TASK QUARTERLY 6 No 3 (2002), 483–492

LEARNING DECISION RULES USING A DISTRIBUTED EVOLUTIONARY ALGORITHM

WOJCIECH KWEDLO AND MAREK KRĘTOWSKI

Department of Computer Science,
Technical University of Bialystok,
Wiejska 45a, 15-351 Bialystok, Poland
{wkwedlo,mkret}@ii.pb.bialystok.pl

(Received 20 April 2002)

Abstract: A new parallel method for learning decision rules from databases by using an evolutionary algorithm is proposed. We describe an implementation of EDRL-MD system in the cluster of multiprocessor machines connected by Fast Ethernet. Our approach consists in a distribution of the learning set into processors of the cluster. The evolutionary algorithm uses a master-slave model to compute the fitness function in parallel. The remainder of evolutionary algorithm is executed in the master node. The experimental results show, that for large datasets our approach is able to obtain a significant speed-up in comparison to a single processor version.

Keywords: decision rule learning, distributed evolutionary algorithms

1. Introduction

Data Mining and Knowledge Discovery in Databases (KDD) is a process of identifying valid, potentially useful and understandable patterns in data [1]. The two main goals of KDD are *prediction i.e.* the use of available data to predict unknown variables and *description i.e.* the search for some interesting patterns and their presentation in an easy to understand way. An important prediction task is *classification* of an example to one of predefined classes. Usually the *classifier* is created from the *learning set*, which contains description of some examples with known class labels.

One of the most well-known classification techniques used in KDD is discovery of decision rules from data. Many rule induction methods *e.g.* AQ-family [2], CN2 [3] or C4.5 [4] were proposed. The advantages of the rule-based approach include natural representation and ease of integration of learned rules with background knowledge.

It is a well-known fact, that most applications of KDD require the capability of efficient processing of large databases. Unfortunately data mining algorithms, which offer very good performance (*e.g.* classification accuracy), are computationally expensive. A possible solution to this problem is a parallel implementation of the given algorithm.

In the paper we present a new distributed approach to decision rules learning. Our method consists in re-implementation of the existing system EDRL-MD (Evolutionary Decision Rule Learner with Multivariate Discretization) [5] on a cluster of Symmetric Multi-Processing (SMP) machines. For communication between nodes of the cluster Message Passing Interface (MPI) [6] is used.

The system learns decision rules using an *evolutionary algorithm* [7] (EA). EAs are stochastic techniques which have been inspired by the process of biological evolution. Their advantage over greedy search methods is the ability to avoid local optima. Several EA-based systems, which learn decision rules in either propositional (*e.g.* GABIL [8], GIL [9], GA-MINER [10], EDRL [11]) or first order (*e.g.* REGAL [12, 13], SIAO1 [14]) form have been proposed.

The main advantage of EDRL-MD in comparison with other EA-based systems lies in dealing with continuous-valued attributes. Most decision rule systems employ univariate discretization methods, which search for threshold values for only one attribute at the same time. In contrast to them, EDRL-MD learns rules simultaneously searching for threshold values for all continuous-valued attributes. This approach is called [5] *multivariate discretization*.

The remainder of the paper is organized as follows. The next section presents EDRL-MD system. The distributed implementation of the system in a parallel environment is described in Section 3. Section 4 is devoted to presentation of the results of computational experiments investigating scalability of our approach. The last section contains the conclusion and possible direction of future work.

2. Learning decision rules with EDRL-MD

In this section we briefly present the main topics (*i.e.* representation of solutions and genetic search operators) of the learning system EDRL-MD. More detailed description can be found in [5].

We assume that a learning set $E=\{e_1,e_2,\dots,e_M\}$ consists of M examples. Each example $e\in E$ is described by N attributes (features) $A_1,A_2,\dots,A_N$ and labeled by a class $c(e)\in C$. The domain of a nominal (discrete-valued) attribute A_i is a finite set $V(A_i)$, while the domain of a continuous-valued attribute A_j is an interval $V(A_j)=[l_j,u_j]$. For each class $c_k\in C$ by $E^+(c_k)=\{e\in E: c(e)=c_k\}$ we denote the set of *positive examples* and by $E^-(c_k)=E-E^+(c_k)$ the set of *negative examples*. A *decision rule* R takes the form IF $t_1\wedge t_2\wedge\dots\wedge t_r$ THEN c_k, where $c_k\in C$ and the left-hand side (LHS) is a conjunction of r $(r\le N)$ conditions $t_1,t_2,\dots,t_r$; each of them concerns one attribute. The right-hand side (RHS) of the rule determines class membership of an example. A *ruleset* RS is a disjunctive set of decision rules with the same RHS. By $c_{RS}\in C$ we denote the class on the right-hand side of the ruleset RS.

