

PART FOUR:

**ANT COLONY OPTIMIZATION
AND IMMUNE SYSTEMS**

Chapter X

An Ant Colony Algorithm for Classification Rule Discovery

Rafael S. Parpinelli and Heitor S. Lopes
CEFET-PR, Curitiba, Brazil

Alex A. Freitas
PUC-PR, Curitiba, Brazil

This work proposes an algorithm for rule discovery called Ant-Miner (Ant Colony-based Data Miner). The goal of Ant-Miner is to extract classification rules from data. The algorithm is based on recent research on the behavior of real ant colonies as well as in some data mining concepts. We compare the performance of Ant-Miner with the performance of the well-known C4.5 algorithm on six public domain data sets. The results provide evidence that: (a) Ant-Miner is competitive with C4.5 with respect to predictive accuracy; and (b) The rule sets discovered by Ant-Miner are simpler (smaller) than the rule sets discovered by C4.5.

INTRODUCTION

In essence, the classification task consists of associating each case (object, or record) to one class, out of a set of predefined classes, based on the values of some attributes (called predictor attributes) for the case.

There has been a great interest in the area of data mining, in which the general goal is to discover knowledge that is not only correct, but also comprehensible and interesting for the user (Fayyad, Piatetsky-Shapiro, & Smyth, 1996; Freitas &

Lavington, 1998). Hence, the user can understand the results produced by the system and combine them with their own knowledge to make a well-informed decision, rather than blindly trusting on a system producing incomprehensible results.

In data mining, discovered knowledge is often represented in the form of IF-THEN prediction (or classification) rules, as follows: *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>*, such as *<Gender = female>*.

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.

To the best of our knowledge the use of Ant Colony algorithms (Dorigo, Coloni, & Maniezzo, 1996) as a method for discovering classification rules, in the context of data mining, is a research area still unexplored by other researchers. Actually, the only Ant Colony algorithm developed for data mining that we are aware of is an algorithm for clustering (Monmarche, 1999), which is, of course, a data mining task very different from the classification task addressed in this paper. Also, Cordón et al. (2000) have proposed another kind of Ant Colony Optimization application that learns fuzzy control rules, but it is outside the scope of data mining.

We believe the development of Ant Colony algorithms for data mining is a promising research area, due to the following rationale. An Ant Colony system involves 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 Ant Colony system has the ability to perform a flexible, robust search for a good combination of logical conditions involving values of the predictor attributes.

This chapter is organized as follows. The second section presents an overview of real Ant Colony systems. The third section describes in detail artificial Ant Colony systems. The fourth section introduces the Ant Colony system for discovering classification rules proposed in this work. The fifth section reports on computational results evaluating the performance of the proposed system. Finally, the sixth section concludes the chapter.

SOCIAL INSECTS AND REAL ANT COLONIES

Insects that live in colonies, such as ants, bees, wasps and termites, follow their own agenda of tasks independent from one another. However, when these insects act as a whole community, they are capable of solving complex problems in their daily lives, through mutual cooperation (Bonabeau, Dorigo, & Theraulaz, 1999). Problems such as selecting and picking up materials, and finding and storing foods, which require sophisticated planning, are solved by insect colonies without any kind of supervisor or controller. This collective behavior which emerges from a

group of social insects has been called “swarm intelligence”.

Ants are capable of finding the shortest route between a food source and the nest without the use of visual information, and they are also capable of adapting to changes in the environment (Dorigo et al., 1996).

