A Parallel Evolutionary Algorithm for Discovery of Decision Rules

Wojciech Kwedlo

Faculty of Computer Science
Technical University of Białystok
Wiejska 45a, 15-351 Białystok, Poland
wkwedlo@ii.pb.bialystok.pl

Abstract. In the paper a new parallel method for learning decision rules is proposed. The method uses evolutionary algorithm to discover decision rules from datasets. We describe a parallelization of the algorithm based on master-slave model. In our approach the dataset is distributed among slave processors of a parallel system. The slave procesors compute fitness function of chromosomes in parallel. The remainder of evolutionary algorithm i.e. selection and genetic search operators is executed by the master processor. Our method was implemented on a cluster of SMP machines connected by Fast Ethernet. The experimental results show, that for large datasets it is possible to obtain a significant speedup.

1 Introduction

One of the most well-known classification techniques used in data mining is discovery of decision rules from data. The advantages of the rule-based approach include natural representation and ease of integration of learned rules with background knowledge.

Evolutionary algorithms (EAs) [5] are stochastic optimization 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 were proposed [1, 3]. The solutions obtained by those systems are often better than the solutions obtained by traditional methods. However, the main disadvantage of EAs is their high computational complexity. In many real-life applications of data mining the size of analyzed dataset is very large. Is such cases the big computational complexity of EAs makes their use extremely difficult. A possible solution of this problem is a parallel implementation of the given algorithm.

In the paper we describe a parallel implementation of the system EDRL-MD (Evolutionary Decision Rule Learner with Multivariate Discretization) [3]. The main advantage of EDRL-MD in comparison with other EA-based systems is the capability of direct extraction of rules from datasets with continuous-valued attributes. The other systems require prior *discretization* of such attributes.

The reminder of the paper is organized as follows. The next section presents EDRL-MD system. The parallel formulation of the system 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 conclusions.

2 Description of the system EDRL-MD

In this section we present two main topics, i.e. representation of solutions by chromosomes and the fitness function, which are most important for parallel formulation of the algorithm. Description of the remaining components, e.g. genetic operators, can be found in [3] or [4].

2.1 Basic notions

We assume that a learning set $E = \{e_1, e_2, \ldots, e_M\}$ consists of M examples. Each example $e \in E$ is described by N attributes (features) $A_1, A_2, \ldots, A_N$ and labelled 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 $t_1 \wedge t_2 \wedge \ldots \wedge t_r$ THEN c_k, where $c_k \in C$ and the left-hand side (LHS) is a conjunction of $r(r \leq N)$ conditions $t_1, t_2, \ldots, t_r$; 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.

2.2 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.

The string is composed (Fig. 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

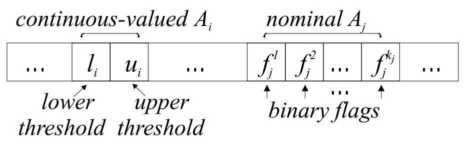

Fig. 1. The string encoding the LHS of a decision rule ($k_j = |V(A_j)|$). The chromosome representing the ruleset is the concatenation of strings. The number of strings in a chromosome can be adjusted by some search operators.

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.

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:

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

The fitness function is defined as:

$$f(RS) = \frac{Pr(RS)}{Compl(RS)}, \tag{2}$$

where $Compl(RS)$ is the complexity of the ruleset. As a measure of complexity we take:

$$Compl(RS) = \alpha log_{10}(L + 1) + 1, \tag{3}$$

where L is total the number of conditions in the ruleset RS and α is a user supplied parameter.

3 Implementation in a parallel system

The main loop of evolutionary algorithm begins with computation of fitness function of all S individuals in the population. In the next step, called *selection* [5], a new population is created by multiple random choice of chromosomes with high fitness from the old population. After selection, some genetic operators like *mutation* and *crossover* are applied. The algorithm iterates these three steps until a termination condition is met.

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. In many practical applications M, i.e. the size of the learning set is very large. Moreover, the CPU time required by remaining components of the EA i.e. genetic operators and selection does not depend on the size of the learning set. (it depends on size of the population S instead). In almost all cases $S << M$. For instance in our experiments we always set $S = 50$, whereas for the smallest dataset the size $M = 2310$. Consequently the the computational complexity of the algorithm is dominated by the calculation of the fitness.

For these reasons in our parallel implementation we decided to focus on computation of the fitness. Selection and genetic operators are executed sequentially. The algorithm runs in a *master–slave* model. The dataset is divided evenly into subsets; each subset is placed on a single slave processor. Each slave processor is responsible for the evaluation of the rules on the corresponding subset. At the begin of an iteration of EA the master processor broadcasts (Fig. 2a) the population i.e., the set of S 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 (Fig. 2b) results from each slave, sums up the counts for each ruleset and computes the fitness.

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.

