Data Mining with an Ant Colony Optimization Algorithm

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

[1] CEFET-PR, CPGEI, Av. Sete de Setembro, 3165, Curitiba - PR, 80230-901, Brazil

[2] PUC-PR, PPGIA-CCET, Rua Imaculada Conceição, 1155, Curitiba - PR, 80215-901, Brazil.

***Abstract* – This work proposes an algorithm for data mining called Ant-Miner (Ant Colony-based Data Miner). The goal of Ant-Miner is to extract classification rules from data. The algorithm is inspired by both research on the behavior of real ant colonies and some data mining concepts and principles. We compare the performance of Ant-Miner with CN2, a well-known data mining algorithm for classification, in six public domain data sets. The results provide evidence that: (a) Ant-Miner is competitive with CN2 with respect to predictive accuracy; and (b) The rule lists discovered by Ant-Miner are considerably simpler (smaller) than those discovered by CN2.**

***Index Terms* – Ant Colony Optimization, data mining, knowledge discovery, classification.**

I. Introduction

In essence, the goal of data mining is to extract knowledge from data. Data mining is an inter-disciplinary field, whose core is at the intersection of machine learning, statistics and databases.

We emphasize that in data mining – unlike for example in classical statistics – the goal is to discover knowledge that is not only accurate but also comprehensible for the user [12] [13]. Comprehensibility is important whenever discovered knowledge will be used for supporting a decision made by a human user. After all, if discovered knowledge is not comprehensible for the user, he/she will not be able to interpret and validate it. In this case, probably the user will not trust enough the discovered knowledge to use it for decision making. This can lead to wrong decisions.

There are several data mining tasks, including classification, regression, clustering, dependence modeling, etc. [12]. Each of these tasks can be regarded as a kind of problem to be solved by a data mining algorithm. Therefore, the first step in designing a data mining algorithm is to define which task the algorithm will address.

In this paper we propose an Ant Colony Optimization (ACO) algorithm [10] [11] for the classification task of data mining. In this task the goal is to assign each case (object, record, or instance) to one class, out of a set of predefined classes, based on the values of some attributes (called predictor attributes) for the case.

In the context of the classification task of data mining, discovered knowledge is often expressed in the form of IF-THEN rules, as follows:

IF <conditions> THEN < class>.

The rule antecedent (IF part) contains a set of conditions, usually connected by a logical conjunction operator (AND). In this paper we will refer to each rule condition as a term, so that the rule antecedent is a logical conjunction of terms in the form: IF *term1* AND *term2* AND ... Each term is a triple <*attribute, operator, value*>, such as <Gender = female>.

The rule consequent (THEN part) specifies the class predicted for cases whose predictor attributes satisfy all the terms specified in the rule antecedent. From a data mining viewpoint, this kind of knowledge representation has the advantage of being intuitively comprehensible for the user, as long as the number of discovered rules and the number of terms in rule antecedents are not large.

To the best of our knowledge, the use of ACO algorithms [9] [10] [11] for discovering classification rules, in the context of data mining, is a research area still unexplored. Actually, the only ant algorithm developed for data mining that we are aware of is an algorithm for clustering [17], which is a data mining task very different from the classification task addressed in this paper.

We believe that the development of ACO algorithms for data mining is a promising research area, due to the following reason. ACO algorithms involve simple agents (ants) that cooperate with one another to achieve an emergent, unified behavior for the system as a whole, producing a robust system capable of finding high-quality solutions for problems with a large search space. In the context of rule discovery, an ACO algorithm has the ability to perform a flexible, robust search for a good combination of terms (logical conditions) involving values of the predictor attributes.

The rest of this paper is organized as follows. Section II presents a brief overview of real ant colonies. Section III discusses the main characteristics of ACO algorithms. Section IV introduces the ACO algorithm for discovering classification rules proposed in this work. Section V reports computational results evaluating the performance of the proposed algorithm. In particular, in that section we compare the proposed algorithm with

CN2, a well-known algorithm for classification-rule discovery, across six data sets. Finally, Section VI concludes the paper and points directions for future research.

II. Social Insects and Real Ant Colonies

In a colony of social insects, such as ants, bees, wasps and termites, each insect usually performs its own tasks independently from other members of the colony. However, the tasks performed by different insects are related to each other in such a way that the colony, as a whole, is capable of solving complex problems through cooperation [2]. Important, survival-related problems such as selecting and picking up materials, finding and storing food, which require sophisticated planning, are solved by insect colonies without any kind of supervisor or centralized controller. This collective behavior which emerges from a group of social insects has been called "swarm intelligence" [2].

In this paper we are interested in a particular behavior of real ants, namely the fact that they are capable of finding the shortest path between a food source and the nest (adapting to changes in the environment) without the use of visual information [9]. This intriguing ability of almost-blind ants has been extensively studied by ethologists. They discovered that, in order to exchange information about which path should be followed, ants communicate with one another by means of pheromone (a chemical substance) trails. As ants move, a certain amount of pheromone is dropped on the ground, marking the path with a trail of this substance. The more ants follow a given trail, the more attractive this trail becomes to be followed by other ants. This process can be described as a loop of positive feedback, in which the probability that an ant chooses a path is proportional to the number of ants that have already passed by that path [9] [11] [23].

When an established path between a food source and the ants' nest is disturbed by the presence of an object, ants soon will try to go around the obstacle. Firstly, each ant can choose to go around to the left or to the right of the object with a 50%-50% probability distribution. All ants move roughly at the same speed and deposit pheromone in the trail at roughly the same rate. Therefore, the ants that (by chance) go around the obstacle by the shortest path will reach the original track faster than the others that have followed longer paths to circumvent the obstacle. As a result, pheromone accumulates faster in the shorter path around the obstacle. Since ants prefer to follow trails with larger amounts of pheromone, eventually all the ants converge to the shorter path.

III. ANT COLONY OPTIMIZATION

An Ant Colony Optimization algorithm (ACO) is essentially a system based on agents which simulate the natural behavior of ants, including mechanisms of cooperation and adaptation. In [10] the use of this kind of system as a new metaheuristic was proposed in order to solve combinatorial optimization problems. This new metaheuristic has been shown to be both robust and versatile – in the sense that it has been successfully applied to a range of different combinatorial optimization problems [11].

ACO algorithms are based on the following ideas:

- Each path followed by an ant is associated with a candidate solution for a given problem.
- When an ant follows a path, the amount of pheromone deposited on that path is proportional to the quality of the corresponding candidate solution for the target problem.
- When an ant has to choose between two or more paths, the path(s) with a larger amount of pheromone have a greater probability of being chosen by the ant.

As a result, the ants eventually converge to a short path, hopefully the optimum or a near-optimum solution for the target problem, as explained before for the case of natural ants.

In essence, the design of an ACO algorithm involves the specification of [2]:

- An appropriate representation of the problem, which allows the ants to incrementally construct/modify solutions through the use of a probabilistic transition rule, based on the amount of pheromone in the trail and on a local, problem-dependent heuristic.
- A method to enforce the construction of valid solutions, that is, solutions that are legal in the real-world situation corresponding to the problem definition.
- A problem-dependent heuristic function (η) that measures the quality of items that can be added to the current partial solution.
- A rule for pheromone updating, which specifies how to modify the pheromone trail (τ).
- A probabilistic transition rule based on the value of the heuristic function (η) and on the contents of the pheromone trail (τ) that is used to iteratively construct a solution.

