Abstract: In this paper, a novel rule discovery system that utilizes the Ant Colony Optimization (ACO) is presented. The ACO is a metaheuristic inspired by the behavior of real ants, where they search optimal solutions by considering both local heuristic and previous knowledge, observed by pheromone changes. In this paper we study a parallel implementation of the Ant-Miner algorithm developed by Parpinelli, Lopes and Freitas for rule induction. The main idea in this paper is to speed up the searching process and diminish the standard deviations obtained in the sequential version of this algorithm. This algorithm is suited for parallelization in the multi-ants population contrary to the standard proposition, when the performance of single ant is observed. The comparative study will be carried out using the 4 data sets from the UCI Machine Learning repository. Parallelization of Ant-Miner demonstrates the improvement of the obtained results. It demonstrates also that high-performance computing of this approach is feasible to analyze large data sets from Machine Learning repository.

Keywords: Parallel Ant-Miner, Ant Colony Optimization, Data Mining, Classification rule, OpenMP.

1 Introduction

Ant Colony Optimization (ACO) is a metaheuristic approach to solve many different optimization problems by using principles of communicative behavior observed in real ant colonies. Ants can communicate with each other about paths they traversed by reinforcement mechanisms of pheromone trails laid on the appropriate edges. The pheromone trails can lead other ants to food sources. ACO was introduced in [6]. It is a population-based approach, where several generations of virtual ants search for good solutions. The following ants of the next generation are attracted by the pheromone changes so that they can search in the solution space near attractive solutions, concerning specific sub-spaces.

Ant-based approaches are good candidates for parallelization but not sufficient research has been done in the parallel version of ACO in the field of data mining so far. It seems quite easy how to parallelize them. Every processor can hold an ant or a colony of ants and after every generation/iteration number, the ants/colonies exchange information about their solutions. The following ant colony algorithms follow this scheme and differ only in granularity and whether the computations for the updating pheromone information are done locally or globally by a master processor which distributes the new value of pheromone trails to the processors.

Recently some possible parallelization strategies for ACO have been proposed and classified into fine-grained and coarse-grained strategies [1]. In fine-grained parallelization strategies usually several artificial ants (or a simple ant) of a colony are assigned to each processor, therefore frequent information exchange between the small sub-colonies of ants (or between processors) takes place. Coarse-grained parallelization schemes run several colonies in parallel. This strategy is also referred to as a multi colony approach. The information exchange among colonies is done at certain intervals (number of iterations is established). Many parallel ACO applications to the combinatorial optimization problems have been analyzed [16, 10, 15, 1, 2, 11, 9, 17]. In [16], parallel MMAS with $k$ independent runs was studied. The experiment performed using TSP instances showed a clear advantage of parallel independent runs both in solution quality and computation time.

The most commonly used approach to parallelization is the island model where multiple colonies exchange information concerning solution, pheromone matrix, parameters. It can be done synchronously or asynchronously. In [1], it is reported that communication of the whole pheromone matrix leads to a decreased solution quality as well as a worse run-time. However, the exchanges of the best-so-far solutions approach leads to a good solution quality.

In [9], a scheme in which the whole pheromone matrix is shared with two colonies using symmetric multi processing (SMP) is studied on TSP instances. Unfortunately, the results presented in this approach showed no clear advantage of the parallel ACO algorithm over traditional ACO approach. In [10], parallel MMAS...
Data mining is a process of extracting useful knowledge from real-world data. Among several data mining tasks – such as clustering and classification – this paper focuses on classification. The aim of the classification algorithm is to discover a set of classification rules. One algorithm for solving this task is Ant-Miner, proposed by Parpinelli and colleagues [13], which employs ant colony optimization techniques [5, 7] to discover classification rules. Ant Colony Optimization is a branch of a newly developed form of artificial intelligence called swarm intelligence. Swarm intelligence is a form of emergent collective intelligence of groups of simple individuals: ants, termites or bees in which a form of indirect communication via pheromone was observed. Pheromone values encourage the ants following the path to build good solutions of the analyzed problem and the learning process occurring in this situation is called positive feedback or auto catalysis.

