An Ant Colony Based System for Data Mining: Applications to Medical Data

Rafael S. Parpinelli[1] **Heitor S. Lopes**[1] **Alex A. Freitas**[2]

[1] CEFET-PR, CPGEI
Av. Sete de Setembro, 3165
Curitiba - PR, 80230-901
Brazil
rsparpin@cpgei.cefetpr.br
hslopes@cpgei.cefetpr.br

[2] PUC-PR, PPGIA-CCET
Rua Imaculada Conceicao, 1155
Curitiba - PR, 80215-901
Brazil
alex@ppgia.pucpr.br
http://www.ppgia.pucpr.br/~alex

Abstract

This work describes an algorithm for rule discovery in databases called AntMiner. The objective of the algorithm is the extraction of classification rules to be applied to unseen data as a decision aid. The algorithm used to discover such rules is inspired in the behavior of a real ant colony, as well as some concepts of information theory and data mining. AntMiner was applied to medical databases to obtain classification rules.

1 INTRODUCTION

Recently, there has been a growing interest in the data mining area, where the objective is the discovery of knowledge that is not only correct, but also comprehensible and even surprising to the user [Fayyad et al, 1996; Freitas and Lavington, 1998]. Therefore, the user can quickly understand the results of the system and combine them with his/her own knowledge of the problem in order to support a decision-making process.

When using data mining techniques, the discovered knowledge is frequently represented in the form of *IF <conditions> THEN <class>* rules. The *<conditions>* part is the antecedent of the rule and is a logical combination of the predicting attributes (for instance: *term1 AND term2 AND...*). Each term is a triple *<attribute, operator, value>*, where the element *operator* is a relational operator. In this work it is aimed to mine data only with categorical attributes, therefore the element *operator* of the triple will be '='. The *<class>* (consequent) contains the predicted class for the case whose attributes satisfy the *<conditions>* part of the rule.

There are several tasks in data mining and the most usual in the literature is classification. The classification task consists in associating an object/case to a class (among a predefined set of classes) based on the object/case's attributes.

To the best of our knowledge, the application of artificial Ant Colony Systems (ACS) [Dorigo et al, 1996] as a tool for classification-rule discovery is still unexplored and, probably, this is the first work to explore such approach. ACS use simple agents (artificial ants) that, when working together, cooperate with each other allowing the solution of problems with very large search spaces. In the context of rule discovery, this is achieved due to its ability to perform a flexible search over all possible logic combinations of the predicting attributes. Based on this feature of ACS's, we believe that they can be very promising for the data mining task addressed here. The only previous work on ant-based rule discovery that we are aware of is [Cordon et al, 2000]. However, in this work the rules are fuzzy rules, used in a fuzzy control system, rather than classification rules in the sense of data mining.

2 SOCIAL INSECTS AND REAL ANT SYSTEMS

Social insects like ants, bees and termites work by themselves in their simple tasks, independently of others members of the colony. However, when they act as a community, they are able to solve complex problems emerging in their daily lives, by means of mutual cooperation. This emergent behavior of a group of social insects is known as "swarm intelligence" [Bonabeau et al, 1999].

Ants are able to find the shortest path between a food source and the nest without the aid of visual information, and also to adapt to a changing environment [Dorigo et al, 1996]. It was found that the way ants communicate with each other to find the right way to follow is based on pheromone[1] trails. While ants move, they drop a certain amount of pheromone on the

[1] Pheromone is a chemical substance used as the communication media among individuals of the same species.

floor, leaving behind a trail of this substance that can be followed by other ants. The more ants follow a pheromone trail, the more attractive the trail becomes to be followed in the near future. This is a kind of autocatalytic behavior, described by a loop of positive feedback, where the probability of an ant choosing a path increases directly with the number of ants that have passed in the path before.

Finding the shortest path around an obstacle is an interesting emerging feature of the above-described autocatalytic behavior. In this case, an interaction between the obstacle shape and the distributed behavior of the ants takes place [Bonabeau et al, 1999]. The basic idea is illustrated in figure 1.