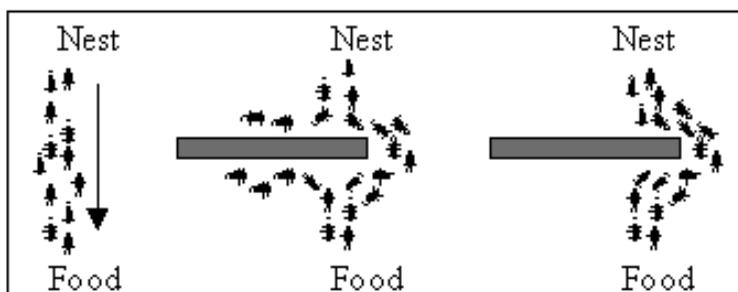
One of the main problems studied by ethnologists is to understand how almost-blind animals, such as ants, manage to find the shortest route between their colony and a food source. It was discovered that, in order to exchange information about which path should be followed, ants communicate with one another by means of pheromone trails. The movement of ants leaves a certain amount of pheromone (a chemical substance) on the ground, marking the path with a trail of this substance. The collective behavior which emerges is a form of autocatalytic behavior, i.e. the more ants follow a trail, the more attractive this trail becomes to be followed by other ants. This process can be described as a loop of positive feedback, where the probability of an ant choosing a path increases as the number of ants that already passed by that path increases (Bonabeau et al., 1999), (Dorigo et al., 1996), (Stutzle & Dorigo 1999), (Dorigo, Di Caro, & Gambardella, 1999).

The basic idea of this process is illustrated in Figure 1. In the left picture the ants move in a straight line to the food. The middle picture illustrates what happens soon after an obstacle is put in the path between the nest and the food. In order to go around the obstacle, at first each ant chooses to turn left or right at random (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. However, the ants that, by chance, choose to turn left will reach the food sooner, whereas the ants that go around the obstacle turning right will follow a longer path, and so will take longer 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 around the obstacle, as shown in the right picture.

Artificial Ant Colony Systems

An artificial Ant Colony System (ACS) is an agent-based system which simulates the natural behavior of ants and develops mechanisms of cooperation and

Figure 1: Ants finding the shortest path around an obstacle



learning. ACS was proposed by Dorigo et al. (1996) as a new heuristic to solve combinatorial-optimization problems. This new heuristic, called Ant Colony Optimization (ACO), has been shown to be both robust and versatile – in the sense that it can be applied to a range of different combinatorial optimization problems. In addition, ACO is a population-based heuristic. This is advantageous because it allows the system to use a mechanism of positive feedback between agents as a search mechanism. Recently there has been a growing interest in developing rule discovery algorithms based on other kinds of population-based heuristics – mainly evolutionary algorithms (Freitas, 2001).

Artificial ants are characterized as agents that imitate the behavior of real ants. However, it should be noted that an artificial ACS has some differences in comparison with a real ACS, as follows (Dorigo et al., 1996):

- Artificial ants have memory;
- They are not completely blind;
- They live in an environment where time is discrete.

On the other hand, an artificial ACS has several characteristics adopted from real ACS:

- 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 in each path.

The key idea is that, when a given ant has to choose between two or more paths, the path that was more frequently chosen by other ants in the past will have a greater probability of being chosen by the ant. Therefore, trails with greater amount of pheromone are synonyms of shorter paths.

In essence, an ACS iteratively performs a loop containing two basic procedures, namely:

- i) a procedure specifying how the ants construct/ modify solutions of the problem being solved; and
- ii) a procedure to update the pheromone trails.

The construction/modification of a solution is performed in a probabilistic way. The probability of adding a new item to the current partial solution is given by a function that depends on a problem-dependent heuristic (η) and on the amount of pheromone (τ) deposited by ants on this trail in the past. The updates in the pheromone trail are implemented as a function that depends on the rate of pheromone evaporation and on the quality of the produced solution. To realize an ACS one must define (Bonabeau et al., 1999):

- 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 heuristic;
- A heuristic function (η) that measures the quality of items that can be added to the current partial solution;

- A method to enforce the construction of valid solutions, i.e. solutions that are legal in the real-world situation corresponding to the problem definition;
- A rule for pheromone updating, which specifies how to modify the pheromone trail (τ);
- A probabilistic rule of transition based on the value of the heuristic function (η) and on the contents of the pheromone trail (τ).

ANT-MINER – A NEW ANT COLONY SYSTEM FOR DISCOVERY OF CLASSIFICATIONS RULES

In this section we discuss in detail our proposed Ant Colony System for discovery of classification rules, called Ant-Miner (Ant Colony-based Data Miner). This section is divided into six parts, namely: an overview of Ant-Miner, rule construction, heuristic function, pheromone updating, rule pruning, and system parameters.