Outline. This article is organized as follows. Section 1 comprises an introduction to the subject of this article. In section 2, Ant Colony Optimization in Rule Induction is presented. Section 3 describes various schemes of the parallel implementation of Ant-Miner approach. In section 4 our proposed modifications are shown. Then the computational results performed in different tests are reported and then the empirical analysis is given. Finally, we conclude with general remarks on this work and further directions for future research are pointed out.

2 Ant Colony Optimization in Rule Induction

Ant-Miner was invented by Parpinelli et al. [13, 12]. It was the first Ant algorithm for rule induction and it has been shown to be robust and comparable with CN2 [3] and C4.5 [14] algorithms for classification. Ant-Miner generates solutions in the form of classification rules. Original Ant–Miner has a limitation i.e. it can only process discrete values of attributes.

The adaptation of ant colony optimization to rule induction and classification is a research area still not well explored and examined. The appeal of this approach similarly to the evolutionary techniques is that they provide an effective mechanism for conducting a more global search. These approaches have been based on a collection of attribute-value terms, then it can be expected that these approaches will also cope better with attribute interaction than greedy induction algorithms [8]. What is more, these applications require minimum understanding of the problem domain; the main components are: the heuristic function and the evaluation function, both of which may be employed in ACO approach in the same shapes as in the existing literature, concerning deterministic rule induction algorithms.

Ant-Miner is an ant-based system and it is more flexible and robust than traditional approaches. This method incorporates a simple ant system in which a heuristic value based on the entropy measure is calculated. Ant-Miner has produced good results when compared with more conventional data mining algorithms, such as C4.5 [14], ID3 and CN2 [3, 4], and it is still a relatively recent algorithm, which motivates us to try to amend it. This work proposes some modifications to the Ant-Miner to improve it. In the original Ant-Miner, the goal of the algorithm was to produce an ordered list of rules, which was then applied to the test data in order in which they were discovered. The original Ant-Miner was compared to CN2 [3, 4], a classification rule discovery algorithm uses a strategy for generating rule sets similar to that of heuristic function used in the main rule of ants’ strategy in Ant-Miner. The comparison was done using 6 data sets from the UCI Machine Learning repository that is accessible at www.ics.uci.edu/~mlearn/MLRepository.html. The results were analyzed according to the predictive accuracy of the rule sets and the simplicity of the discovered rule set, which is measured by the number of terms per rule. While Ant-Miner had a better predictive accuracy than CN2 in 4 of the data sets and a worse one in only one of the data sets, the most interesting result is that Ant-Miner returned much simpler rules than CN2. Similar conclusions could also be drawn from a comparison of Ant-Miner to C4.5, a well-known decision tree algorithm [14].

There are many other characteristics of ACO which are really important in data mining applications. ACO contrary to deterministic decision trees or rule induction algorithms during rule induction, tries to extenuate this problem of premature convergence to local optima because of stochastic element which prefers a global search in the problem’s search space. Secondly, ACO metaheuristics is a population–based one. It permits the system to search in many independently determined points in the search space concurrently and to use the positive feedback between ants as a search mechanism [12].

Ant–Miner algorithm (alg. 1) builds a set of classification rules, based on a training set. A stop condition for the algorithm is the maximum number of uncovered objects, as long as this parameter is not reached, the construction of subsequent rules is carried out (lines 2–26). An object is covered, if the values of its attributes
are the same as the corresponding values in the rule.

Induction of decision rules is preceded by an initialization of the pheromone trail and calculation of the information function. In the next stage all ants create (induce) a set of rules (one per ant) from which the best one is chosen and used to update pheromone trail matrix (lines 6–22). This process continues as long as different rules are constructed or a certain number of iteration was performed.

The most time consuming part of the algorithm is the process of rule induction by agents, therefore it was selected for parallelization (lines 8–18). Values of heuristic and probability functions are calculated for each possible attribute–value pairs (called terms) and next some of them are selected to construct a rule. Then the decision part of the rule is determined and evaluation of the rule quality is performed, followed by a prune, to eliminate redundant components.