Artificial ants have several characteristics similar to real ants, namely:

- Artificial ants have a probabilistic preference for paths with a larger amount of pheromone.

- Shorter paths tend to have larger rates of growth in their amount of pheromone.
- The ants use an indirect communication system based on the amount of pheromone deposited on each path.

IV. Ant-Miner: A New ACO Algorithm for Data Mining

In this section we discuss in detail our proposed Ant Colony Optimization algorithm for the discovery of classification rules, called Ant-Miner. The section is divided into five subsections, namely: a general description of Ant-Miner, heuristic function, rule pruning, pheromone updating, and use of the discovered rules for classifying new cases.

A. A General Description of Ant-Miner

In an ACO algorithm each ant incrementally constructs/modifies a solution for the target problem. In our case the target problem is the discovery of classification rules. As discussed in the introduction, each classification rule has the form:

IF < *term1* AND *term2* AND ...> *THEN* <*class*> .

Each term is a triple <*attribute, operator, value*>, where *value* is a value belonging to the domain of *attribute*. The operator element in the triple is a relational operator. The current version of Ant-Miner copes only with categorical attributes, so that the operator element in the triple is always "=". Continuous (real-valued) attributes are discretized in a preprocessing step.

A high-level description of Ant-Miner is shown in Algorithm I. Ant-Miner follows a sequential covering approach to discover a list of classification rules covering all, or almost all, the training cases. At first, the list of discovered rules is empty and the training set consists of all the training cases. Each iteration of the WHILE loop of Algorithm I, corresponding to a number of executions of the REPEAT-UNTIL loop, discovers one classification rule. This rule is added to the list of discovered rules, and the training cases that are correctly covered by this rule (i.e., cases satisfying the rule antecedent and having the class predicted by the rule consequent) are removed from the training set. This process is iteratively performed while the number of uncovered training cases is greater than a user-specified threshold, called *Max_uncovered_cases*.

Each iteration of the REPEAT-UNTIL loop of Algorithm I consists of three steps, comprising rule construction,

rule pruning, and pheromone updating, detailed as follows.

First, Ant_t starts with an empty rule, that is, a rule with no term in its antecedent, 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 the direction in which the current path will be extended. The choice of the term to be added to the current partial rule depends on both a problem-dependent heuristic function (η) and on the amount of pheromone (τ) associated with each term, as will be discussed in detail in the next subsections. Ant_t keeps adding one-term-at-a-time to its current partial rule until one of the following two stopping criteria is met:

- Any term to 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).
- All attributes have already been used by the ant, so that there is no more attributes to be added to the rule antecedent. Note that each attribute can occur only once in each rule, to avoid invalid rules such as “IF (Sex = male) AND (Sex = female) ...”.

ALGORITHM I: A HIGH-LEVEL DESCRIPTION OF ANT-MINER

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

Second, rule R_t constructed by Ant_t is pruned in order to remove irrelevant terms, as will be discussed later. For the moment, we only mention that 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.

Third, the amount of pheromone in each trail is updated, increasing the pheromone in the trail followed by Ant_t (according to the quality of rule R_t) and decreasing the pheromone in the other trails (simulating the pheromone evaporation). Then another ant starts to construct its rule, using the new amounts of pheromone to guide its search.

This process is repeated until one of the following two conditions is met:

- The number of constructed rules is equal to or greater than the user-specified threshold *No_of_ants*.
- The current Ant_t has constructed a rule that is exactly the same as the rule constructed by the previous $No\_rules\_converg - 1$ ants, where $No\_rules\_converg$ stands for the number of rules used to test convergence of the ants. The term *convergence* is defined in Section V.

Once the REPEAT-UNTIL loop is completed, the best rule among the rules constructed by all ants is added to the list of discovered rules, as mentioned earlier, and the system starts a new iteration of the WHILE loop, by reinitializing all trails with the same amount of pheromone.

It should be noted that, in a standard definition of ACO [10], a population is defined as the set of ants that build solutions between two pheromone updates. According to this definition, in each iteration of the WHILE loop Ant-Miner works with a population of a single ant, since pheromone is updated after a rule is constructed by an ant. Therefore, strictly speaking, each iteration of the WHILE loop of Ant-Miner has a single ant which performs many iterations. Note that different iterations of the WHILE loop correspond to different populations, since each population's ant tackles a different problem, that is, a different training set. However, in the text we refer to the *t*-th iteration of the ant as a separate ant, called the *t*-th ant (Ant_t), in order to simplify the description of the algorithm.

From a data mining viewpoint the core operation of Ant-Miner is the first step of the REPEAT-UNTIL loop of Algorithm I, in which the current ant iteratively adds one term at a time to its current partial rule. Let $term_{ij}$ be a rule condition of the form $A_i = V_{ij}$, where A_i is the *i*-th attribute and V_{ij} is the *j*-th value of the domain of A_i. The probability that $term_{ij}$ is chosen to be added to the current partial rule is given by Equation (1):

$$P_{ij} = \frac{\eta_{ij} \cdot \tau_{ij}(t)}{\sum_{i=1}^{a} x_i \cdot \sum_{j=1}^{b_i} (\eta_{ij} \cdot \tau_{ij}(t))} \quad (1)$$

where:

- η_{ij} is the value of a problem-dependent heuristic function for $term_{ij}$. The higher the value of η_{ij}, the more relevant for classification the $term_{ij}$ is, and so the higher its probability of being chosen. The function that defines the problem-dependent heuristic value is based on information theory, and it will be discussed in the next section.
- $\tau_{ij}(t)$ is the amount of pheromone associated with $term_{ij}$ at iteration *t*, corresponding to the amount of

pheromone currently available in the position *i,j* of the path being followed by the current ant. The better the quality of the rule constructed by an ant, the higher the amount of pheromone added to the trail segments visited by the ant. Therefore, as time goes by, the best trail segments to be followed – that is, the best terms (attribute-value pairs) to be added to a rule – will have greater and greater amounts of pheromone, increasing their probability of being chosen.

- a is the total number of attributes.
- x_i is set to 1 if the attribute A_i was not yet used by the current ant, or to 0 otherwise.
- b_i is the number of values in the domain of the i-th attribute.

A $term_{ij}$ is chosen to be added to the current partial rule with probability proportional to the value of Equation (1), subject to two restrictions, namely:

- The attribute A_i cannot be already contained in the current partial rule. In order to satisfy this restriction the ants must "remember" which terms (attribute-value pairs) are contained in the current partial rule.
- A $term_{ij}$ cannot be added to the current partial rule if this makes it cover less than a predefined minimum number of cases, called the *Min_cases_per_rule* threshold, as mentioned earlier.

Once the rule antecedent is completed, the system chooses the rule consequent (i.e., the predicted class) that maximizes the quality of the rule. This is done by assigning to the rule consequent the majority class among the cases covered by the rule.