An Overview of Ant-Miner

Recall that each ant can be regarded as an agent that incrementally constructs/modify a solution for the target problem. In our case the target problem is the discovery of classification rules. As mentioned in the Introduction, 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>*, 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 can cope 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 an empty rule, i.e. 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 to 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 the term to be added to the current partial rule 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 the next subsections.

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, namely: (a) when whichever term that 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 is 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 a separate subsection later.

When an ant completes its rule and the amount of pheromone in each trail is updated, another ant starts 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.

Figure 2: Overview of Ant-Miner

```

Training Set = all training cases;
WHILE (No. of uncovered cases in the Training Set >
      Max_Uncovered_Cases)
  i = 0;
  REPEAT
    i=i+1;
    Anti incrementally constructs a classification rule;
    Prune the just-constructed rule;
    Update the pheromone of the trail followed by Anti;
  UNTIL (i ≥ No_of_Ants) OR
        (Anti constructed the same rule than the previous
         No_Rules_Converg - 1 Ants)
  Select the best rule among all constructed rules;
  Remove the cases correctly covered by the selected rule
  from the Training Set;
END WHILE

```

Then all cases correctly covered by the discovered rule are removed from the training set, and another iteration is started. Hence, the Ant-Miner 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.

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.

Rule Construction

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).

$$P_{ij}(t) = \frac{\tau_{ij}(t) \cdot \eta_{ij}}{\sum_i^a \sum_j^{b_i} \tau_{ij}(t) \cdot \eta_{ij}}, \forall i \in I \quad (1)$$

where:

- η_{ij} is the value of a problem-dependent heuristic function for $term_{ij}$;
- $\tau_{ij}(t)$ is the amount of pheromone currently available (at time t) in the position i,j of the trail being followed by the ant;
- a is the total number of attributes;
- b_i is the total number of values on the domain of attribute i ;
- I are the attributes i not yet used by the ant.

The problem-dependent heuristic function η_{ij} is a measure of the predictive power of $term_{ij}$. 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. This heuristic function will be explained in detail in the next subsection. For now we just mention that the value

of this function is always the same for a given term, regardless of which terms already occur in the current partial rule and regardless of the path followed by previous ants.

The amount of pheromone $\tau_{ij}(t)$ is also independent of the terms which already occur in the current partial rule, but is entirely dependent on the paths followed by previous ants. Hence, $\tau_{ij}(t)$ incorporates an indirect form of communication between ants, where successful ants leave a “clue” (pheromone) suggesting the best path to be followed by other ants, as discussed earlier. When the first ant starts to build its rule, all trail positions i, j – i.e. all $term_{ij}, \forall i, j$ – have the same amount of pheromone. However, as soon as an ant finishes its path the amounts of pheromone in each position i, j visited by the ant is updated, as will be explained in detail in a separate subsection later. Here we just mention the basic idea: the better the quality of the rule constructed by the ant, the higher the amount of pheromone added to the trail positions visited by the ant. Hence, with time the “best” trail positions to be followed – i.e. 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.

The $term_{ij}$ chosen to be added to the current partial rule is the term with the highest value of equation (1) subject to two restrictions. The first restriction is that the attribute i cannot occur yet in the current partial rule. Note that to satisfy this restriction the ants must “remember” which terms (attribute-value pairs) are contained in the current partial rule. This small amount of “memory” is one of the differences between artificial ants and natural ants, as discussed earlier.

The second restriction is that a $term_{ij}$ cannot be added to the current partial rule if this makes the extended partial rule cover less than a predefined minimum number of cases, called the *Min_cases_per_rule* threshold, as mentioned above.

Note that the above procedure constructs a rule antecedent, but it does not specify which rule consequent will be assigned to the rule. This decision is made only after the rule antecedent is completed. More precisely, 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.

In the next two subsections we discuss in detail the heuristic function and the pheromone updating procedure.

