVM)[12] consisting of four 350-MHz Pentium II computers, each with 32 MB of memory and 6GB of disk, with operating system Linux RedHat 5.2 and PVM 3.3.11. The interconnection network was Ethernet with 10Mbps. In our system one of the four processors runs the master program, which controls the slave programs (each running a subpopulation). The processor running the master program also runs one slave program. However, this is not a serious problem, since in our system the processing time taken by the master program is very small, in comparison with the processing time taken by the slave programs (which perform all the evolution of individuals).

Our experiments have measured the speed of the parallel version of the algorithm over the sequential one. The speed up is defined as the ratio T_s / T_p, where T_s is the sequential processing time (on a single processor) and T_p is the parallel processing time. Tables 3 and 4 show the speed up results. In each of these tables, the parallel processing time shown in the fourth column is the sum of the processing time itself and the initialization time (required to distributed the data and individuals across the processors).

As shown in these tables, the parallel version achieved a reasonable speed up over the sequential version. As expected, the speed up was greater in the case of the adult data set. The reason is that this data set is larger than the nursery data set, so there is more opportunity for the exploration of data parallelism in the former. Of course, real-world databases can be much larger than the two public domain data sets used in our experiments. Therefore, we can expect that our system will achieve even higher speed ups in larger real-world data databases. This point deserves further investigation.

Table 1: Results for the nursery data set.

J1-measure	**Rule**	**Training Data**	**Test Data**
		|*C&P*| / |*C*|	**|*C&P*| / |*C*|**
0.0317	IF health = recommended THEN recomendation = priority	0.6184	0.6229
0.0916	IF health = priority AND has_nurs = very_crit THEN recomendation = spec_prior	0.9922	0.9790
0.0000	IF housing = convenient AND finance = inconv THEN recomendation = not_recom	0.3447	0.3067
0.0015	IF (recomendation = very_recom) THEN social = nonprob	0.4660	0.5700
0.0073	IF health = recommended AND recomendation = spec_prior THEN social = problematic	0.5307	0.5260
0.0029	IF (recomendation = very_recom) AND (health = recommended) THEN social = slightly_prob	0.5339	0.4299
0.0034	IF recomendation = very_recom AND housing = convenient THEN finance = convenient	0.7681	0.7428
0.0028	IF housing = convenient AND recomendation = spec_prior THEN finance = inconv	0.6446	0.6993
	average	**0.6124**	**0.6096**

Table 2: Results for the adult data set.

J1-measure	**Rule**	**Training Data** $\lvert C\&P\rvert / \lvert C\rvert$	**Test Data** $\lvert C\&P\rvert / \lvert C\rvert$
0.0152	IF native-country = United-States AND ocuppation = Farming-fishing THEN workclass = Self-emp-not-inc	0.4755	0.5000
0.0042	IF marital-status = Never-married AND class = <=50K THEN workclass = Private	0.7709	0.7505
0.0061	IF class = >50K AND sex = Male THEN workclass = Self-emp-inc	0.0872	0.0899
0.0465	IF marital-status = Never-married THEN class = <=50K	0.9540	0.9554
0.0416	IF sex = Male AND relationship = Husband THEN class = >50K	0.4486	0.4488
	average	**0,5472**	**0,5489**

Table 3: Speed up results for the nursery data set (12906 data instances)

Number of Processors	Number of sub populations	Sequential processing time	Parallel processing time	Speed up
1	1	134 sec.	-	-
3	3	402 sec.	252 sec.	1.595
4	4	536 sec.	320 sec.	1.675

Table 4: Speed up results for the adult data set (48,844 data instances)

Number of Processors	Number of sub Populations	Sequential processing time	Parallel processing time	Speed up
1	1	1052 sec.	-	-
3	3	3156 sec.	1364 sec.	2.313

4. DISCUSSION AND CONCLUSIONS

In a classification task, there are some well-defined measures of predictive accuracy. For instance, a commonly used measure is the accuracy rate, i.e., the ratio of the number of correctly classified test instances over the total number of test instances. Despite its popularity this measure has several defects, and other measures can (and probably should) be used [13].

The target of this work is the dependence modeling task which, as mentioned before, is a generalization of the classification task, where different rules can predict different attributes.

In both tasks the evaluation of the discovered rules must take into account their predictive accuracy on a separate test set. The difference is as follows. In classification we usually aim at discovering a rule set that can classify any test instance that appears in the future. Therefore it makes sense to compute an accuracy rate or related measure over all instances in the test set.