In our approach the EA is called once for each class $c_k\in C$ to find the ruleset separating the set of positive examples $E^+(c_k)$ from the set of negative examples $E^-(c_k)$. The search criterion, in terminology of EAs called the *fitness function* prefers rulesets consisting of few conditions, which cover many positive examples and very few negative ones.

2.1. Representation

The EA processes a population of candidate solutions to a search problem called *chromosomes*. In our case a single chromosome encodes a ruleset RS. Since the number of rules in the optimal ruleset for a given class is not known, we use variable-length chromosomes and provide the search operators which change the number of rules. The chromosome representing the ruleset is a concatenation of *strings*. Each fixed-length string represents the LHS of one decision rule. Because the EA is called to find a ruleset for the given class c_{RS}, there is no need for encoding the RHS.

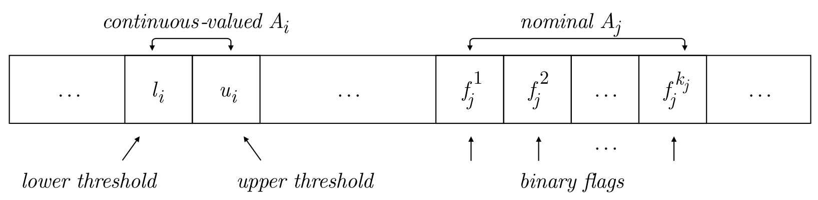

Figure 1. The string encoding the LHS of a decision rule ($k_j = |V(A_j)|$). The chromosome representing the ruleset is the concatenation of strings. The number of strings in a chromosome can be adjusted by some search operators

The string is composed (Figure 1) of N *substrings*. Each substring encodes a condition related to one attribute. The LHS is the conjunction of these conditions. In case of a continuous-valued attribute A_i the substring encodes the lower l_i and the upper u_i threshold of the condition $l_i < A_i \leq u_i$. It is possible that $l_i = -\infty$ or $u_i = +\infty$.

For a nominal attribute A_j the substring consists of binary flags. Each of the flags corresponds to one value of the attribute.

Note that it is possible that a condition related to an attribute is not present on the LHS. For a continuous-valued attribute A_i it can be achieved by setting both $l_i = -\infty$ and $u_i = +\infty$. For a nominal A_j it is necessary to set all the flags $f_j^1, f_j^2, \ldots, f_j^{|V(A_j)|}$.

(a)

Salary		*Amount*		*Purpose*		
				Car	*House*	*School*
$-\infty$	$+\infty$	$-\infty$	250	1	1	1
100	250	$-\infty$	500	1	1	1
750	$+\infty$	$-\infty$	$+\infty$	1	1	0

(b)

IF (*Amount* $<$ 250)
THEN *Accept*

IF (100 $<$ *Salary* $<$ 250) & (*Amount* $<$ 500)
THEN *Accept*

IF (*Salary* $>$ 750) & (*Purpose* $=$ *car or house*)
THEN *Accept*

Figure 2. Representation of rulesets: (a) an example chromosome consisting of three strings, (b) the corresponding ruleset