B. Heuristic Function

For each $term_{ij}$ that can be added to the current rule, Ant-Miner computes the value η_{ij} of a heuristic function that is an estimate of the quality of this term, with respect to its ability to improve the predictive accuracy of the rule. This heuristic function is based on Information Theory [7]. More precisely, the value of η_{ij} for $term_{ij}$ involves a measure of the entropy (or amount of information) associated with that term. For each $term_{ij}$ of the form $A_i=V_{ij}$ – where A_i is the i-th attribute and V_{ij} is the j-th value belonging to the domain of A_i – its entropy is

$$H(W|A_i = V_{ij}) = -\sum_{w=1}^{k} \left(P(w|A_i = V_{ij}) \cdot \log_2 P(w|A_i = V_{ij}) \right) \quad (2)$$

where:

- W is the class attribute (i.e., the attribute whose domain consists of the classes to be predicted).
- k is the number of classes.
- $P(w|A_i=V_{ij})$ is the empirical probability of observing class w conditional on having observed $A_i=V_{ij}$.

The higher the value of $H(W|A_i=V_{ij})$, the more uniformly distributed the classes are and so, the smaller the probability that the current ant chooses to add $term_{ij}$ to its partial rule. It is desirable to normalize the value of the heuristic function to facilitate its use in Equation (1). In order to implement this normalization, it is used the fact that the value of $H(W|A_i=V_{ij})$ varies in the range $0 \leq H(W|A_i=V_{ij}) \leq \log_2 k$, where k is the number of classes. Therefore, the proposed normalized, information-theoretic heuristic function is:

$$\eta_{ij} = \frac{\log_2 k - H(W|A_i = V_{ij})}{\sum_{i=1}^{a} x_i \cdot \sum_{j=1}^{b_i} \left(\log_2 k - H(W|A_i = V_{ij})\right)} \tag{3}$$

where a, x_i, and b_i have the same meaning as in Equation (1).

Note that the $H(W|A_i=V_{ij})$ of $term_{ij}$ is always the same, regardless of the contents of the rule in which the term occurs. Therefore, in order to save computational time, the $H(W|A_i=V_{ij})$ of all $term_{ij}$ is computed as a preprocessing step.

In the above heuristic function there are just two minor caveats. First, if the value V_{ij} of attribute A_i does not occur in the training set then $H(W|A_i=V_{ij})$ is set to its maximum value of $\log_2 k$. This corresponds to assigning to $term_{ij}$ the lowest possible predictive power. Second, if all the cases belong to the same class then $H(W|A_i=V_{ij})$ is set to 0. This corresponds to assigning to $term_{ij}$ the highest possible predictive power.

The heuristic function used by Ant-Miner, the entropy measure, is the same kind of heuristic function used by decision-tree algorithms such as C4.5 [19]. The main difference between decision trees and Ant-Miner, with respect to the heuristic function, is that in decision trees the entropy is computed for an attribute as a whole, since an entire attribute is chosen to expand the tree, whereas in Ant-Miner the entropy is computed for an attribute-value pair only, since an attribute-value pair is chosen to expand the rule.

In addition, we emphasize that in conventional decision tree algorithms the entropy measure is normally the only heuristic function used during tree building, whereas in Ant-Miner the entropy measure is used together with

pheromone updating. This makes the rule-construction process of Ant-Miner more robust and less prone to get trapped into local optima in the search space, since the feedback provided by pheromone updating helps to correct some mistakes made by the shortsightedness of the entropy measure. Note that the entropy measure is a local heuristic measure, which considers only one attribute at a time, and so is sensitive to attribute interaction problems. In contrast, pheromone updating tends to cope better with attribute interactions, since pheromone updating is directly based on the performance of the rule as a whole (which directly takes into account interactions among all attributes occurring in the rule).

The process of rule construction used by Ant-Miner should lead to very bad rules at the beginning of the REPEAT-UNTIL loop, when all terms have the same amount of pheromone. However, this is not necessarily a bad thing for the algorithm as a whole. Actually, we can make here an analogy with evolutionary algorithms, say Genetic Algorithms (GA).

In a GA for rule discovery, the initial population will contain very bad rules as well. Actually, the rules in the initial population of a GA will probably be even worse than the rules built by the first ants of Ant-Miner, because a typical GA for rule discovery creates the initial population at random, without any kind of heuristic (whereas Ant-Miner uses the entropy measure). As a result of the evolutionary process, the quality of the rules in the population of the GA will gradually improve, producing better and better rules, until it converges to a good rule or a set of good rules, depending on how the individual is represented.

This is the same basic idea, at a very high level of abstraction, of Ant-Miner. Instead of natural selection in GA, Ant-Miner uses pheromone updating to produce better and better rules. Therefore, the basic idea of Ant-Miner's search method is more similar to evolutionary algorithms than to the search method of decision tree and rule induction algorithms. As a result of this approach, Ant-Miner (and evolutionary algorithms in general) perform a more global search, which is less likely to get trapped into local maxima associated with attribute interaction.

For a general discussion about why evolutionary algorithms tend to cope better with attribute interactions than greedy, local search-based decision tree and rule induction algorithms, the reader is referred to [8] [14].

C. Rule Pruning

Rule pruning is a commonplace technique in data mining [3]. As mentioned earlier, the main goal of rule pruning is to remove irrelevant terms that might have been unduly included in the rule. Rule pruning potentially

increases the predictive power of the rule, helping to avoid its overfitting to the training data. Another motivation for rule pruning is that it improves the simplicity of the rule, since a shorter rule is usually easier to be understood by the user than a longer one.

As soon as the current ant completes the construction of its rule, the rule pruning procedure is called. The strategy for the rule pruning procedure is similar to that suggested by [18], but the rule quality criteria used in the two procedures are very different.

The basic idea is to iteratively remove one-term-at-a-time from the rule while this process improves the quality of the rule. More precisely, in the first iteration one starts with the full rule. Then it is tentatively tried to remove each of the terms of the rule – each one in turn – and the quality of the resulting rule is computed using a given rule-quality function (to be defined by Equation (5)). It should be noted that this step might involve replacing the class in the rule consequent, since the majority class in the cases covered by the pruned rule can be different from the majority class in the cases covered by the original rule. The term whose removal most improves the quality of the rule is effectively removed from it, completing the first iteration. In the next iteration it is removed again the term whose removal most improves the quality of the rule, and so on. This process is repeated until the rule has just one term or until there is no term whose removal will improve the quality of the rule.

D. Pheromone Updating

Recall that each $term_{ij}$ corresponds to a segment in some path that can be followed by an ant. At each iteration of the WHILE loop of Algorithm I all $term_{ij}$ are initialized with the same amount of pheromone, so that when the first ant starts its search, all paths have the same amount of pheromone. The initial amount of pheromone deposited at each path position is inversely proportional to the number of values of all attributes, and is defined by Equation (4):

$$\tau_{ij}(t=0) = \frac{1}{\sum_{i=1}^{a} b_i} \tag{4}$$

where a is the total number of attributes, and b_i is the number of possible values that can be taken on by attribute A_i.

The value returned by this equation is normalized to facilitate its use in Equation (1), which combines this value and the value of the heuristic function. Whenever an ant constructs its rule and that rule is pruned (see Algorithm I) the amount of pheromone in all segments of all paths must be updated. This pheromone updating is supported by two basic ideas, namely:

- The amount of pheromone associated with each $term_{ij}$ occurring in the rule found by the ant (after pruning) is increased in proportion to the quality of that rule.
- The amount of pheromone associated with each $term_{ij}$ that does not occur in the rule is decreased, simulating pheromone evaporation in real ant colonies.