Heuristic Function

For each term that can be added to the current rule, Ant-Miner computes 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 (Cover & Thomas, 1991). More precisely, the value of the heuristic function for a term 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 given by equation (2).

$$\text{info}T_{ij} = - \sum_{w=1}^k \left(\frac{\text{freq}T_{ij}^w}{|T_{ij}|} \right) * \log_2 \left(\frac{\text{freq}T_{ij}^w}{|T_{ij}|} \right) \quad (2)$$

where:

- k is the number of classes;
- $|T_{ij}|$ is the total number of cases in partition T_{ij} (partition containing the cases where attribute A_i has value V_{ij});
- $\text{freq}T_{ij}^w$ is the number of cases in partition T_{ij} that have class w .

The higher the value of $\text{info}T_{ij}$, the more uniformly distributed the classes are, and so the lower the predictive power of term_{ij} . We obviously want to choose terms with a high predictive power to be added to the current partial rule. Therefore, the value of $\text{info}T_{ij}$ has the following role in Ant-Miner: the higher the value of $\text{info}T_{ij}$, the smaller the probability of an ant choosing term_{ij} to be added to its partial rule. Before we map this basic idea into a heuristic function, one more point must be considered. It is desirable to normalize the value of the heuristic function, to facilitate its use in a single equation – more precisely, equation (1) – combining both this function and the amount of pheromone. In order to implement this normalization, we use the fact that the value of $\text{info}T_{ij}$ varies in the range:

$$0 \leq \text{info}T_{ij} \leq \log_2(k)$$

where k is the number of classes.

Therefore, we propose the normalized, information-theoretic heuristic function given by equation (3).

$$\eta_{ij} = \frac{\log_2(k) - \text{info}T_{ij}}{\sum_i^a \sum_j^{b_i} \log_2(k) - \text{info}T_{ij}} \quad (3)$$

where:

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

Note that the $\text{info}T_{ij}$ 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, we compute the $\text{info}T_{ij}$ of all term_{ij} , $\forall i, j$, as a preprocessing step.

In order to use the above heuristic function, there are just two minor caveats. First, if the partition T_{ij} is empty, i.e. the value V_{ij} of attribute A_i does not occur in the training set, then we set $\text{info}T_{ij}$ to its maximum value, i.e. $\text{info}T_{ij} = \log_2(k)$. This corresponds to assigning to term_{ij} the lowest possible predictive power.

Second, if all the cases in the partition T_{ij} belong to the same class then $\text{info}T_{ij} = 0$. This corresponds to assigning to term_{ij} the highest possible predictive power.

Rule Pruning

Rule pruning is a commonplace technique in rule induction. As mentioned above, 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 over-fitting to the training data. Another motivation for rule pruning is that it improves the simplicity of the rule, since a shorter rule is in general more easily interpretable by the user than a long rule.

The rule pruning procedure is performed for each ant as soon as the ant completes the construction of its rule. The search strategy of our rule pruning procedure is very similar to the rule pruning procedure suggested by Quinlan (1987), although the rule quality criterion used in the two procedures are very different from each other.

The basic idea is to iteratively remove one-term-at-a-time from the rule while this process improves the quality of the rule. A more detailed description is as follows.

In the first iteration one starts with the full rule. Then one tentatively tries to remove each of the terms of the rule – each one in turn – and computes the quality of the resulting rule, using the quality function defined by formula (5) – to be explained later. (This step may involve re-assigning another class to the rule, since a pruned rule can have a different majority class in its covered cases.) The term whose removal most improves the quality of the rule is effectively removed from the rule, completing the first iteration. In the next iteration one removes 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.

Pheromone Updating

Recall that each $term_{ij}$ corresponds to a position in some path that can be followed by an ant. All $term_{ij}$, $\forall i, j$, are initialized with the same amount of pheromone. In other words, when the system is initialized and 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, as given by equation (4).

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

where:

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

The value returned by this equation is already normalized, which facilitates its

use in a single equation – more precisely, equation (1) – combining both this value and the value of the heuristic function.

Each time an ant completes the construction of a rule (i.e. an ant completes its path) the amount of pheromone in all positions of all paths must be updated. This pheromone updating has two basic ideas, namely:

- (a) The amount of pheromone associated with each $term_{ij}$ occurring in the constructed rule is increased;
- (b) The amount of pheromone associated with each $term_{ij}$ that does not occur in the constructed rule is decreased, corresponding to the phenomenon of pheromone evaporation in real Ant Colony Systems.