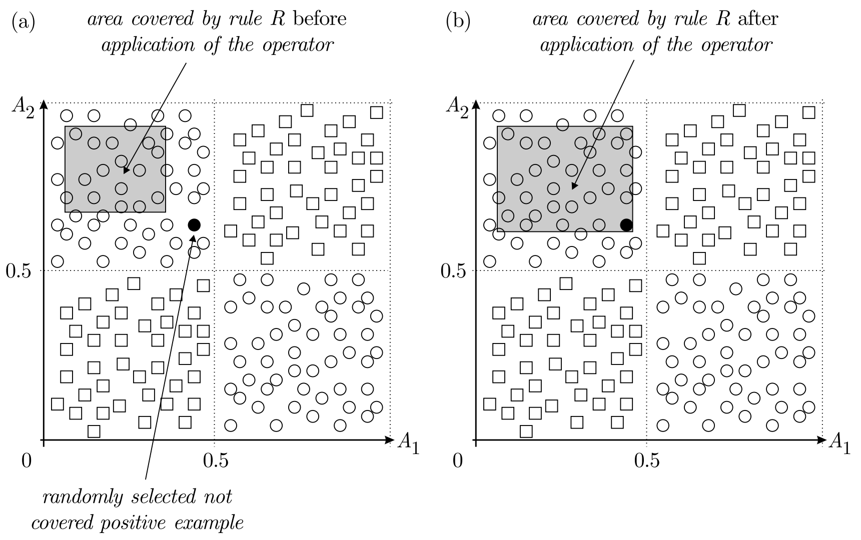
(a)
area covered by rule R before application of the operator
(b)
area covered by rule R after application of the operator
A_2
A_1
0
0.5
0.5
randomly selected not covered positive example

Figure 3. Positive example insertion operator

Figure 2 shows an example chromosome for a dataset with two numerical attributes: *Salary* and *Amount*, and one nominal attribute *Purpose*. It is assumed that the EA is searching for the optimal ruleset for the class *Accept*.

2.2. Genetic operators

EDRL-MD employs six search operators. Four of them: *changing condition*, *positive example insertion*, *negative example removal*, *rule drop* are applied to a single ruleset *RS* (represented by a chromosome). The other two: *crossover* and *rule copy* require two arguments RS_1 and RS_2.

The changing condition is a mutation-like operator, which alters a single condition related to an attribute A_i. If A_i is nominal, a flag randomly chosen from $f_i^1, f_i^2, \ldots, f_i^{|V(A_i)|}$ is flipped. For a continuous-valued attribute a threshold (l_i or u_i) is replaced by a random boundary threshold.

The positive example insertion operator modifies a single decision rule R in the ruleset RS to allow it to cover a new random positive example $e^+ \in E^+(c_{RS})$, currently uncovered by R (Figure 3). All conditions in the rule, which conflict with e^+, have to be altered. In case of a condition related to a nominal attribute A_i the flag, which corresponds to $A_i(e^+)$, is set. If A_i is a continuous-valued attribute and the condition $l_i < A_i \le u_i$ is not satisfied because $u_i < A_i(e^+)$ the threshold u_i is replaced by $\hat{u}_i$, where $\hat{u}_i$ is the smallest boundary threshold such that $\hat{u}_i \ge A_i(e^+)$. The case when $A_i(e^+) \le l_i$ is handled in a similar way.

The negative example removal operator alters a single rule R from the ruleset RS. It selects at random a negative example e^- from the set of all the negative examples covered by R (Figure 4). Then it alters a random condition in R in such a way, that the modified rule does not cover e^-. If the chosen condition concerns a nominal attribute A_i the flag which corresponds to $A_i(e^-)$ is cleared. If A_i is a continuous-valued attribute then the condition $l_i < A_i \le u_i$ is narrowed down either to $\hat{l}_i < A_i \le u_i$ or to $l_i < A_i \le \hat{u}_i$, where $\hat{l}_i$ is the smallest boundary threshold such that $A_i(e^-) \le \hat{l}_i$ and $\hat{u}_i$ is the largest boundary threshold such that $\hat{u}_i < A_i(e^-)$.