1) Increasing the Pheromone of Used Terms

Increasing the amount of pheromone associated with each $term_{ij}$ occurring in the rule found by an ant corresponds to increasing the amount of pheromone along the path completed by the ant. In a rule discovery context, this corresponds to increasing the probability of $term_{ij}$ being chosen by other ants in the future in proportion to the quality of the rule. The quality of a rule, denoted by Q, is computed by the formula: Q = *sensitivity* • *specificity* [16], defined as:

$$Q = \frac{TP}{TP + FN} \cdot \frac{TN}{FP + TN} \tag{5}$$

where:

- *TP* (true positives) is the number of cases covered by the rule that have the class predicted by the rule.
- *FP* (false positives) is the number of cases covered by the rule that have a class different from the class predicted by the rule.
- *FN* (false negatives) is the number of cases that are not covered by the rule but that have the class predicted by the rule.
- *TN* (true negatives) is the number of cases that are not covered by the rule and that do not have the class predicted by the rule.

Q´s value is within the range $0 \leq Q \leq 1$ and, the larger the value of Q, the higher the quality of the rule. Pheromone updating for a $term_{ij}$ is performed according to Equation (6), for all terms $term_{ij}$ that occur in the rule.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t) \cdot Q, \forall i,j \in R \quad (6)$$

where R is the set of terms occurring in the rule constructed by the ant at iteration t.

Therefore, for all $term_{ij}$ occurring in the rule found by the current ant, the amount of pheromone is increased by a fraction of the current amount of pheromone, and this fraction is given by Q.

2) Decreasing the Pheromone of Unused Terms

As mentioned above, the amount of pheromone associated with each $term_{ij}$ that does not occur in the rule found by the current ant has to be decreased in order to simulate pheromone evaporation in real ant colonies. In Ant-Miner, pheromone evaporation is implemented in a somewhat indirect way. More precisely, the effect of pheromone evaporation for unused terms is achieved by normalizing the value of each pheromone τ_{ij}. This normalization is performed by dividing the value of each τ_{ij} by the summation of all τ_{ij}, $\forall i,j$.

To see how this implements pheromone evaporation, remember that only the terms used by a rule have their amount of pheromone increased by Equation (6). Therefore, at normalization time the amount of pheromone of an unused term will be computed by dividing its current value (not modified by Equation (6)) by the total summation of pheromone for all terms (which was increased as a result of applying Equation (6) to all used terms). The final effect will be the reduction of the normalized amount of pheromone for each unused term. Used terms will, of course, have their normalized amount of pheromone increased due to the application of Equation (6).

E. Use of the Discovered Rules for Classifying New Cases

In order to classify a new test case, unseen during training, the discovered rules are applied in the order they were discovered (recall that discovered rules are kept in an ordered list). The first rule that covers the new case is applied – that is, the case is assigned the class predicted by that rule's consequent.

It is possible that no rule of the list covers the new case. In this situation the new case is classified by a default rule that simply predicts the majority class in the set of uncovered training cases, this is, the set of cases that are

not covered by any discovered rule.

V. Computational Results and Discussion

A. Data Sets and Discretization Method Used in the Experiments

The performance of Ant-Miner was evaluated using six public-domain data sets from the UCI (University of California at Irvine) repository[1].

The main characteristics of the data sets used in our experiment are summarized in Table I. The first column of this table gives the data set name, while the other columns indicate, respectively, the number of cases, the number of categorical attributes, the number of continuous attributes, and the number of classes of the data set.

As mentioned earlier, Ant-Miner discovers rules referring only to categorical attributes. Therefore, continuous attributes have to be discretized in a preprocessing step. This discretization was performed by the C4.5-Disc discretization method [15]. This method simply uses the very well-known C4.5 algorithm [19] for discretizing continuous attributes. In essence, for each attribute to be discretized it is extracted, from the training set, a reduced data set containing only two attributes: the attribute to be discretized and the goal (class) attribute. C4.5 is then applied to this reduced data set. Therefore, C4.5 constructs a decision tree in which all internal nodes refer to the attribute being discretized. Each path in the constructed decision tree corresponds to the definition of a categorical interval produced by C4.5. See [15] for further details.

TABLE I: Data sets used in The experiments: the first column gives the data set name, while The other columns indicate, respectively, The number of cases, the number of categorical attributes, the number of continuous attributes, and the number of classes of the data set

Data Set	#cases	#categ. attrib.	#contin. attrib.	#classes
Ljubljana breast cancer	282	9	-	2
Wisconsin breast cancer	683	-	9	2
tic-tac-toe	958	9	-	2
dermatology	366	33	1	6
hepatitis	155	13	6	2
Cleveland heart disease	303	8	5	5

[1]Available on the Internet at: http://www.ics.uci.edu/~mlearn/MLRepository.html.

B. Ant-Miner's Parameter Setting

Recall that Ant-Miner has the following four user-defined parameters (mentioned throughout Section IV):

- Number of ants (*No_of_ants*): This is also the maximum number of complete candidate rules constructed and pruned during an iteration of the WHILE loop of Algorithm I, since each ant is associated with a single rule. In each iteration the best candidate rule found is considered a discovered rule. The larger *No_of_ants*, the more candidate rules are evaluated per iteration, but the slower the system is.
- Minimum number of cases per rule (*Min_cases_per_rule*): Each rule must cover at least *Min_cases_per_rule* cases, to enforce at least a certain degree of generality in the discovered rules. This helps to avoid an overfitting of 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.
- 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 of the WHILE loop of Algorithm I is therefore stopped, and another iteration is started.

In all the experiments reported in Subsections C, D and E 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 could be tried in a future research. It is interesting to note that even the above non-optimized parameters' setting has produced quite good results, as will be shown later. In addition, the fact that Ant-Miner parameters were not optimized for the data sets used in the experiments makes the comparison with CN2 fair, since we used the default, non-optimized parameters for CN2 as well.

In any case, we also report in Subsection F the results of some experiments evaluating the robustness of Ant-

Miner to changes in some parameter settings.

There is a caveat in the interpretation of the value of *No_of_ants*. Recall that this parameter defines the maximum number of ants for each iteration of the WHILE loop of Algorithm I. In practice much fewer ants are necessary to complete an iteration, since an iteration is considered finished when *No_rules_converg* successive ants converge to the same rule. In the experiments reported here, the actual number of ants per iteration was around 1500, rather than 3000.

C. Comparing Ant-Miner with CN2

We have evaluated the performance of Ant-Miner by comparing it with CN2 [5] [6], a well-known classification-rule discovery algorithm. In essence, CN2 searches for a rule list in an incremental fashion. It discovers one rule at a time. Each time it discovers a rule it adds that rule to the end of the list of discovered rules, removes the cases covered by that rule from the training set and calls again the procedure to discover another rule for the remaining training cases. Note that this strategy is also used by Ant-Miner. In addition, both Ant-Miner and CN2 construct a rule by starting with an empty rule and incrementally add one term at a time to the rule.