Let us elaborate each of these two ideas in turn.

Increasing the Pheromone of Used Terms

Increasing the amount of pheromone associated with each $term_{ij}$ occurring in the constructed rule 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, since that term was beneficial for the current ant. This increase is proportional to the quality of the rule constructed by the ant – i.e. the better the rule, the higher the increase in the amount of pheromone for each $term_{ij}$ occurring in the rule.

The quality of the rule constructed by an ant, denoted by Q , is computed by the formula: $Q = sensitivity \times specificity$ (Lopes, Coutinho, & Lima, 1998), as defined in equation (5).

$$Q = \left(\frac{TruePos}{TruePos + FalseNeg} \right) \times \left(\frac{TrueNeg}{FalsePos + TrueNeg} \right) \quad (5)$$

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 that do not have the class predicted by the rule.

The larger the value of Q , the higher the quality of the rule. Note that Q varies in the range: $0 \leq Q \leq 1$. Pheromone update for a $term_{ij}$ is performed according to equation (6), $\forall i, j$.

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

Hence, for all $term_{ij}$ occurring in the rule constructed by the ant, the amount of pheromone is increased by a fraction of the current amount of pheromone, and this fraction is directly proportional to Q .

Decreasing the Pheromone of Unused Terms

As mentioned above, the amount of pheromone associated with each $term_{ij}$ that does not occur in the constructed rule has to be decreased, to simulate the phenomenon of pheromone evaporation in real ant colony systems.

In our system the effect of pheromone evaporation is obtained by an indirect strategy. More precisely, the effect of pheromone evaporation for unused terms is achieved by normalizing the value of each pheromone t_{ij} . This normalization is performed by dividing the value of each t_{ij} by the summation of all t_{ij} , $\forall i, j$. To see how this achieves the same effect as pheromone evaporation, note that when a rule is constructed only the terms used by an ant in the constructed rule have their amount of pheromone increased by equation (6). Hence, 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 the *used* terms). The final effect will be to reduce the normalized amount of pheromone for each *unused* term.

Used terms will, of course, have their normalized amount of pheromone increased due to application of equation (6).

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;
- Minimum 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 over-fitting 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 chapter these parameters were set as follows:

- Number of Ants (*No_of_ants*) = 3000;
- Minimum number of cases per rule (*Min_cases_per_rule*) = 10;

- Maximum number of uncovered cases in the training set ($Max_uncovered_cases$) = 10;
- Number of rules used to test convergence of the ants ($No_Rules_Converg$) = 10.

We have made no serious attempt to optimize these parameter values. Such an optimization will 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.

There is one caveat in the interpretation of the value of No_of_ants . Recall that this parameter defines the *maximum* number of ants per iteration of the system. In our experiments the actual number of ants per iteration was on the order of 1500, rather than 3000. The reason why in practice much fewer ants are necessary to complete an iteration of the system is that an iteration is considered finished when $No_Rules_Converg$ successive ants converge to the same path (rule).

COMPUTATIONAL RESULTS

We have evaluated Ant-Miner across six public-domain data sets from the UCI (University of California at Irvine) data set repository (Aha & Murphy, 2000). The main characteristics of the data sets used in our experiment are summarized in Table 1. The first column of this table identifies the data set, whereas the other columns indicate, respectively, the number of cases, number of categorical attributes, number of continuous attributes, and 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 as a preprocessing step. This discretization was performed by the C4.5-Disc discretization algorithm, described in Kohavi and Sahami (1996). This algorithm simply uses the C4.5 algorithm for discretizing continuous attributes. In essence, for each attribute to be discretized we extract, 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 from the root to a leaf node in the constructed decision tree corresponds to the definition of a

Table 1: Data Sets Used in Our Experiments

Data set	#cases	#categ. attrib.	#contin. attrib.	#classes
breast cancer (Ljubljana)	282	9	0	2
breast cancer (Wisconsin)	683	0	9	2
Tic-tac-toe	958	9	0	2
Dermatology	358	33	1	6
Hepatitis	155	13	6	2
Heart disease (Cleveland)	303	8	5	5

categorical interval produced by C4.5. See the above-mentioned paper for details.