After the best rule was selected among the constructed rules, each object which matches the rule is removed from the list of uncovered objects (line 25).

All cells in the pheromone table are initialized equally to the following value:

$$
\tau_{ij}(t = 0) = \frac{1}{\sum_{i=1}^{a} b_i}
$$

where:

- $a$ – the total number of attributes,
- $b_i$ – the number of values in the domain of attribute $i$.

The probability is calculated for all of the attribute–value pairs, and the one with the highest probability is added to the rule. The transition rule in Ant-Miner is given by the following equation:

$$
p_{ij} = \frac{\tau_{ij}(t) \cdot \eta_{ij}}{\sum_{i} \sum_{j} \tau_{ij}(t) \cdot \eta_{ij}}, \forall i \in I
$$

where:

- $\eta_{ij}$ is a problem-dependent heuristic value for each term,
- $\tau_{ij}$ is the amount of pheromone currently available at time $t$ on the connection between attribute $i$ and value $j$,
- $I$ is the set of attributes that are not yet used by the ant,
- parameter $\beta$ is equal to 1.

In Ant-Miner, the heuristic value is an information theoretic measure for the quality of the term to be added to the rule. For preferring the quality is measured in terms of the entropy this considered pair to the others, and the measure is given as follows:

$$
\eta_{ij} = \frac{\log_2(k) - \text{Info}_{T_{ij}}}{\sum_{i} \sum_{j} (\log_2(k) - \text{Info}_{T_{ij}})}
$$

where the function $\text{Info}$ is similar to another function employed in C4.5 approach:

$$
\text{Info}_{T_{ij}} = -\sum_{w=1}^{k} \left[ \frac{\text{freq}_{T_{ij}^w}}{T_{ij}} \right] \log_2 \left[ \frac{\text{freq}_{T_{ij}^w}}{|T_{ij}|} \right]
$$

where $k$ is the number of classes, $|T_{ij}|$ is the total number of cases in the partition $T_{ij}$ (the partition containing the cases, where the attribute $A_i$ has the value $V_{ij}$), $\text{freq}_{T_{ij}^w}$ is the number of cases in partition $T_{ij}$ with a class $w$, $b_i$ is a number of values in the domain of attribute $A_i$ ($a$ is the total number of attributes). The higher the value of $\text{Info}_{T_{ij}}$ is, the less likely is that the ant will choose term $T_{ij}$ to add to its partial rule. Please note that this heuristic function is a local method and it is sensitive to the attribute interaction. The pheromone values assigned to the term have a more global nature. The pheromone updates depend on the evaluation of a rule as a whole, i.e. we must take into account interaction among attributes appearing in the rule. The heuristic function employed here comes from the decision tree world and it is similar to the method used in algorithm C4.5. There are many other heuristic functions that may be adapted and used in Ant-Miner for example according the Laplace error estimate or similar to the information function in CART. We can derive them from different fields: information theory, distance measures or dependence measures.
The quality measure of a rule is determined by:

\[ Q = \text{sensitivity} \cdot \text{specificity} \]  

We can say that the accuracy among positive instances determines sensitivity, and the accuracy among negative instances determines specificity. Currently we take into account only the rule accuracy, but it can be changed to analyze the rule length and interestingness.

Once each ant completes the construction of the rule, pheromone updating is carried out as follows:

\[ \tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t) \cdot Q, \forall \text{term} \in \text{the rule} \]  

The amount of the pheromone of terms belonging to the constructed rule is increased proportionately to the quality of \( Q \). To simulate pheromone evaporation \( \tau_{ij} \), the amount of pheromone associated with each \( \text{term} \) which does not occur in the constructed rule must be decreased. The reduction of pheromone of an unused term is performed by dividing the value of each \( \tau_{ij} \) by the summation of all \( \tau_{ij} \). The pheromone levels of all terms are then normalized.