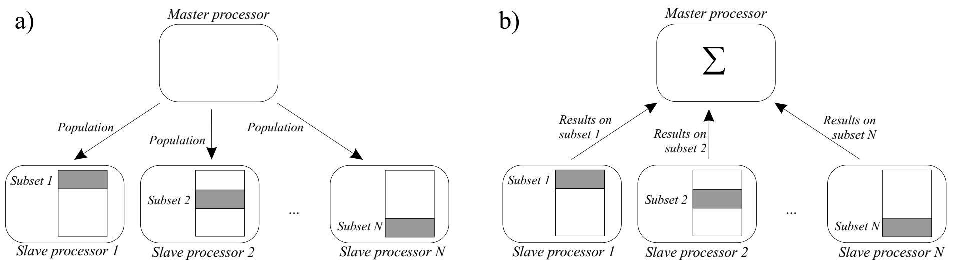

Fig. 2. Computation of the fitness function.

4 Experimental results

In this section experimental results are presented. We have tested the parallel version of EDRL-MD on six datasets of varying size. The datasets were taken from the repository of publicly available data at University of California, Irvine [6]. The description of the datasets is shown in Table 1.

Dataset name	No. of records	No. of attributes	No. of classes
page	5473	10	5
segmentation	2310	19	7
hypothyroid	3773	21	3
cmc	1473	9	3
shuttle	43500	9	7
satimage	4435	36	6

Table 1. The datasets used in the experiments

To evaluate the proposed approach we performed an experiment on a 8 CPU cluster consisting of one four-processor and two two-processor machines (4 * Pentium III Xeon 700 + 2 * 2 * Pentium III 750) running Linux 2.4.18. Machines in the cluster were connected via a Intel 440T Fast Ethernet switch. For message passing MPICH [2] MPI implementation was used. We used the default configuration of MPICH.

Usually, to compute a speedup or efficiency of an parallel algorithm the total processing time is used. In our case this time is proportional to the number of iterations, which in turn depends on the termination condition. The algorithm terminates when the fitness of the best chromosome does not improve during consecutive N_{TERM} generations. Because the EA is a probabilistic algorithm, number of iterations and the processing time can significantly vary with different runs on the same data. However the time of a single iteration is approximately constant for the given learning set. Therefore an average time of a single iteration was used to compute speedup of our method. To compute the average time of an iteration we divided total processing time by the number of iterations.

Figure 3 shows the speedup obtained by our implementation for all datasets. For the largest data set (shuttle) we were able to achieve speedup greater than 7.2 for eight processors.

5 Conclusions

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

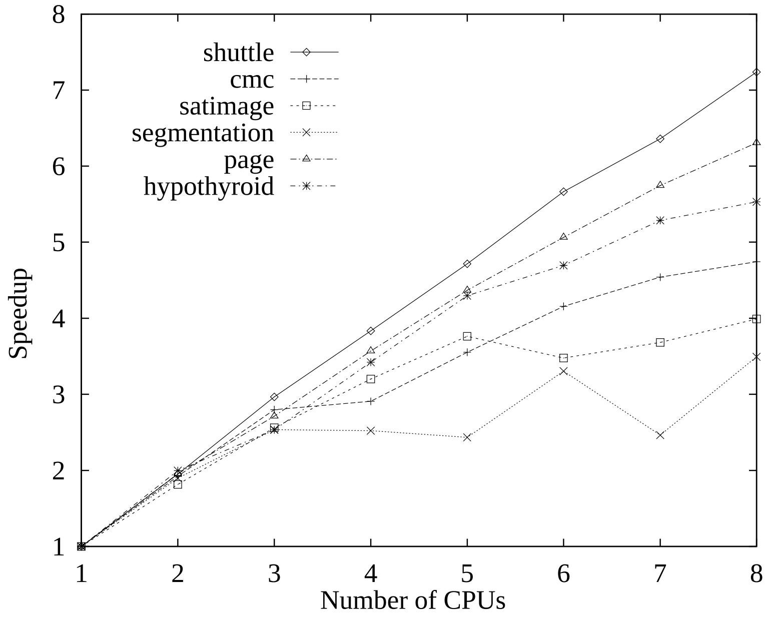

Fig. 3. Speedup obtained for different datasets.

References

1. Janikow, C.: A knowledge intensive genetic algorithm for supervised learning. *Machine Learning* 13 (1993) 192–228.
2. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing standard. *Parallel Computing*, 22 (1996) 789–828.
3. Kwedlo, W. Kretowski, M.: An evolutionary algorithm using multivariate discretization for decision rule induction. *3rd European Conference on Principles and Practice of Knowledge Discovery in Databases. Lecture Notes in Artificial Intelligence 1704*, (1999) 392–397, Springer Verlag.
4. Kwedlo, W. Kretowski, M.: Learning decision rules using a distributed evolutionary algorithm. *TASK Quarterly*, 6 (2002) 483–492.
5. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. 3^{rd} edn. Springer Verlag (1996).
6. Murphy, P., Aha, D.: *UCI repository of machine-learning databases*, available online: http://www.ics.uci.edu/pub/machine-learning-databases.