We have evaluated the performance of Ant-Miner by comparing it with C4.5 (Quinlan, 1993), a well-known rule induction algorithm. Both algorithms were trained on data discretized by the C4.5-Disc algorithm, to make the comparison between Ant-Miner and C4.5 fair.

The comparison was carried out across two criteria, namely the predictive accuracy of the discovered rule sets and their simplicity, as discussed in the following.

Predictive accuracy was measured by a 10-fold cross-validation procedure (Weiss & Kulikowski, 1991). In essence, the data set is divided into 10 mutually exclusive and exhaustive partitions. Then a classification algorithm is run 10 times. Each time a different partition is used as the test set and the other 9 partitions are used as the training set. The results of the 10 runs (accuracy rate on the test set) are then averaged and reported as the accuracy rate of the discovered rule set.

The results comparing the accuracy rate of Ant-Miner and C4.5 are reported in Table 2. The numbers after the “±” symbol are the standard deviations of the corresponding accuracy rates. As shown in this table, Ant-Miner discovered rules with a better accuracy rate than C4.5 in four data sets, namely Ljubljana breast cancer, Wisconsin breast cancer, Hepatitis and Heart disease. In two data sets, Ljubljana breast cancer and Heart disease, the difference was quite small. In the other two data sets, Wisconsin breast cancer and Hepatitis, the difference was more relevant. Note that although the difference of accuracy rate in Wisconsin breast cancer seems very small at first glance, this holds only for the absolute value of this difference. In reality the relative value of this difference can be considered relevant, since it represents a reduction of 20% in the error rate of C4.5. $((96.04 - 95.02)/(100 - 95.02) = 0.20)$

On the other hand, C4.5 discovered rules with a better accuracy rate than Ant-Miner in the other two data sets. In one data set, Dermatology, the difference was quite small, whereas in the Tic-tac-toe the difference was relatively large. (This result will be revisited later.) Overall one can conclude that Ant-Miner is competitive with C4.5 in terms of accuracy rate, but it should be noted that Ant-Miner’s accuracy rate has a larger standard deviation than C4.5’s one.

Table 2: Accuracy Rate of Ant-Miner vs. C4.5

Data Set	Ant-Miner’s accuracy rate (%)	C4.5’s accuracy rate (%)
Breast cancer (Ljubljana)	75.42 ± 10.99	73.34 ± 3.21
Breast cancer (Wisconsin)	96.04 ± 2.80	95.02 ± 0.31
Tic-tac-toe	73.04 ± 7.60	83.18 ± 1.71
Dermatology	86.55 ± 6.13	89.05 ± 0.62
Hepatitis	90.00 ± 9.35	85.96 ± 1.07
Heart disease (Cleveland)	59.67 ± 7.52	58.33 ± 0.72

We now turn to the results concerning the simplicity of the discovered rule set. This simplicity was measured, as usual in the literature, by the number of discovered rules and the total number of terms (conditions) in the antecedents of all discovered rules.

The results comparing the simplicity of the rule set discovered by Ant-Miner and by C4.5 are reported in Table 3. Again, the numbers after the “±” symbol denote standard deviations. As shown in this table, in five data sets the rule set discovered by Ant-Miner was simpler – i.e. it had a smaller number of rules and terms – than the rule set discovered by C4.5. In one data set, Ljubljana breast cancer, the number of rules discovered by C4.5 was somewhat smaller than the rules discovered by Ant-Miner, but the rules discovered by Ant-Miner was simpler (shorter) than the C4.5 rules. To simplify the analysis of the table, let us focus on the number of rules only, since the results for the number of terms are roughly analogous. In three data sets the difference between the number of rules discovered by Ant-Miner and C4.5 is quite large, as follows.