However, the rule construction procedure is very different in the two algorithms. CN2 uses a beam search to construct a rule. At each search iteration CN2 adds all possible terms to the current partial rules, evaluates each partial rule and retains only the best *b* partial rules, where *b* is the beam width. This process is repeated until a stopping criterion is met. Then, only the best rule, among the *b* rules currently kept by the beam search, is returned as a discovered rule.

In CN2 there is no mechanism to allow the quality of a discovered rule to be used as a feedback for constructing other rules. This feedback (using the mechanism of pheromone) is the major characteristic of ACO algorithms, and can be considered the main difference between Ant-Miner and CN2. In addition, Ant-Miner performs a stochastic search, whereas CN2 performs a deterministic search.

A comparison between Ant-Miner and CN2 is particularly interesting because both algorithms discover an ordered rule list. In contrast, most of the well-known algorithms for classification-rule discovery, such as C4.5, discover an unordered rule set. In other words, since both Ant-Miner and CN2 discover rules expressed in the same knowledge representation, differences in their performance reflect differences in their search strategies. In data mining and machine learning terminology, one can say that both algorithms have the same representation

bias, but different search (or preference) biases.

All the results of the comparison were obtained using a Pentium II PC with clock rate of 333 MHz and 128 MB of main memory. Ant-Miner was developed in C language and it took about the same processing time as CN2 (on the order of seconds for each data set) to obtain the results.

The comparison was carried out across two criteria, namely the predictive accuracy of the discovered rule lists and their simplicity. Predictive accuracy was measured by a well-known 10-fold cross-validation procedure [24]. In essence, each data set is divided into 10 mutually exclusive and exhaustive partitions and the algorithm is run once for each partition. Each time a different partition is used as the test set and the other 9 partitions are grouped together and used as the training set. The predictive accuracies (on the test set) of the 10 runs are then averaged and reported as the predictive accuracy of the discovered rule list.

The results comparing the predictive accuracy of Ant-Miner and CN2 are reported in Table II. The numbers right after the "±" symbol are the standard deviations of the corresponding predictive accuracies rates. As shown in this table, Ant-Miner discovered rules with a better predictive accuracy than CN2 in four data sets, namely Ljubljana breast cancer, Wisconsin breast cancer, dermatology, and Cleveland heart disease. In two data sets, namely Wisconsin breast cancer and Cleveland heart disease, the difference in predictive accuracy between the two algorithms was quite small. In two data sets, Ljubljana breast cancer and dermatology, Ant-Miner was significantly more accurate than CN2 – that is, the corresponding predictive accuracy intervals (taking into account the standard deviations) do not overlap.

In one data set, hepatitis, the predictive accuracy was the same in both algorithms.

On the other hand, CN2 discovered rules with a better predictive accuracy than Ant-Miner in the tic-tac-toe data set. In this data set the difference was quite large.

TABLE II: PREDICTIVE ACCURACY OF ANT-MINER AND OF CN2 AFTER THE 10-FOLD CROSS-VALIDATION PROCEDURE. THE NUMBERS RIGHT AFTER THE "±" SYMBOL ARE THE STANDARD DEVIATIONS OF THE CORRESPONDING PREDICTIVE ACCURACIES RATES

Data Set	Ant-Miner's predictive accuracy (%)	CN2's Predictive accuracy (%)
Ljubljana breast cancer	75.28 ± 2.24	67.69 ± 3.59
Wisconsin breast cancer	96.04 ± 0.93	94.88 ± 0.88
tic-tac-toe	73.04 ± 2.53	97.38 ± 0.52
dermatology	94.29 ± 1.20	90.38 ± 1.66
hepatitis	90.00 ± 3.11	90.00 ± 2.50
Cleveland heart disease	59.67 ± 2.50	57.48 ± 1.78

We now turn to the results concerning the simplicity of the discovered rule list, measured, as usual in the literature, by the number of discovered rules and the average number of terms (conditions) per rule. The results comparing the simplicity of the rule lists discovered by Ant-Miner and by CN2 are reported in Table III.

An important observation is that for all six data sets the rule list discovered by Ant-Miner was simpler – that is, it had a smaller number of rules and terms – than the rule list discovered by CN2. In four out of the six data sets the difference between the number of rules discovered by Ant-Miner and CN2 is quite large, as follows.

In the Ljubljana breast cancer data set Ant-Miner discovered a compact rule list with 7.1 rules and 1.28 terms per rule, whereas CN2 discovered a rule list on the order of 10 times larger than this, having 55.4 rules and 2.21 conditions per rule.

In the Wisconsin breast cancer, Tic-tac-toe and Cleveland heart disease data sets Ant-Miner discovered rule lists with 6.2, 8.5 and 9.5 rules respectively, whereas CN2 discovered rule lists with 18.6, 39.7 and 42.4 rules respectively.

In the other two data sets, namely dermatology and hepatitis, Ant-Miner discovered rule lists simpler than the rule lists found by CN2, although in these two data sets the difference was not as large as in the other four data sets.

TABLE III: SIMPLICITY OF RULE LISTS DISCOVERED BY ANT-MINER AND BY CN2: FOR EACH DATA SET, THE TWO FIRST COLUMNS SHOW THE NUMBER OF DISCOVERED RULES (AND THEIR STANDARD DEVIATION), AND THE TWO OTHER SHOW RELATIVE NUMBER OF TERMS PER RULE

	No. of rules		No. of terms / No. of rules	
Data Set	Ant-Miner	CN2	Ant-Miner	CN2
Ljubljana breast cancer	7.10 ± 0.31	55.40 ± 2.07	1.28	2.21
Wisconsin breast cancer	6.20 ± 0.25	18.60 ± 0.45	1.97	2.39
tic-tac-toe	8.50 ± 0.62	39.70 ± 2.52	1.18	2.90
dermatology	7.30 ± 0.15	18.50 ± 0.47	3.16	2.47
hepatitis	3.40 ± 0.16	7.20 ± 0.25	2.41	1.58
Cleveland heart disease	9.50 ± 0.92	42.40 ± 0.71	1.71	2.79

Taking into account both the predictive accuracy and rule list simplicity criteria, the results of our experiments can be summarized as follows. Concerning classification accuracy, Ant-Miner obtained results somewhat better than CN2 in four of the six data sets, whereas CN2 obtained a result much better than Ant-Miner in the Tic-tac-toe

data set. In one data set both algorithms obtained the same predictive accuracy. Therefore, overall one can say that the two algorithms are roughly competitive in terms of predictive accuracy, even though the superiority of CN2 in the tic-tac-toe is more significant than the superiority of Ant-Miner in four data sets.

Concerning the simplicity of discovered rules, overall Ant-Miner discovered rule lists much simpler (smaller) than the rule lists discovered by CN2.

This seems a good trade-off, since in many data mining applications the simplicity of a rule list/set tends to be even more important than its predictive accuracy. Actually, there are several classification-rule discovery algorithms that were explicitly designed to improve rule set simplicity, even at the expense of reducing the predictive accuracy [1] [3] [4].

D. The Effect of Pruning

In order to analyze the influence of rule pruning in the overall Ant-Miner algorithm, Ant-Miner was also run without rule pruning. All the other procedures of Ant-Miner, as described in Algorithm I, were kept unchanged. To make a fair comparison between Ant-Miner with and without pruning, the experiments without pruning used the same parameters setting specified in Subsection B, and they also consisted of a 10-fold cross-validation procedure applied to the six data sets described in Table I. The results of Ant-Miner without rule pruning are reported in Table IV.