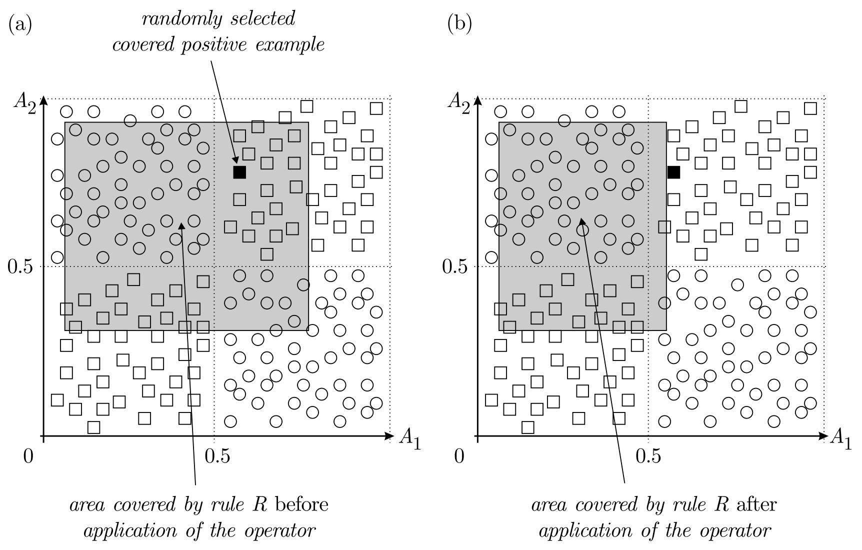
(a)
randomly selected covered positive example
A_2
0.5
0
0.5
A_1
area covered by rule R before *application of the operator*
(b)
A_2
0.5
0
0.5
A_1
area covered by rule R after *application of the operator*

Figure 4. Negative example removal operator

Rule drop and rule copy operators are the only ones capable of changing the number of rules in a ruleset. The single argument rule drop removes a random rule from a ruleset RS. The two argument rule copy adds to one of its arguments RS_1, a copy of a rule selected at random from RS_2, provided that the number of rules in RS_1 is lower than $\max_R$. $\max_R$ is an user-supplied parameter, which limits the maximal number of rules in the ruleset.

The crossover operator selects at random two rules R_1 and R_2 from the respective arguments RS_1 and RS_2. Then it applies a uniform crossover [7] to the strings representing R_1 and R_2.

2.3. The fitness function

Consider a ruleset RS which covers pos positive examples and neg negative ones. The number of positive and negative examples in the learning set is denoted by POS and NEG, respectively. The ruleset RS classifies correctly pos positive examples and $NEG-neg$ negative ones. Hence the probability of classifying correctly an example from the learning set is given by:

$$Pr(RS)=\frac{pos+NEG-neg}{POS+NEG}. \tag{1}$$

Obviously we are interested in rulesets with possibly high classification accuracy. However, optimizing directly Equation (1) will yield disastrous results. For instance, it is trivial to find a ruleset consisting of POS rules, that each of them covers one positive example and no negative ones. Although for that ruleset $Pr(RS)=1.0$, it will not be able to classify correctly unknown examples because it is too specialized and *overfits* the learning data.

To avoid the overfitting we take two steps. Firstly, we limit the number of rules in a chromosome to $\max_R$, where $\max_R$ is a user-supplied parameter. Secondly, our fitness function depends not only on the probability (1), but also on the complexity of the ruleset. As a measure of complexity we take:

$$Compl(RS) = \alpha \log_{10}(L+1)+1, \tag{2}$$

where L is the total number of conditions in the ruleset RS and α is a user supplied parameter. Finally, the fitness function is given by:

$$f(RS) = \frac{Pr(RS)}{Compl(RS)}. \tag{3}$$

3. Implementation in a distributed environment

As Equation (1) shows, to determine the fitness of a chromosome it is necessary to calculate the counts of positive and negative examples denoted by *pos* and *neg*, respectively. To obtain *pos* and *neg* the algorithm has to iterate through all the examples in the learning set. For each example $e_i \in E$ the algorithm checks if e_i is covered by the ruleset RS. If the example matches a premise of at least one rule from the RS it is regarded as covered. Then, depending on the type of the example either the counter of positive examples or the counter of negative examples is incremented. This step is shown on Figure 5.

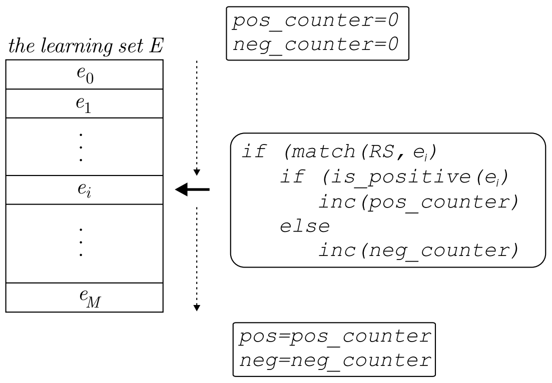

Figure 5. Computation of the fitness of the ruleset RS

In many practical applications the size of the learning set is very large. Moreover, the CPU time required by the remaining components of the EA *i.e.* genetic search operators and selection does not depend on the size of the learning set. Consequently, the computational complexity of the algorithm is dominated by the calculation of the fitness.