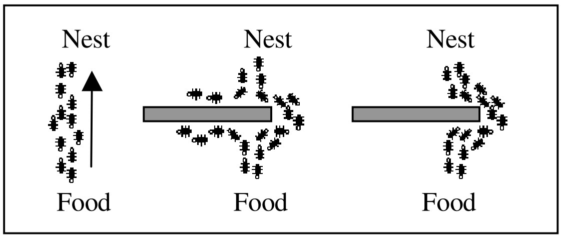

Figure 1: Ants finding the shortest path around an obstacle

Ants move roughly at the same speed and drop pheromone at the same rate. However, to go around the longer path an ant takes more time than going by the shorter path. This makes the pheromone to be accumulated faster in the shorter path than in the longer one. Besides, ants prefer to follow paths with more pheromone, leading to a faster convergence to the shorter path.

3 ARTIFICIAL ANT COLONY SYSTEMS

An artificial Ant Colony System is an algorithm based on agents that simulate the natural behavior of ants, developing mechanisms of cooperation and learning. The ACS was first proposed by [Dorigo et al, 1996] to be applied to combinatorial optimization. This new heuristics has been shown to be robust and versatile for different problems. In addition, ACS is a population-based heuristics that enables the exploration of the positive feedback between agents as a search mechanism.

There are some differences between real ants and ACS: artificial ants have memory and are not completely blind. Also, in the environment where they exist, time is discrete.

On the other hand, besides the pheromone-based communication medium, an ACS has another characteristic found in real ants: an artificial ant has a probabilistic preference for paths with a larger amount of pheromone. Consequently, shorter paths tend to have a high rate of growth in the amount of pheromone.

Essentially, an ACS algorithm performs a loop applying two basic procedures:

- A procedure specifying how ants construct or modify a solution for the problem in hand;
- A procedure for updating the pheromone trail.

The construction or modification of a solution is performed in a probabilistic way. The probability of adding a new item to the solution under construction is, in turn, a function of a problem-dependent heuristic (η) and the amount of pheromone (τ) previously deposited in this trail. The pheromone trails are updated considering the evaporation rate and the quality of the current solution. Therefore, a practical implementation of an ACS includes the following definitions [Bonabeau et al, 1999]:

- An appropriate representation for the problem with which ants can incrementally construct or modify solutions, by means of a probabilistic transition rule based on the amount of pheromone in the trail and on a local heuristic;
- A heuristic function (η) that measures the quality of the items that can be added to the current partial solution;
- A method to enforce the construction of valid solutions (in the real-world search space);
- A rule that specifies how a pheromone trail (τ) should be updated;
- A probabilistic transition rule that uses the current value of the heuristic function (η) and the current amount of pheromone in the trail (τ).

4 ANTMINER – THE PROPOSED ACS FOR CLASSIFICATION RULE DISCOVERY

This section discusses in detail the proposed system and it is divided into seven parts: AntMiner overview, heuristic function, pheromone updating, rule construction, rule pruning, classification of unseen cases and system parameters.

4.1 ANTMINER OVERVIEW

Recall that each ant can be regarded as an agent that incrementally constructs/modifies a solution for the target problem. In our case the target problem is the discovery of classification rules. As mentioned before, the rules are expressed in the form: *IF <conditions> THEN <class>*.

The *<conditions>* part (antecedent) of the rule contains a logical combination of predictor attributes, in the form: *term1* AND *term2* AND ... . Each term is a triple *<attribute, operator, value>*. The current version of AntMiner copes only with categorical attributes, so that the *operator* element in the triple is always "=". Continuous (real-valued) attributes are discretized as a preprocessing step. The *<class>* part (consequent) of the rule contains the class predicted for cases (objects or records) whose predictor attributes satisfy the <*conditions*> part of the rule.

Each ant starts with a rule with no term in its antecedent (empty rule), and adds one term at a time to its current partial rule. The current partial rule constructed by an ant corresponds to the current partial path followed by that ant. Similarly, the choice of a term to be added to the current partial rule corresponds to the choice of direction for which the current path will be extended, among all the possible directions (all terms that could be added to the current partial rule).