TABLE IV: RESULTS FOR ANT-MINER WITHOUT RULE PRUNING: THE SECOND COLUMN SHOWS THE PREDICTIVE ACCURACY, THE THIRD COLUMN THE NUMBER OF DISCOVERED RULES AND THE FORTH COLUMN THE RELATIVE NUMBER OF TERMS PER RULE

Data Set	Predictive Accuracy (%)	No. of rules	No. of terms / No. of rules
Ljubljana breast cancer	70.69 ± 3.87	19.60 ± 0.22	3.25
Wisconsin breast cancer	95.74 ± 0.74	22.8 ± 0.20	5.72
tic-tac-toe	76.83 ± 2.27	68.8 ± 0.32	3.47
dermatology	83.05 ± 1.94	25.9 ± 0.31	16.86
Hepatitis	92.5 ± 2.76	6.8 ± 0.13	6.01
Cleveland heart disease	54.82 ± 2.56	21.8 ± 0.20	4.32

Let us first analyze the influence of rule pruning in the predictive accuracy of Ant-Miner. Comparing the second column of Table IV with the second column of Table II it can seen that Ant-Miner without pruning obtained a lower predictive accuracy than Ant-Miner with pruning in three data sets, namely Ljubljana breast cancer, dermatology, and Cleveland heart disease. The reduction of predictive accuracy was particularly strong in the dermatology data set, where Ant-Miner without pruning obtained an predictive accuracy of 83.05%, whereas Ant-Miner with pruning obtained 94.29%.

On the other hand, Ant-Miner without pruning obtained an predictive accuracy somewhat larger than Ant-Miner with pruning in two data sets, namely tic-tac-toe and hepatitis. In the other data set, Wisconsin breast cancer, the predictive accuracy with and without pruning was almost equal.

Therefore, in the six data sets used in the experiments, rule pruning seems to be beneficial more often than it is harmful, concerning predictive accuracy. The fact that rule pruning reduces predictive accuracy in some data sets is not surprising. It stems from the fact that rule pruning is a form of inductive bias [20] [21] [22], and any inductive bias is suitable for some data sets and unsuitable for others.

The results concerning the simplicity of the discovered rule list are reported in the third and fourth columns of Table IV, and should be compared with the second and fourth columns of Table III. It can be seen that in all data sets the rule list discovered by Ant-Miner without pruning was much more complex – that is, it had a higher number of rules and terms – than the rule list discovered by Ant-Miner with pruning. So, rule pruning is essential to improve the comprehensibility of the rules discovered by Ant-Miner.

E. The Influence of Pheromone

We have also analyzed the role played by pheromone in the overall Ant-Miner performance. This was done by setting the amount of pheromone to a constant value for all pairs of attributes-values during an entire run of Ant-Miner. More precisely, we set $\tau_{ij}(t) = 1$, $\forall i,j,t$, that is, the pheromone updating procedure was removed from Algorithm I. Recall that an ant chooses a term to add to its current partial rule based on Equation (1). Since now all candidate terms (attribute-value pairs) have their amount of pheromone set to 1, ants choose $term_{ij}$ with probability η_{ij} – that is, Ant-Miner's search for rules is guided only by the information-theoretic heuristic defined in Equation (3). All other procedures of Ant-Miner, as described in Algorithm I, were kept unaltered.

Again, to make a fair comparison between Ant-Miner with pheromone and Ant-Miner without pheromone, the

experiments without pheromone also used the parameters setting specified in Subsection B, and they also consisted of a 10-fold cross-validation procedure applied to the six data sets described in Table I.

The results of Ant-Miner without pheromone are reported in Table V. By comparing the second column of Table V with the second column of Table II one can see that Ant-Miner without pheromone consistently achieved a lower predictive accuracy than Ant-Miner with pheromone in five of the six data sets. The only exception was the Ljubljana breast cancer data set, where the predictive accuracy was almost the same for both versions of Ant-Miner.

TABLE V: RESULTS OF ANT-MINER WITHOUT PHEROMONE: THE SECOND COLUMN SHOWS THE PREDICTIVE ACCURACY, THE THIRD COLUMN THE NUMBER OF DISCOVERED RULES AND THE FORTH COLUMN THE RELATIVE NUMBER OF TERMS PER RULE

Data Set	Predictive Accuracy (%)	No. of rules	No. of terms / No. of rules
Ljubljana breast cancer	75.17 ± 3.22	6.60 ± 0.30	1.13
Wisconsin breast cancer	93.84 ± 0.82	5.90 ± 0.23	1.52
tic-tac-toe	67.37 ± 2.96	9.30 ± 0.56	1.07
dermatology	88.16 ± 2.68	7.50 ± 0.16	2.97
hepatitis	85.00 ± 3.63	3.30 ± 0.15	1.54
Cleveland heart disease	54.70 ± 3.12	9.30 ± 0.26	1.60

With respect to the simplicity of the discovered rule lists, there is not much difference between Ant-Miner with pheromone and Ant-Miner without pheromone, as can be seen by comparing the third and fourth columns of Table V with the second and fourth columns of Table III.

Overall, one can conclude that the use of the pheromone updating procedure is important to improve the predictive accuracy of the rules discovered by Ant-Miner – at least when pheromone updating interacts with the information-theoretic heuristic defined by Equation (3). Furthermore, the improvement of predictive accuracy associated with pheromone updating is obtained without unduly sacrificing the simplicity of the discovered rules.

F. Robustness to Parameter Setting

Finally, we have also investigated Ant-Miner's robustness to different settings of some parameters. As described

in Subsection B, Ant-Miner has the following four parameters:

- Number of Ants (*No_of_ants*).
- Minimum number of cases per rule (*Min_cases_per_rule*).
- Maximum number of uncovered cases in the training set (*Max_uncovered_cases*).
- Number of rules used to test convergence of the ants (*No_rules_converg*).

Among these four parameters, we consider that the two most important ones are *Min_cases_per_rule* and *Max_uncovered_cases*, because they are directly related to the degree of generality of the rules discovered by Ant-Miner which, in turn, can have a potentially significant effect on the accuracy and simplicity of the discovered rules.

Therefore, in order to analyze the influence of the setting of these two parameters on the performance of Ant-Miner, Ant-Miner was run with different settings for these two parameters. More precisely, in the experiments three different values for each of these two parameters were used, namely 5, 10 and 15. All the corresponding nine combinations of parameter values were considered, so that Ant-Miner was run once for each combination of values of *Min_cases_per_rule* and *Max_uncovered_cases*. Note that one of these nine combinations of parameter values, namely *Min_cases_per_rule* = 10 and *Max_uncovered_cases* = 10, was the combination used in all the experiments reported in the previous subsections. The results of these experiments are reported in Table VI.