In our approach we decided to implement the computation of fitness function in a distributed manner. The algorithm runs in a *master-slave* model. The dataset is divided evenly into subsets; each subset is placed on a single slave processor (see Figure 6). Each slave processor is responsible for the evaluation of the rules on the corresponding subset. At the beginning of an iteration of EA the master processor broadcasts (Figure 7a) the population *i.e.* the set of chromosomes to all slave processors. For each ruleset in the population a slave processor counts the number of covered positive and negative examples from its subset. The master processor gathers

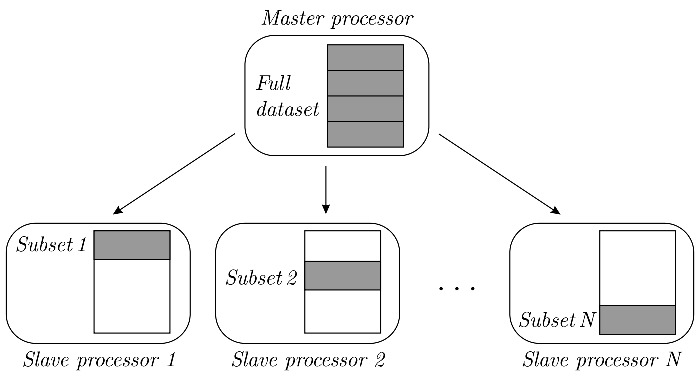

Figure 6. Initialization of the algorithm

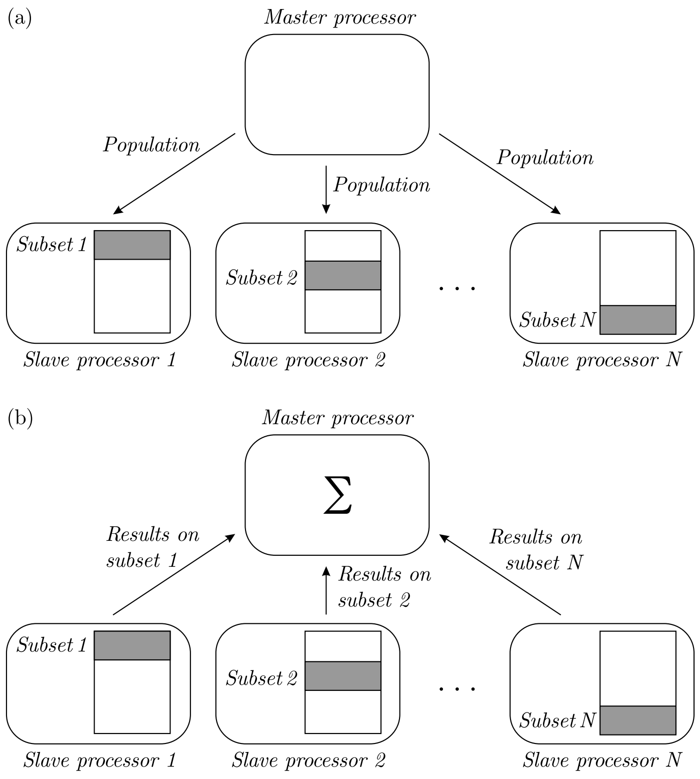

Figure 7. Computation of the fitness function

(Figure 7b) results from each slave, sums up the counts for each ruleset and computes the fitness.

In addition to the number of positive and negative examples each slave chooses randomly one positive example not covered by the ruleset and one negative example covered by the ruleset. The numbers of these examples are also sent to master, because they are necessary to apply positive example insertion and negative example removal operators. The data concerning all the rulesets is packed by the slave in one message, which is sent using a single MPI_Gather operation.