3 Parallelization of Ant-Miner approach

A common procedure for implementing parallel programs is to employ low level message passing MIMD architecture (which presents different issues compared to shared memory architecture) in conjunction with a standard sequential language such as Fortran, C and C++. With this approach explicit communication calls for non local data access have to be inserted into the sequential program. In this paper we hope to show that this implementation with freely available OpenMP is well-aimed in order to reduce the cost of parallelization of Ant-Miner. OpenMP is an application programming interface for parallel programming on shared memory computers (multiprocessors). It consists of a set of compiler directives and a library of support functions. OpenMP works in conjunction with standard Fortran, C and C++ programming languages.

Distributed memory architecture was also considered, together with the corresponding programming interface (e.g. Message Passing Interface), but was rejected because of the additional costs associated with communication. Deeper changes in algorithm, such as increase in autonomy of the agents, could justify the use of a different architecture.

Our goal in parallelization of the Ant-Miner algorithm is to decrease the execution time of the algorithm without altering its behavior. Possible improvements of Ant-Miner solution quality through the exploitation of parallelism will be presented below. In this article different approaches of parallel algorithms will be discussed. Each of these models were used as standards to make Population-based Algorithms run in parallel.

- **Farm Model (Master-Slave Model)**

  In this architecture the processors used for the computation are divided into one Master (Farmer) and many Slave (Worker) processors. The Master processor divides a workload among the Worker processors. The results of each processor, which run independently, are transmitted back to the Master. The Master will evaluate them and if necessary re-transmit back to the Workers for another round of computation. The transmission of the initial workload and the re-transmission of data to the Workers may be considered as the parents for the next generation. The Workers will generate a set of results, which may be considered as the evaluated offspring of that generation.

- **Island Model**

  Here each processor is considered as one individual unit working independently. The results of each of them are migrated or exchanged with the others in some pre-defined rule or strategy. This is done at the end of each generation. Each processor (or island) is considered to have an equal workload. Other than the exchange of data at the end of each generation among the other processors all other time they run independently.

- **Diffusion Model (Neighborhood Model)**

  This is actually an extreme case of the island model. Almost all of the work is done completely locally within each processor. Only when the final result is needed thus they transmit their values, so that the necessary computations can be done. Here too generations are developed within each processor to obtain better solutions.

With the availability of OpenMP, which permits the parallelization of the existing sequential C program, it is interesting to examine this Farm model of Ant-Miner to solve the presented problem. As we can see in presented
Algorithm 1: Algorithm Ant-Miner

1. Create training set;
2. while training set > Max_uncovered_cases do
3.   Initialization_pheromone_trail;
4.   Calculate the_information_function;
5.   i := 0;
6. repeat
7.     i := i + 1;
8.     for a := 1 to Number_of_ants do
9.       while Number_of_admissible_attributes > 0 and Length_rule < Max_len_rule do
10.      Calculate the_heuristic for ant[i];
11.      Calculate the_probability;
12.      Select the_best_descriptor;
13.      Remove the_selected_attribute;
14.     endWhile;
15.     Determine decision_rule;
16.     Calculate quality_rule;
17.     Prune_rule;
18.     Select the_best_ant;
19. // rule with the best quality
20.     Update_pheromone_trail;
21. until i > Max_i or Is too much the same rules;
22. Select the_best_ants;
23. // rule with the highest quality
24. Remove the correctly covered by the new rule set of cases from the training set;
25. endWhile;

below algorithm the „for loop” is the main part of the algorithm and the source of its complexity. Besides these operations are strongly independent for each ant of a given procedure so they can be easily parallelized.

In our implementation p ants simply run independently until the predefined criteria is met. When all runs are completed, the master processor collects the solutions. Then we analyze the solutions through the master processor and choose the rule with the highest quality every determined interval of iterations synchronously (See Fig. 1).

Figure 1: Parallel Ant-Miner.