TABLE VI: PERFORMANCE OF ANT-MINER WITH DIFFERENT PARAMETER SETTINGS. THE FIRST COLUMN INDICATES THE VALUES OF THE TWO PARAMETERS UNDER STUDY (*MIN_CASES_PER_RULE* AND *MAX_UNCOVERED_CASES*). EACH OF THE OTHER COLUMNS REFERS TO A SPECIFIC DATA SET. EACH CELL INDICATES THE PREDICTIVE ACCURACY OF ANT-MINER FOR THE CORRESPONDING COMBINATION OF PARAMETER VALUES AND DATA SET. THE LAST ROW SHOWS THE PREDICTIVE ACCURACY AVERAGES OF ALL COMBINATIONS OF VALUES

	Ljubl.	Wisc.	tic-tac.	derm.	hepat.	Clev.
5, 5	77.06	95.75	74.01	93.93	91.25	55.12
5, 10	74.76	95.03	73.29	96.05	91.25	58.79
5, 15	74.48	94.74	75.75	95.30	78.75	57.39
10, 5	75.50	94.58	72.00	94.05	92.50	57.21
10, 10	75.05	94.45	76.00	94.39	91.25	59.42
10, 15	75.42	95.92	73.80	92.62	80.00	58.12
15, 5	74.08	95.61	73.35	94.39	77.50	58.03
15, 10	74.16	95.17	72.08	95.02	77.50	59.97
15, 15	75.35	95.17	73.53	94.96	76.25	59.15
Average	75.10	95.16	73.76	94.52	84.03	58.13

As can be observed in Table VI, Ant-Miner seems to be quite robust to different parameter settings in almost all data sets. Indeed, one can observe that the average results in the last row of Table VI are similar to the results of the second column of Table II. The only exception is the hepatitis data set, where the predictive accuracy of Ant-Miner can vary from 76.25% to 92.50%, depending on the values of the parameters.

As expected, no single combination of parameter values is the best for all data sets. Indeed, each combination of parameter values has some influence in the inductive bias of Ant-Miner, and it is a well-known fact in data mining and machine learning that there is no inductive bias that is the best for all data sets [20] [21] [22].

We emphasize that the results of Table VI are reported here only to analyze the robustness of Ant-Miner to variations in two of its parameters. We have *not* used the results of Table VI to "optimize" the performance of Ant-Miner for each data set, in order to make the comparison between Ant-Miner and CN2 fair – as discussed in Subsection B. Of course, we could optimize the parameters of both Ant-Miner and CN2 for each data set, but this optimization would have to be done by measuring predictive accuracy on a validation set separated from the test set – otherwise the results would be (unfairly) optimistically-biased. This point is left for future research.

G. Analysis of Ant-Miner's Computational Complexity

In this section we present an analysis of Ant-Miner's computational complexity. This analysis is divided into three parts, namely: preprocessing; a single iteration of the WHILE loop of Algorithm I; the entire WHILE loop of Algorithm I. Then we combine the results of these three steps in order to determine the computational complexity of an entire execution of Ant-Miner:

- Computational complexity of preprocessing – The values of all η_{ij} are pre-computed, as a preprocessing step, and kept fixed throughout the algorithm. These values can be computed in a single scan of the training set, so the time complexity of this step is $O(n \bullet a)$, where n is the number of cases and a is the number of attributes.
- Computational complexity of a single iteration of the WHILE loop of Algorithm I – Each iteration starts by initializing pheromone, that is, specifying the values of all $\tau_{ij}(t_0)$. This step takes $O(a)$, where a is the number of attributes. Strictly speaking, it takes $O(a \bullet v)$, where v is the number of values per attribute. However, the current version of Ant-Miner copes only with categorical attributes. Hence, we can assume that each attribute can take only a small number of values, so that v is a relatively small integer for any data set. Therefore, the formula can be simplified to $O(a)$.

Next we have to consider the REPEAT loop. Let us first consider a single iteration of this loop, that is, a single ant, and later on the entire REPEAT loop. The major steps performed for each ant are: (a) rule construction, (b) rule evaluation, (c) rule pruning and (d) pheromone updating. The computational complexities of these steps are as follows:

(a) Rule Construction – The choice of a condition to be added to the current rule requires that an ant considers all possible conditions. The values of η_{ij} and $\tau_{ij}(t)$ for all conditions have been pre-computed. Therefore, this step takes O(a). (Again, strictly speaking it has a complexity of O($a \bullet v$), but we are assuming that v is a small integer, so that the formula can be simplified to O(a).) In order to construct a rule, an ant will choose k conditions. Note that k is a highly variable number, depending on the data set and on previous rules constructed by other ants. In addition, $k \leq a$ (since each attribute can occur at most once in a rule). Hence, rule construction takes O($k \bullet a$).

(b) Rule Evaluation – This step consists of measuring the quality of a rule, as given by Equation (5). This requires matching a rule with k conditions with a training set with N cases, which takes O($k \bullet n$).

(c) Rule Pruning – The first pruning iteration requires the evaluation of k new candidate rules (each one obtained by removing one of the k conditions from the unpruned rule). Each of these rule evaluations takes on the order of ($n \bullet (k$-1)) operations. Thus, the first pruning iteration takes on the order of ($n \bullet (k$-1) $\bullet k$) operations, this is, O($n \bullet k^2$). The second pruning iteration takes ($n \bullet (k$-2) $\bullet (k$-1)) operations, and so on. The entire rule pruning process is repeated at most k times, so rule pruning takes at most: $n \bullet (k$-1) $\bullet k$ + $n \bullet (k$-2) $\bullet (k$-1) + $n \bullet (k$-3) $\bullet (k$-2) + ... + $n \bullet (1) \bullet (2)$, this is, O($k^3 \bullet n$).

(d) Pheromone Updating – This step consists of increasing the pheromone of terms used in the rule, which takes O(k), and decreasing the pheromone of unused terms, which takes O(a). Since $k \leq a$, pheromone update takes O(a).

Adding up the results derived in (a), (b) (c) and (d), a single iteration of the REPEAT loop, corresponding to a single ant, takes: O($k \bullet a$) + O($n \bullet k$) + O($n \bullet k^3$) + O(a), which collapses to: O($k \bullet a + k^3 \bullet n$).

In order to derive the computational complexity of a single iteration of the WHILE loop of Algorithm I, the previous result, O($k \bullet a + n \bullet k^3$), has to be multiplied by z, where the z is the number of ants. Hence, a single iteration of the WHILE loop takes: O($z \bullet [k \bullet a + n \bullet k^3]$).

- In order to derive the computational complexity for the entire WHILE loop we have to multiply $O(z \bullet [k \bullet a + n \bullet k^3])$ by r, the number of discovered rules, which is highly variable for different data sets. Finally, we then add the computational complexity of the preprocessing step, as explained earlier. Therefore, the computational complexity of Ant-Miner as a whole is:

$$O(r \bullet z \bullet [k \bullet a + k^3 \bullet n] + a \bullet n).$$

It should be noted that this complexity depends very much on the values of k, the number of conditions per rule, and r, the number of rules. The values of k and r vary a lot from data set to data set, depending on the contents of the data set.

At this point it is useful to make a distinction between the computational complexity of Ant-Miner in the worst case and in the average case. In the worst case the value of k is equal to a, so the formula for worst-case computational complexity is:

$$O(r \bullet z \bullet a^3 \bullet n)$$

However, we emphasize that this worst case is very unlikely to occur, and in practice the time taken by Ant-Miner tends to be much shorter than that suggested by the worst-case formula. This is mainly due to three reasons.