When the fitness of all chromosomes is computed the remaining part of the iteration of EA (*i.e.* the selection and genetic operators) is executed solely on the master processor. For each ruleset the master chooses two examples needed by the insertion and removal operators randomly from the candidates sent by slaves.

4. Experimental results

In this section experimental results are presented. We have tested the parallel version of EDRL-MD on five datasets of varying size. The datasets were taken from the repository of publicly available data at the University of California, Irvine [15]. The description of the datasets is shown in Table 1.

Table 1. The datasets used in the experiments

Dataset	Size	No. of attributes (Numeric/Nominal)	No. of classes
german	1000	7/13	2
cmc	1473	2/7	3
page blocks	5473	10/0	5
nursery	12960	0/8	5
thyroid	7200	6/15	3

To evaluate the proposed approach we performed an experiment on a 6 CPU cluster consisting of three two-processor machines (Pentium III 700) running Linux. The machines in the cluster were connected via a Fast Ethernet switch. For message passing LAM (Local Area Multicomputer) MPI [16] implementation was used.

The total processing time of the algorithm is proportional to the number of generations, which in turn depends on the termination condition. In our approach algorithm stops when the fitness of the best chromosome does not improve during consecutive N_{TERM} generations. Because the EA is a probabilistic algorithm, the number of generations and the processing time can vary significantly with different runs on the same data. However, an average time of a *single generation* is approximately the same for the given learning set. This time was used to evaluate scalability of our method.

Figure 8 shows average time of generation depending on the number of slave CPU processors. Note that in case when the number of slaves is equal to the number of CPUs one processor acts both as slave and master.

Figure 9 shows the speed-up obtained by our implementation for all datasets compared to the ideal case of linear speed-up.

The results show that for two smaller datasets (*german* and *cmc*) the speed-up is relatively small. The most probable reason for such result is very high latency of our Fast Ethernet interconnect. However, for the larger dataset the algorithm is able to obtain a significant, almost linear speed-up.

5. Conclusions and future work

In this paper we have shown, that the computational efficiency of evolutionary algorithms for data mining applications can be significantly improved by the use