The choice of a term (attribute-value pair) to be added depends on both a problem-dependent heuristic function and on the amount of pheromone associated with each term, as will be discussed in detail in subsections 4.2 and 4.3, respectively.

An ant keeps adding terms one-at-a-time to its current partial rule until the ant is unable to continue constructing its rule. This situation can arise in two cases (described in more detail in subsection 4.4), namely: (a) when whichever term which could be added to the rule would make the rule cover a number of cases smaller than a user-specified threshold, called *Min_cases_per_rule* (minimum number of cases covered per rule); (b) when all attributes have already been used by the ant, so that there are no more attributes to be added to the rule antecedent.

When one of these two stopping criteria is satisfied the ant has built a rule (i.e. it has completed its path), and, in principle, we could use the discovered rule for classification. In practice, however, it is desirable to prune the discovered rules in a post-processing step, to remove irrelevant terms that might have been unduly included in the rule. These irrelevant terms may have been included in the rule due to stochastic variations in the term selection procedure and/or due to the use of a shortsighted, local heuristic function - which considers only one-attribute-at-a-time, ignoring attribute interactions. The pruning method used in Ant-Miner will be described in subsection 4.5.

When an ant completes its rule and the amount of pheromone in each trail is updated, another ants start to construct its rule, using the new amounts of pheromone to guide its search. This process is repeated for at most a predefined number of ants. This number is specified as a parameter of the system, called *No_of_ants*. However, this iterative process can be interrupted earlier, when the current ant has constructed a rule that is exactly the same as the rule constructed by the previous *No_Rules_Converg* – 1 ants. *No_Rules_Converg* (number of rules used to test convergence of the ants) is also a system parameter. This second stopping criterion detects that the ants have already converged to the same constructed rule, which is equivalent to converging to the same path in real Ant Colony Systems.

The best rule among the rules constructed by all ants is considered a discovered rule. The other rules are discarded. This completes one iteration of the system.

Then all cases correctly covered by the discovered rule are removed from the training set, and another iteration is started. Hence, the AntMiner algorithm is called again to find a rule in the reduced training set. This process is repeated for as many iterations as necessary to find rules covering almost all cases of the training set. More precisely, the above process is repeated until the number of uncovered cases in the training set is less than a predefined threshold, called *Max_uncovered_cases* (maximum number of uncovered cases in the training set).

A summarized description of the above-discussed iterative process is shown in the algorithm of Figure 2.

```
TrainingSet = {all training cases};
DiscoveredRuleList = [];  /* rule list is initialized with an empty list */
WHILE (TrainingSet ≥ Max_Uncovered_Cases)
   i = 1;  /* ant index */
   No_Ants_Converg = 1;  /* convergence test index */
   Initialize all trails with the same amount of pheromone;
   REPEAT
      Ant_i starts with an empty rule and incrementally
      constructs a classification rule R_i, by adding one term at
      a time to the current rule;
      Prune rule R_i;
      Update the pheromone of all trails, by increasing
      pheromone in the trail followed by Ant_i (in proportion to
      the quality of R_i) and decreasing pheromone in the other
      trails (simulating pheromone evaporation);
      IF (R_i is equal to R_i-1) /* update convergence test */
         THEN No_Ants_Converge = No_Ants_Converge + 1;
         ELSE No_Ants_Converge = 1;
      END IF
      i = i + 1;
   UNTIL (i ≥ No_of_Ants) OR
            (No_Ants_Converg ≥ No_Rules_Converg)
   Choose the best rule R_best among all rules R_i constructed by
   all the ants;
   Add rule R_best to DiscoveredRuleList;
   TrainingSet = TrainingSet - {set of cases correctly covered
   by R_best};
END WHILE
```

Figure 2: Overview of AntMiner

When the number of cases left in the training set is less than *Max_uncovered_cases* the search for rules stops. At this point the system has discovered several rules. The discovered rules are stored in an ordered rule list (in order of discovery), which will be used to classify new cases, unseen during training. The system also adds a default rule to the last position of the rule list. The default rule has an empty antecedent (i.e. no condition) and has a consequent predicting the majority class in the set of training cases that are not covered by any rule. This default rule is automatically applied if none of the previous rules in the list cover a new case to be classified.