First, in the previous analysis of step (c) – rule pruning – the factor $O(k^3 \bullet n)$ was derived because it was considered that the pruning process can be repeated k times for all rules, which seems unlikely. Second, the above worst-case analysis considered that $k = a$, which is very unrealistic. In the average case, k tends to be much smaller than a. Evidence supporting this claim is provided in Table VII. This table reports, for each data set used in the experiments of Section V, the value of the ratio k/a – where k is the average number of conditions in the discovered rules and a is the number of attributes in the data set. Note that the value of this ratio is on average just 14%.

TABLE VII: RATIO OF *K*/*A* FOR THE DATA SETS USED IN THE EXPERIMENTS OF SECTION V, WHERE *K* IS THE AVERAGE NUMBER OF CONDITIONS IN THE DISCOVERED RULES AND *A* IS THE NUMBER OF ATTRIBUTES IN THE DATA SET. THE LAST ROW SHOWS THE AVERAGE *K*/*A* FOR ALL DATA SETS

Data set	*k*/*a*
Ljubljana breast cancer	0.14
Wisconsin breast cancer	0.22
tic-tac-toe	0.13
dermatology	0.09
hepatitis	0.13
Cleveland heart disease	0.13
Average	0.14

Third, the previous analysis has implicitly assumed that the computational complexity of an iteration of the WHILE loop of Algorithm I is the same for all the r iterations constituting the loop. This is a pessimistic assumption. At the end of each iteration, all the cases correctly covered by the just-discovered rule are removed from the current training set. Hence, as the iteration counter increases, Ant-Miner will access new training subsets that have fewer and fewer cases (i.e., smaller and smaller values of n), which considerably reduces the computational complexity of Ant-Miner.

VI. Conclusions and Future Work

This work has proposed an algorithm for rule discovery called Ant-Miner. The goal of Ant-Miner is to discover classification rules in data sets. The algorithm is based both on research on the behavior of real ant colonies and on data mining concepts and principles.

We have compared the performance of Ant-Miner and the well-known CN2 algorithm in six public domain data sets. The results showed that, concerning predictive accuracy, Ant-Miner obtained somewhat better results in four data sets, whereas CN2 obtained a considerably better result in one data set. In the remaining data set both algorithms obtained the same predictive accuracy. Therefore, overall one can say that Ant-Miner is roughly equivalent to CN2 with respect to predictive accuracy.

On the other hand, Ant-Miner has consistently found much simpler (smaller) rule lists than CN2. Therefore, Ant-Miner seems particularly advantageous when it is important to minimize the number of discovered rules and rule terms (conditions), in order to improve comprehensibility of the discovered knowledge. It can be argued that this point is important in many (probably most) data mining applications, where discovered knowledge will be shown to a human user as a support for intelligent decision making, as discussed in the introduction.

Two important directions for future research are as follows. First, it would be interesting to extend Ant-Miner to cope with continuous attributes, rather than requiring that this kind of attribute be discretized in a preprocessing step.

Second, it would be interesting to investigate the performance of other kinds of heuristic function and pheromone updating strategy.

ACKNOWLEDGEMENTS

Authors would like to thank Dr. Marco Dorigo and the anonymous reviewers of this paper for their useful comments and suggestions.

REFERENCES

[1] M. Bohanec and I. Bratko, "Trading accuracy for simplicity in decision trees," *Machine Learning*, vol. 15, pp. 223-250, 1994.

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

[3] L. A. Brewlow and D. W. Aha, "Simplifying decision trees: a survey," *The Knowledge Engineering Review*, vol. 12, no. 1, pp. 1-40, 1997.

[4] J. Catlett, "Overpruning large decision trees," In: *Proceedings International Joint Conference on Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann, 1991.

[5] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine Learning*, vol. 3, pp. 261-283, 1989.

[6] P. Clark and R. Boswell, "Rule induction with CN2: some recent improvements," In: *Proceedings of the European Working Session on Learning (EWSL-91)*, Lecture Notes in Artificial Intelligence. Berlin, Germany: Springer-Verlag, vol. 482, pp. 151-163, 1991.

[7] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, New York, NY: John Wiley & Sons, 1991.

[8] V. Dhar, D. Chou and F. Provost, "Discovering interesting patterns for investment decision making with GLOWER - a genetic learner overlaid with entropy reduction," *Data Mining and Knowledge Discovery*, vol. 4, no 4, pp. 251-280, 2000.

[9] M. Dorigo, A. Colorni and V. Maniezzo, "The Ant System: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, no. 1, pp. 29-41, 1996.

[10] M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic," In: *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover Eds. London, UK: McGraw Hill, pp. 11-32, 1999.

[11] M. Dorigo, G. Di Caro and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137-172, 1999.

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

[13] A. A. Freitas and S. H. Lavington, *Mining Very Large Databases with Parallel Processing*, London, UK: Kluwer, 1998.

[14] A. A. Freitas, "Understanding the crucial role of attribute interaction in data mining," *Artificial Intelligence Review,* vol.16, no 3, pp. 177-199, 2001.

[15] R. Kohavi and M. Sahami, "Error-based and entropy-based discretization of continuous features," In: *Proceedings of the 2nd International Conference Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press, pp. 114-119, 1996.

[16] H. S. Lopes, M. S. Coutinho and W. C. Lima, “An evolutionary approach to simulate cognitive feedback learning in medical domain,” In: *Genetic Algorithms and Fuzzy Logic Systems*: *Soft Computing Perspectives*, E. Sanchez, T. Shibata and L.A. Zadeh (Eds.) Singapore: World Scientific, pp. 193-207, 1998.

[17] N. Monmarché, “On data clustering with artificial ants,” In: *Data Mining with Evolutionary Algorithms*, Research Directions – Papers from the AAAI Workshop, A.A. Freitas (Ed.) Menlo Park, CA: AAAI Press, pp. 23-26, 1999.

[18] J. R. Quinlan, “Generating production rules from decision trees,” In: *Proceedings of the International Joint Conference on Artificial Intelligence*. San Francisco: CA: Morgan Kaufmann, pp. 304-307, 1987.

[19] J. R. Quinlan, *C4.5: Programs for Machine Learning*, San Francisco, CA: Morgan Kaufmann, 1993.

[20] R. B. Rao, D. Gordon and W. Spears, “For every generalization action, is there really an equal and opposite reaction? Analysis of the conservation law for generalization performance”, In: *Proceedings of the 12th International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann, pp. 471-479, 1995.

[21] C. Schaffer, “Overfitting avoidance as bias,” *Machine Learning*, vol. 10, pp. 153-178, 1993.

[22] C. Schaffer, “A conservation law for generalization performance”, In: *Proceedings of the 11th International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann, pp. 259-265, 1994.

[23] T. Stützle and M. Dorigo, “ACO algorithms for the traveling salesman problem,” In: *Evolutionary Algorithms in Engineering and Computer Science*, K. Miettinen, M. Makela, P. Neittaanmaki and J. Periaux (Eds.) New York, NY: Wiley, pp. 163-183, 1999.

[24] S. M. Weiss and C. A. Kulikowski, *Computer Systems that Learn*, San Francisco, CA: Morgan Kaufmann, 1991.