In dependence modeling, in the sense addressed in this paper, we do not aim to classify the whole test set. Rather, the goal is to discover a few interesting rules to be shown to the user. We can think of the discovered rules as the most valuable "knowledge nuggets" extracted from the data. These knowledge nuggets are valuable even if they do not cover the whole test set. In other words, the value of the discovered rules depends on their predictive accuracy on the part of the test set covered by those rules, but not on the test set as a whole. After all, there are several goal attributes, and it is not realistic to expect that the discovered rules can predict the value of all goal attributes for all instances in the test set. In fact, we could mine such a large rule set by running one classification algorithm for each goal attribute, but we would get too many rules, and the task being solved would be simply "multiple classification". In contrast, in the dependence modeling task addressed in this paper we aim at discovering a much smaller set of interesting rules.

Hence, it does not make much sense to evaluate the performance of the discovered rule set as a whole in test set, and the discovered rules are better evaluated on a rule-by-rule basis.

The value of the *J1-measure* for a rule is computed by equation [2]. For each discovered rule, the tables also show, in the last two columns, the "confidence factor" (CF) of the rule – i.e., $\lvert C\&P\rvert \,/\, \lvert C\rvert$ – in the training set and in the test set.

Comparing the two CFs for each rule we can evaluate the generalization performance of the discovered rules. In general, the CF in the test set is not significantly lower than the CF in the training set, as can be seen in the last row (with the average results) of the table. In particular, for the nursery data set (Table 1), out of the 8 discovered rules, 3 have a CF in the test set even higher than its CF in the training set.

Overall, the results show that the discovered rules do have a good generalization performance on the unseen test set. Recall that the CF of rule in the training set usually is an optimistic estimate of its CF in the test set. Hence, the J1-measure seems to be doing a good job of selecting rules with high predictive accuracy on the test set.

However, future work should include a more extensive set of experiments mining other data sets, to further validate the empirical results reported in this paper. In particular, experiments mining very large databases are necessary for better validating the parallelization strategy proposed in this paper.

In any case, note that our system offers not only an opportunity for the exploitation of data parallelism in very large databases but also an opportunity to perform a parallel search in the dependence modeling task, which has a very large search space (since we search for rules predicting different goal attributes).

5. REFERENCES

[1] Neri F. and Giordana A. – "A parallel genetic algorithm for concept learning". In *Eshelman L. J.* (editor), *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 436-443, Morgan Kaufmann,1995.

[2] Danyluk A. P. and Provost F.J. – "Small disjuncts in action: learning to diagnose errors in the local loop of telephone network". *Proceedings of the 10th International Conference on Machine-Learning*, pp. 81-88, 1993.

[3] Holte R., Acker L. and Porter B. – "Concept learning and the problem of small disjuncts". *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 813-818, Detroit, MI, 1989.

[4] Quinlan J. R. – "Improved estimates for the accuracy of small disjuncts". *Machine Learning*, vol. 6, pp. 93-98, 1991.

[5] DeJong K. A., Spears W. M. and Gordon F. D. – "Using genetic algorithms for concept learning". *Machine Learning*, vol. 13, pp. 161-188, 1993.

[6] Giordana A. and Saitta L. – "Learning disjunctive concepts by means of genetic algorithms". *Proceedings of the 11th International Conference on Machine Learning*, pp. 96-104, 1994.

[7] Greene D. P. and Smith S. F. – "Competition-based induction of decision models from examples". *Machine Learning*, vol. 13, pp. 229-258, 1993.

[8] Smyth P. and Goodman R. M. – "Rule induction using information theory". In Piatetsky-Shapiro G. and Frawley J. (editors), *Knowledge Discovery in Databases*, pp. 159-176, Cambridge: MIT press, 1991.

[9] Wang K., Tay S. H.W. and Liu B. – "Interestingness-based interval merger for numeric association rules". *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pp. 121-127, AAAI Press, 1998.

[10] Flockhart I. W. - *GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithms* – final report. *EPCC-AIKMS-GA-MINER-Report 1.0*, The University of Edinburgh, 1995.

[11] Bohanec M., Rajkovic V. – "Expert system for decision making". *Sistemica*, vol. 1, pp. 145-157, 1990.

[12] Geist A., Benguelin A., Dongarra J., Jiang W., Manchek R. and Sunderam V. – *PVM – Parallel Virtual Machine – A Users Guide and Tutorial for Networked Parallel Computing*., Cambridge: MIT Press, 1994.

[13] Hand D. - *Construction and Assessment of Classification Rules.* Chichester: John Wiley & Sons, 1997.