Once the rule list is complete, the system is finally ready to classify a new test case unseen during training. In order to do this the system tries to apply the discovered rules, in order. The first rule that covers the new case is applied – i.e. the case is assigned the class predicted by that rule's consequent.

4.2 HEURISTIC FUNCTION

The heuristic function (η) is based on the amount of information (measured by the entropy[2]) associated with the attribute *i* with value *j*, i.e., (*i*|*j*) [Weiss and Kulikowski, 1991]. The amount of information is given by equation 1:

$$\text{info}T_{ij} = -\sum_{w=1}^{k}\left(\frac{freqT_{ij}^{w}}{|T_{ij}|}\right) * \log_2\left(\frac{freqT_{ij}^{w}}{|T_{ij}|}\right) \quad [1]$$

where:

- k is the number of classes in the dataset;
- $|T_{ij}|$ is the total number of cases in the data partition T_{ij} (partition that contains the cases where the attribute *i* is equal to the value *j*);
- freq T_{ij}^{w} represents the number of cases in T_{ij} that belong to class *w*.

The larger the entropy ($infoT_{ij}$), meaning classes more evenly distributed, the smaller the predictive power of the attribute-value pair (*i*|*j*).

In the case that attribute *i* with value *j* (*i*|*j*) does not appear in any case of the partition T_{ij}, that is, the value *j* is not present in the training set, then we set $infoT_{ij}$ = log_2*(number of classes)*, which is the maximum entropy. If the attribute *i* with value *j* identifies only one class in the partition T_{ij}, then $infoT_{ij} = 0$, which is the minimum entropy. The larger the value of $infoT_{ij}$ ($0 \le infoT_{ij} \le \log_2$(number of classes)), the smaller the probability that the ant chose attribute *i* with value *j*. Therefore, the heuristic criterion is given by equation 2:

$$ç_{ij} = \frac{\log_2(k) - \text{infoT}_{ij}}{\sum_{i}^{a}\sum_{j}^{b_i} \log_2(k) - \text{infoT}_{ij}} \quad [2]$$

where:

- a is the total number of attributes;
- b_i is the number of values in the domain of attribute *i*.

4.3 PHEROMONE UPDATING

Initially, for all attributes *i* and their possible values *j*, a given initial amount of pheromone is deposited in the respective position. This initial normalized amount is proportional to the total number of values of all attributes, and is given by equation 3:

$$\tau_{ij}(t=0) = \frac{1}{\left(\sum_{i}^{a} b_i\right)} \quad [3]$$

where:

- a is the total number of attributes;
- b_i is the number of possible values that can be taken by attribute *i.*

At the completion of a rule, the amount of pheromone in the *i*|*j* (attributes *i* | values *j*) that constitute the rule must be updated. This is accomplished with a rule quality criterion given by the product *sensitivity* * *specificity* [Lopes et al, 1998]. Equation 4 shows the rule quality criterion in detail. The larger the value of *Q*, the higher the quality of the rule ($0 \le Q \le 1$).

$$Q = \left(\frac{\text{TruePos}}{\text{TruePos} + \text{FalseNeg}}\right) * \left(\frac{\text{TrueNeg}}{\text{FalsePos} + \text{TrueNeg}}\right) \quad [4]$$

where:

- *TruePos* (true positives) is the number of cases covered by the rule that have the class predicted by the rule;
- *FalsePos* (false positives) is the number of cases covered by the rule that have a class different from the class predicted by the rule;
- *FalseNeg* (false negatives) is the number of cases that are not covered by the rule but that have the class predicted by the rule;
- *TrueNeg* (true negatives) is the number of cases that are not covered by the rule and do not have the class predicted by the rule.