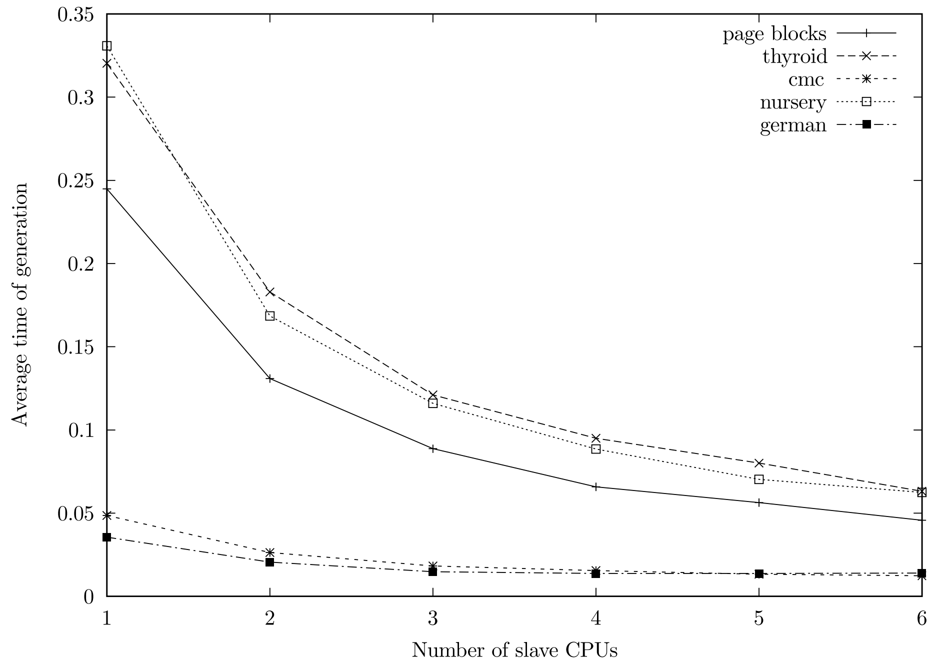

Figure 8. Average time of EA generation for five datasets

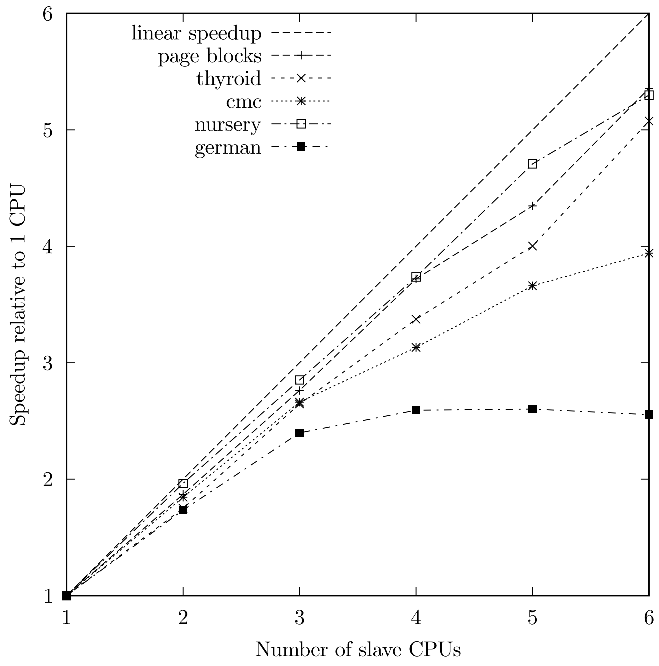

Figure 9. Speedup relative to one slave CPU

of parallel machines. We proposed a parallel version of EDRL-MD based on data distribution approach. The experimental results suggest that for large datasets near-linear speed-up is possible.

Several directions of future research exist. One of them is the load balancing in heterogeneous clusters, which could be easily achieved by non-even division of the learning set.

Especially for smaller datasets our results were negatively affected by the high latency of Fast Ethernet used as the interconnect. However, the latency could be reduced almost by two orders of magnitude, by using more modern interconnect (*e.g.* Dolphin or Myrinet). In such case it would be possible to achieve higher than linear speedup for some datasets because the distributed learning set would more efficiently utilize the cache memory of slave processors. We are going to test this hypothesis using hardware with a low-latency interconnect.

Acknowledgements

The authors are grateful to Prof. Leon Bobrowski for his support and useful comments. This work was supported by the grant W/WI/1/02 from Technical University of Bialystok.

References

[1] Fayyad U M, Piatetsky-Shapiro G, Smyth P and Uthurusamy R (Eds.) 1996 *Advances in Knowledge Discovery and Data Mining*, AAAI Press

[2] Michalski R S, Mozetic I, Hong J and Lavrac N 1986 *Proc. 5th Nat. Conf. on Artificial Intelligence*, Philadelphia, PA, pp. 1041–1045

[3] Clark P and Niblett T 1989 *Machine Learning* **3** 261

[4] Quinlan J R 1993 *C4.5: Programs for Machine Learning*, Morgan Kaufmann

[5] Kwedlo W and Krętowski M 1999 *Springer Lecture Notes in Computer Science* **1704** 392

[6] Snir M (Ed.) 1998 *MPI – The Complete Reference*, 2nd Edn., MIT Press

[7] Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd Edn., Springer

[8] De Jong K, Spears W M and Gordon D F 1993 *Machine Learning* **13** 168

[9] Janikow C 1993 *Machine Learning* **13** 192

[10] Flockhart I W and Radcliffe N J 1996 *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, KDD-96*, Portland, OR, AAAI Press, pp. 299–302

[11] Kwedlo W and Krętowski M 1998 *Springer Lecture Notes in Computer Science* **1510** 370

[12] Giordana A and Neri F 1995 *Evolutionary Computation* **3** (4) 375

[13] Neri F and Saitta L 1996 *IEEE Trans. on Pattern Analysis and Machine Intelligence* **18** 1135

[14] Augier S, Venturini G and Kodratoff Y 1995 *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, KDD-95*, Montreal, Canada, AAAI Press, pp. 21–26

[15] Murphy P and Aha D *UCI Repository of Machine-learning Databases*, http://www.ics.uci.edu/pub/machine-learning-databases

[16] Burns G, Daoud R and Vaigl J 1994 *Proc. of Supercomputing Symposium '94*, University of Toronto, pp. 379–386