In the Tic-tac-toe and Dermatology data sets Ant-Miner discovered 8.5 and 7.0 rules, respectively, whereas C4.5 discovered 83 and 23.2 rules, respectively. In both data sets C4.5 achieved a better accuracy rate. So, in these two data sets Ant-Miner sacrificed accuracy rate to improve rule set simplicity. This seems a reasonable trade-off, since in many data mining applications the simplicity of a rule set tends to be even more important than its accuracy rate. Actually, there are several rule induction algorithms that were explicitly designed to improve rule set simplicity, even at the expense of reducing accuracy rate (Bohanec & Bratko, 1994; Brewlow & Aha, 1997; Catlett, 1991).

In the Heart disease data set Ant-Miner discovered 9.5 rules, whereas C4.5 discovered 49 rules. In this case the greater simplicity of the rule set discovered by Ant-Miner was achieved without unduly sacrificing accuracy rate – both algorithms have similar accuracy rates, as can be seen in the last row of Table 1.

There is, however, a caveat in the interpretation of the results of Table 3. The rules discovered by Ant-Miner are organized into an ordered rule list. This means

Table 3: Simplicity of Rule Sets Discovered by Ant-Miner vs. C4.5

Data set	No. of rules		No. of terms	
	Ant-Miner	C4.5	Ant-Miner	C4.5
breast cancer (Ljubljana)	7.20 ± 0.60	6.2 ± 4.20	9.80 ± 1.47	12.8 ± 9.83
breast cancer (Wisconsin)	6.20 ± 0.75	11.1 ± 1.45	12.2 ± 2.23	44.1 ± 7.48
Tic-tac-toe	8.50 ± 1.86	83.0 ± 14.1	10.0 ± 6.42	384.2 ± 73.4
Dermatology	7.00 ± 0.00	23.2 ± 1.99	81.0 ± 2.45	91.7 ± 10.64
Hepatitis	3.40 ± 0.49	4.40 ± 0.93	8.20 ± 2.04	8.50 ± 3.04
Heart disease (Cleveland)	9.50 ± 0.92	49.0 ± 9.4	16.2 ± 2.44	183.4 ± 38.94

that, in order for a rule to be applied to a test case, the previous rules in the list must not cover that case. As a result, the rules discovered by Ant-Miner are not as modular and independent as the rules discovered by C4.5. This has the effect of reducing a little the simplicity of the rules discovered by Ant-Miner, by comparison with the rules discovered by C4.5. In any case, this effect seems to be quite compensated by the fact that, overall, the size of the rule list discovered by Ant-Miner is much smaller than the size of the rule set discovered by C4.5. Therefore, it seems safe to say that, overall, the rules discovered by Ant-Miner are simpler than the rules discovered by C4.5, which is an important point in the context of data mining.

Taking into account both the accuracy rate and rule set simplicity criteria, the results of our experiments can be summarized as follows.

In three data sets, namely Wisconsin breast cancer, Hepatitis and Heart disease, Ant-Miner discovered a rule set that is both simpler and more accurate than the rule set discovered by C4.5. In one data set, Ljubljana breast cancer, Ant-Miner was more accurate than C4.5, but the rule sets discovered by Ant-Miner and C4.5 have about the same level of simplicity. (C4.5 discovered fewer rules, but Ant-Miner discovered rules with a smaller number of terms.)

Finally, in two data sets, namely Tic-tac-toe and Dermatology, C4.5 achieved a better accuracy rate than Ant-Miner, but the rule set discovered by Ant-Miner was simpler than the one discovered by C4.5.

It is also important to notice that in all six data sets the total number of terms of the rules discovered by Ant-Miner was smaller than C4.5's one, which is a strong evidence of the simplicity of the rules discovered by Ant-Miner.

These results were obtained for 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 C4.5 (on the order of seconds for each data set) to obtain the results.

It is worthwhile to mention that the use of a high-performance programming language like C++, as well as an optimized code, is very important to improve the computational efficiency of Ant-Miner and data mining algorithms in general. The current C++ implementation of Ant-Miner is about three orders of magnitude (i.e., thousands of times) faster than a previous MatLab implementation.

CONCLUSIONS AND FUTURE WORK

This work has proposed an algorithm for rule discovery called Ant-Miner (Ant Colony-based Data Miner). The goal of Ant-Miner is to extract classification rules from data. The algorithm is based on recent research on the behavior of real ant colonies as well as in some data mining concepts.