A first step in the parallelization process could be to naively affect the generation and evaluation of each ant to a different processor and by copying the pheromone matrix outside the parallel region. Only one update at a time could be done, after choosing the best rule created during the Ant-Miner procedures. In this reason, the chosen number of ants, which is a parameter now in our implementation of Ant-Miner, is limited by the number of processors we have at our disposal. It creates a problem when we need more ants for experimental study. It is then better to share the load between processors in a way to give more than one ant to each processor.

One of the most important structures that is used by ants is the pheromone matrix. Update of pheromone values is performed (via normalization of these stored values) once each cycle and cannot be parallelized without changing the structure of the presented algorithm, therefore it is done globally (by the Master processor) at the end of each cycle.
The whole rule induction procedure is of complexity:

\[ O(ants \cdot a^3 \cdot n) \]  

where:

- \( ants \) – the number of ants,
- \( a \) – number of attributes,
- \( n \) – number of cases in the training set.

Parallel version (under the assumption, that \( ants = p \), and \( p \) determines the number of processors) is of complexity:

\[ O(a^3 \cdot n) \]

There are two main reasons for using parallel Ant-Miner algorithm:

- increase in the quality of the solutions found within the fixed time to search,
- reduction of the time to find a solution not worse than given a fixed solution quality.

4 Experiments and results

4.1 Data sets used in our experiments

The evaluation of the performance behavior of different experiments of Parallel Ant-Miner was performed using 4 public-domain data sets, that allow easy comparison of results with other algorithms from the UCI (University of California at Irvine) data set repository available from: [www.ics.uci.edu/~mlearn/MLRepository.html](http://www.ics.uci.edu/~mlearn/MLRepository.html).

Table 2 shows the main characteristics of the data sets.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dermatology</td>
<td>366</td>
<td>34</td>
<td>139</td>
<td>6</td>
</tr>
<tr>
<td>German Credit Data</td>
<td>1000</td>
<td>20</td>
<td>1075</td>
<td>2</td>
</tr>
<tr>
<td>Mushroom</td>
<td>8124</td>
<td>22</td>
<td>112</td>
<td>7</td>
</tr>
<tr>
<td>Nursery</td>
<td>12960</td>
<td>8</td>
<td>27</td>
<td>6</td>
</tr>
</tbody>
</table>

Both the original Ant-Miner and our proposal have some parameters. The first one – the number of ants will be examined during the experiments. The rest of parameters are presented in Tab. 2. There are the following values of the parameters:

- the minimum number of cases per rule,
- the maximum number of uncovered cases in the training set,
- the number of rules used to test the convergence of the ants.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. Cases per Rule</td>
<td>5</td>
</tr>
<tr>
<td>Max. uncovered Cases</td>
<td>10</td>
</tr>
<tr>
<td>Rules for Convergence</td>
<td>10</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>50</td>
</tr>
</tbody>
</table>

4.2 Results

Before analyzing the results, let us see the run time of each experiment. We can see that each run time of the parallel model obtained almost better value as the run time of its base model. The biggest speed-up is observed for 64 ants (see Fig. 2). Looking at the values presented in Fig. 2, using 64 ants and 2 processors the speed-up is equal to 1.9 and for 64 ants and 8 processors the increase is equal to 4.7, respectively. This means that master-slave model finds best solutions faster in each run than classical Ant-Miner approach. Overall results shows that our parallel Ant-Miner implementation for rule induction leads to achievement of significant speed-ups.
The implementation used in this research executed the algorithm in parallel without significant quality degradation. All results with the parallel tests for different data sets show coincidence of solution quality as compared to the quality of their corresponding base model of Ant-Miner. From the results shown in Tab. 3 we can see the diminishing process. We observed about 10% diminished number of rules for each analyzed data sets. However, experiments are not as convincing as we expected, especially in the context of the accuracy of the classification (see Tab. 3). Nevertheless, the degradation of the efficiency obtained when increasing the number of processor is not globally observed during these experiments. In fact, when the number of ants is equal to 64 we get good efficiency, but not as high as we expected when 16 or 32 ants are employed in the case of dermatology data sets.