The pheromone updating is performed as follows: for all terms *i*|*j* belonging to the rule created by the ant, the amount of pheromone is increased proportionally to *Q*, according to equation 5. The factor that represents the pheromone evaporation for the terms *i*|*j* that do not belong to the rule is obtained by normalizing the overall distribution of pheromone (given by equation 5) by the sum of all τ_{ij}.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t) * Q, \forall\ i \mid j \in \text{ to the rule} \quad [5]$$

4.4 RULE CONSTRUCTION

The probability P_{ij} of an ant chooses a given *i*|*j* not yet used in its current rule is given by equation 6. Note that ants build rules using two memories: memories for the attributes|values (*i*|*j*) and memories for the attributes (*i*). These memories contain the attribute-value pair *i*|*j* and the attribute *i* that was already used in the partial rule built by the ant.

[2] The entropy models the degree of 'disorganization' of the training cases considering the distribution of the classes to be predicted.

$$P_{ij} = \frac{\tau_{ij}(t).\eta_{ij}}{\sum_{i}^{a} \sum_{j}^{b_i} \tau_{ij}(t).\eta_{ij}}, \forall i \in I \qquad [6]$$

where:

- a is the total number of attributes;
- b_i is the total number of values on i domain;
- I are the attributes i not yet used by the ant.

These memories are updated according to the following conditions:

- If the attribute i was not yet used by the ant, its memory value is zero, being free to be chosen;
- If the insertion of the $i|j$ in the current rule of the ant would yield a rule covering a number of cases smaller than a given threshold (called *Min_cases_per_rule*), then this $i|j$ cannot be included in the rule. Thus, the memory value for this $i|j$ is adjusted to -1, a flag indicating that this $i|j$ will not be chosen anymore.

This term-inclusion procedure is repeated until all attributes are analyzed.

The definition of which class the rule generated by the ant predicts is given by the class of the majority (positive class) of the cases covered by the rule.

At the end of each iteration, the best rule generated so far by the ants (the rule with the highest Q) is kept. Then, all trails are reinitialized with the same amount of pheromone and a new iteration takes place (see Figure 2).

The search for better rules is stopped in two situations: either when the training set has a number of cases smaller than a specified threshold (*Max_uncovered_cases*), or when an ant cannot proceed constructing the rule. The latter condition occurs when any new attribute value to be inserted in the rule would cause the rule to cover a number of cases smaller than *Min_cases_per_rule*.

4.5 RULE PRUNING

A pruning procedure is used to reduce the number of terms of a rule in order to increase its quality (measured by equation 4). This procedure induces the discovery of more comprehensible (smaller) rules and helps to avoid the overfitting of rules to the training dataset.

An ant can build a rule as long as the number of partitions that can be done in the training set, respecting the threshold *Min_cases_per_rule*. After an ant builds a rule, the pruning procedure takes place by iteratively removing one condition at time. More precisely at each iteration the procedure computes, for each condition currently in the rule, what would be the value of the quality Q of the rule if that condition was removed. (This might require to modify the class predicted by the rule, since this is always the majority class among all cases covered by the rule antecedent.) After doing this for all conditions, the condition whose removal most improves the rule quality Q is effectively removed from the rule, and another iteration of the rule pruning procedure starts. This procedure is repeated until one can not improve the quality of the rule.

4.6 USING THE DISCOVERED RULES FOR CLASSIFYING NEW CASES

To classify a new test case, unseen during training, we try to apply the discovered rules, in the order they were discovered. The first rule that covers a new case is applied - i.e. the case is assigned the class predicted by that rule's consequent.

It is possible that no rule in the list of discovered rules covers the new case. In this situation the new case is classified by the default rule, which simply predicts the majority class in the set of training examples that are not covered by any discovered rule.

4.7 SYSTEM PARAMETERS

Our Ant Colony System has the following four user-defined parameters:

- Number of Ants (*No_of_ants*) → This is also the maximum number of complete candidate rules constructed during a single iteration of the system, since each ant constructs a single rule (see Figure 2). In each iteration, the best candidate rule constructed in that iteration is considered a discovered rule. Note that the larger the *No_of_ants*, the more candidate rules are evaluated per iteration, but the slower the system is;
- Mimimum number of cases per rule (*Min_cases_per_rule*) → Each rule must cover at least *Min_*cases*_per_rule*, to enforce at least a certain degree of generality in the discovered rules. This helps avoiding overfitting to the training data;
- Maximum number of uncovered cases in the training set (*Max_uncovered_cases*) → The process of rule discovery is iteratively performed until the number of training cases that are not covered by any discovered rule is smaller than this threshold (see Figure 2);
- Number of rules used to test convergence of the ants (*No_Rules_Converg*) → If the current ant has constructed a rule that is exactly the same as the rule constructed by the previous *No_Rules_Converg* – 1 ants, then the system concludes that the ants have converged to a single rule (path). The current iteration is therefore stopped, and another iteration is started (see Figure 2).