We have compared the performance of Ant-Miner with the performance of the well-known C4.5 algorithm in six public domain data sets. Overall the results show that, concerning predictive accuracy, Ant-Miner is competitive with C4.5. In additions, Ant-Miner has consistently found considerably simpler (smaller) rules than C4.5.

We consider these results very promising, bearing in mind that C4.5 is a well-know, sophisticated decision tree algorithm, which has been evolving from early decision tree algorithms for at least a couple of decades. By contrast, our Ant-Miner algorithm is in its first version, and the whole area of artificial Ant Colony Systems is still in its infancy, by comparison with the much more traditional area of decision-tree learning.

One research direction consists of performing several experiments to investigate the sensitivity of Ant-Miner to its user-defined parameters.

Other research direction consists of extending the system to cope with continuous attributes as well, rather than requiring that this kind of attribute be discretized in a preprocessing step.

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

REFERENCES

- Aha, D. W. & Murphy P. M. (2000). *UCI Repository of Machine Learning Databases*. Retrieved August 05, 2000 from the World Wide Web: <http://www.ics.uci.edu/~mllearn/MLRepository.html>
- Bohanec, M. & Bratko, I. (1994). Trading accuracy for simplicity in decision trees. *Machine Learning*, 15, 223-250.
- Bonabeau, E., Dorigo, M. & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press.
- Brewlow, L.A. & Aha, D.W. (1997). Simplifying decision trees: a survey. *The Knowledge Engineering Review*, 12, No. 1, 1-40.
- Catlett, J. (1991). Overpruning large decision trees. *Proc. 1991 Int. Joint Conf. on Artif. Intel. (IJCAI)*. Sidney.
- Cordón, O., Casillas J. & Herrera F. (2000). Learning Fuzzy Rules Using Ant Colony Optimization. *Proc. ANTS'2000 – From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms*, 13-21.
- Cover, T. M. & Thomas, J. A. (1991). *Elements of Information Theory*. New York: John Wiley & Sons.
- Dorigo, M., Colomi A. & Maniezzo V. (1996). The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26, No. 1, 1-13.
- Dorigo, M., Di Caro, G. & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, 5, No. 2, 137-172.
- Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996). From data mining to knowledge discovery: an overview. In: Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. (Eds.) *Advances in Knowledge Discovery & Data Mining*, 1-34. Cambridge: AAAI/MIT.
- Freitas, A. A. & Lavington, S. H. (1998). *Mining Very Large Databases with Parallel Processing*. London: Kluwer.
- Freitas, A.A. (2001). A survey of evolutionary algorithms for data mining and knowledge discovery. To appear in: Ghosh, A.; Tsutsui, S. (Eds.) *Advances in evolutionary computation*. Springer-Verlag.
- Kohavi, R. & Sahami, M. (1996). Error-based and entropy-based discretization of continu-

- ous features. *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, 114-119.
- Lopes, H. S., Coutinho, M. S. & Lima, W. C. (1998). An evolutionary approach to simulate cognitive feedback learning in Medical Domain. In: *Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives*, Singapore: Word Scientific, 193-207.
- Monmarche, N. (1999). On data clustering with artificial ants. In: Freitas, A.A. (Ed.) *Data Mining with Evolutionary Algorithms: Research Directions – Papers from the AAAI Workshop*. AAAI Press, 23-26.
- Quinlan, J.R. (1987). Generating production rules from decision trees. *Proc. 1987 Int. Joint Conf. on Artif. Intel.(IJCAI)*, 304-307.
- Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. San Francisco: Morgan Kaufmann.
- Stutzle, T. & Dorigo. M. (1999). ACO algorithms for the traveling salesman problem. In K. Miettinen, M. Makela, P. Neittaanmaki, J. Periaux. (Eds.) *Evolutionary Algorithms in Engineering and Computer Science*, New York: John Wiley & Sons.
- Weiss, S.M. & Kulikowski, C.A. (1991). *Computer Systems That Learn*. San Francisco: Morgan Kaufmann.