In this experimental study the smallest standard deviations in the execution time (see Tab. 4) can be observed among all instances. For 4 processors and 8 ants it reaches 18% and for 64 ants – about 14%, respectively.

As a summary of analysis in this section, we can say Master-Slave model showed the clear advantage for instances analyzed here. Also we could not observe which instances are the best for analyzed criteria. The main disadvantage of the used model of parallelism (Master-Slave) is that it prevents achievement of

Table 3: Comparative study. Accuracy of classification and standard deviation % depending on the ants number.

<table>
<thead>
<tr>
<th>Database</th>
<th>Ants</th>
<th>Accuracy</th>
<th>Number of rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>derm</td>
<td>8</td>
<td>0.92 (0.02)</td>
<td>5.58 (0.49)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>0.92 (0.02)</td>
<td>5.51 (0.50)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>0.91 (0.02)</td>
<td>5.35 (0.48)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.90 (0.01)</td>
<td>5.02 (0.15)</td>
</tr>
<tr>
<td>german</td>
<td>8</td>
<td>0.70 (0.00)</td>
<td>6.25 (0.45)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>0.70 (0.00)</td>
<td>6.28 (0.47)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>0.70 (0.00)</td>
<td>6.15 (0.39)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.70 (0.00)</td>
<td>6.10 (0.29)</td>
</tr>
<tr>
<td>mushroom</td>
<td>8</td>
<td>0.63 (0.02)</td>
<td>18.19 (2.23)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>0.63 (0.02)</td>
<td>17.39 (1.62)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>0.64 (0.02)</td>
<td>17.09 (1.32)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.64 (0.01)</td>
<td>16.54 (1.32)</td>
</tr>
<tr>
<td>nursery</td>
<td>8</td>
<td>0.84 (0.00)</td>
<td>14.21 (0.92)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>0.84 (0.00)</td>
<td>14.01 (0.20)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>0.84 (0.00)</td>
<td>14.00 (0.00)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.84 (0.00)</td>
<td>14.00 (0.00)</td>
</tr>
</tbody>
</table>

Table 4: Run time of the parallel implementation depending on the number of processors for data set `german`.

<table>
<thead>
<tr>
<th>Processors</th>
<th>Num. of ants</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>17.5 (3.4)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>24.7 (3.2)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>39.6 (6.9)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>92.6 (12.0)</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>10.7 (1.8)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>14.3 (2.2)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>24.0 (3.5)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>49.4 (7.4)</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>5.5 (1.0)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>7.9 (1.1)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>12.2 (1.9)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>25.8 (3.7)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processors</th>
<th>Num. of ants</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
<td>5.3 (1.0)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>5.8 (0.8)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>9.0 (1.3)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>20.0 (2.6)</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4.6 (0.8)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>6.0 (1.0)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>9.5 (1.4)</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>19.7 (2.5)</td>
</tr>
</tbody>
</table>
better speedups because of the need to synchronize processes (ants) before each global pheromone update. Processors, who have completed work faster, have to wait for the others. One way to solve the problem could be dynamic allocation of work, depending on the individual processors workload.

5 Conclusions

In this paper the efficiency and speed-up for parallelization of Ant-Miner algorithm has been reported. The speed-up obtainable through parallelization should be highly important for solving larger problem instances in real time. Our aim was to diminish the execution time of the analyzed Ant-Miner algorithm with different number of ants. The resulting implementation has shown that it was possible to design an efficient parallel Ant-Miner approach in rule induction. It also shown some limitation, as we increased the number of ants used in the adaptation in the specific data sets application, what causes some drawbacks, shown especially in the context of the tested data sets. The results showed a parallel implementation studied here improved solution quality of the basic algorithm. Although detail improvement features were different depending on problem instances of data mining, our results suggest that using the master-slave model, is promising in solution quality and the search speed for solving the presented problem.

The study of other parallel models, such as a combination of island and diffusion model remains for future work. To apply these models to problems with larger instances also remains for the next study.

Acknowledgement: We thank the Wroclaw Centre for Networking and Supercomputing (computing grant 04/97) where the computations of our project were carried out.

References