In all the experiments reported in this paper these parameters were set as follows:

- *No_of_ants = 3000;*

- *Min_cases_per_rule = 10;*
- *Max_uncovered_cases = 10;*
- *No_Rules_Converg = 10.*

We have made no serious attempt to optimize the setting of these parameters. Such an optimization should be tried in future research. It is interesting to notice that even the above non-optimized parameters' setting has produced quite good results, as will be seen in the next section. In addition, the fact that Ant-Miner parameters were not optimized for the data sets used in our experiments makes the comparison with C4.5 (reported in the next section) fair, since we used the default, non-optimized parameters for C4.5 as well. (In passing we mention that unfortunately this kind of fair comparison is not very often seen in the literature. Authors often report results comparing a parameter-optimized version of their algorithm with a non-parameter-optimized version of another algorithm. This makes the comparison less fair.)

5 DATA SETS USED IN THE EXPERIMENTS

Experiments were done using four public-domain datasets, obtained from the Machine Learning Repository [Aha and Murphy, 1994].

For all datasets, the continuous attributes were discretized using the C4.5-Disc algorithm [Kohavi and Sahami, 1996]. For each continuous attribute to be discretized, this class-driven discretization algorithm consists of using the well-known C4.5 algorithm [Quinlan, 1993] for generating a decision tree where: (a) internal nodes are tests on the values of the attribute being discretized; and (b) leaf nodes are classes. Each leaf node of the generated decision tree is associated with an interval of values of the attribute being discretized (defined by the path from the root node to that leaf node). Each of these generated intervals is considered a discrete value for the attribute being discretized. (See the above reference for details.)

The AntMiner system was tested using the following datasets:

- Ljubljana breast cancer: this database has 282 cases, two classes and nine predicting attributes (all categorical);
- Wisconsin breast cancer: This database has 683 cases, two classes and nine predicting attributes. All predicting attributes are continuous (in the range of 1 to 10) and were discretized;
- Hepatitis: This database has 155 cases, two classes and 19 predicting attributes (six of them were continuous, and so were discretized);
- Dermatology: This database has 358 cases, six classes and 34 predicting attributes (only one is continuous – age, and so was discretized).

6 COMPUTATIONAL RESULTS

Among the several criteria that could be used to evaluate the predictive accuracy of discovered rules, the cross-validation accuracy rate[3] was used. Although this measure is computationally expensive, it gives a wide exploration of the characteristics of the cases in the dataset [Weiss and Kulikowski, 1991]. For all datasets, a 10-fold cross validation ($k=10$) was used. In this procedure, all cases are used only once as testing and ($k-1$) times as training. The final accuracy rate is simply the average of the accuracy rate of the k iterations. All the k data partitions are randomly generated considering all available cases.

Table 1 summarizes the results obtained by the proposed AntMiner algorithm in the four datasets. The table shows the accuracy rate, the number of rules found and the number of terms (the shown values are the average values of the cross-validation procedure followed by the corresponding standard deviation).

Table 1: Results With The AntMiner Algorithm

Data Sets	**Predictive Accuracy**	**Number of Rules**	**Number of Conditions**
Ljubljana Breast Cancer	75.13% ± 6.00	5.20 ± 0.87	8.80 ± 1.89
Wisconsin Breast Cancer	95.47% ± 1.62	5.60 ± 0.80	12.50 ± 2.84
Hepatitis	88.75% ± 6.73	2.70 ± 0.46	7.50 ± 2.01
Dermatology	84.21% ± 6.34	6.00 ± 0.00	79.00 ± 3.46

The obtained results were compared with other machine learning methods found in the literature, for the same datasets. Table 2 compares the accuracy rate (on the test set) of AntMiner with the accuracy rate of the well-known C4.5 algorithm [Quinlan, 1993] using a 10-fold cross-validation procedure for both algorithms.

Table 2: AntMiner Versus C4.5

	Accuracy	**Number of Rules**	**Number of Conditions**
Ljubljana Breast Cancer Data Set			
AntClass	**75.13% ± 6.00**	5.20 ± 0.87	8.80 ± 1.89
C4.5	73.34% ± 3.21	6.2 ± 4.2	12.8 ± 9.83
Wisconsin Breast Cancer Data Set			
AntClass	**95.47% ± 1.62**	5.60 ± 0.80	12.50 ± 2.84
C4.5	95.02% ± 0.31	11.1 ± 1.45	44.1 ± 7.48
Hepatitis Data Set			
AntClass	**88.75% ± 6.73**	2.70 ± 0.46	7.50 ± 2.01
C4.5	85.96% ± 1.07	4.4 ± 0.93	8.5 ± 3.04
Dermatology Data Set			
AntClass	84.21% ± 6.34	6.00 ± 0.00	79.00 ± 3.46
C4.5	**89.05% ± 0.62**	23.2 ± 1.99	91.7 ± 10.64

Furthermore, Table 3 compares AntMiner with an evolutionary algorithm for rule discovery called ESIA - Extended Genetic Rule Induction Algorithm [Liu and Kwok, 2000], on the Wisconsin breast cancer dataset.

[3] Accuracy rate is defined as the quotient between the number of test cases correctly classified and the total number of test cases.

Table 3: Comparison between AntMiner and ESIA (Wisconsin breast cancer data set)

	Accuracy	Number of Rules	Number of Conditions
AntClass	**95.47% ± 1.62**	5.60 ± 0.80	12.50 ± 2.84
ESIA	94.71% ±0.04	23.9	-

7 CONCLUSIONS

We have described an Ant Colony System called AntMiner for the discovery of classification rules in databases. We have also shown results indicating that AntMiner had a good classification performance on the four datasets used in our experiments. These results also show that the proposed algorithm is able to achieve both good predictive accuracy and a reduced number of rules at the same time. This facilitates the practical use of the system, since it usually generates comprehensible rules. The main drawback is still its computational cost, especially when the search space (number of predicting attributes) is too large. Notwithstanding the algorithm proposed here is very promising, and more experiments will be done in the future, as well as other improvements.

REFERENCES

D.W. Aha and P.M. Murphy (1994). UCI Repository of machine learning databases. [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

E. Bonabeau, M. Dorigo, and G. Theraulaz (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press.

O. Cordón, J. Casillas, and F. Herrera (2000). Learning Fuzzy Rules Using Ant Colony Optimization. *Proc. ANTS'2000 – From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms*, pp. 13-21.

M. Dorigo, A. Colorni, and V. Maniezzo (1996). The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, no 1, pp. 1-13.

U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth (1996). From data mining to knowledge discovery: an overview. In: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.) *Advances in Knowledge Discovery & Data Mining*, 1-34. Cambridge: AAAI/MIT.

A. A. Freitas and S. H. Lavington (1998). *Mining Very Large Databases with Parallel Processing*. London: Kluwer.

R. Kohavi and M. Sahami (1996). Error-based and entropy-based discretization of continuous features. *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, pp. 114-119.

J. J. Liu and J. T. Kwok (2000). An Extended genetic rule induction algorithm. In *Proc. CEC'2000*, pp. 458-463.

H. S. Lopes, M. S. Coutinho, and W. C. Lima (1998). An Evolutionary approach to simulate cognitive Feedback learning in medical domain. In *Genetic algorithms and Fuzzy Logic Systems*. Soft Computing Perspectives, Singapore: World Scientific, pp. 193-207.

J. R. Quinlan (1993). *C4.5: Programs for Machine Learning*. San Francisco: Morgan Kaufmann.

S. M. Weiss and C. A. Kulikowski (1991). *Computer Systems that Learn* - Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems, San Francisco: Morgan Kaufmann.