Data Mining Ant Colony for Classifiers

Ahmed Sameh*, Khalid Magdy**

*Department of Computer & Information Systems
Prince Sultan University,
P.O.Box 66833, Riyadh 11586
**Department of Computer Science and Engineering,
The American University in Cairo
P.O.Box 2511, Cairo, Egypt
Email: sameh.aucegypt.edu@gmail.com

Abstract – Self-organizing Ant Colony Optimization (ACO) is a technique that is inspired by the behavior of the ants as social insect that work together to accomplish a common goal using wisdom of the crowd. ACO is one of the algorithms that put swarm intelligence into action. Swarm intelligence, which is based on the idea of collective behavior, has occupied ACO in various fields and problem solving domains. Data mining is one of the domains where ACO has been applied successfully and provided scalable solutions. In this paper, we describe a knowledge discovery classification technique based on ACO. AntMiner, first proposed in [5], is a rule induction algorithm that occupies collective intelligence to construct classification rules. Experimental results are shown as the AntMiner+ is implemented with different variations inspired by discrete optimization, fuzzy rule induction, self-organizing map (SOM), dimensionality reduction, parallel simultaneous rule learning and tested on different datasets. Moreover, further combinations of these variations that produced enhancement are also proposed and tested.

Index Term-- Swarm Intelligence, Ant Colony Optimization (ACO), Data Mining, Classification

1. INTRODUCTION
Swarm intelligence is an area of computational intelligence that is inspired by the biological collective behavior found in social insects. An individual ant does not have much cognitive ability, yet an entire colony, overtime, can accomplish complex tasks like building a nest or finding the nearest food source to the colony. This is due to the collective intelligence of the whole swarm. Elements in the swarm are homogenous. Each element has the same simple behavioral model. For example, in a bird flock, each bird has a few simple rules; collision avoidance, velocity matching, flock centering. With these simple rules they manage to fly in a discipline order for long distances. Swarm intelligence, which is the aggregation of the member behaviors, has been adapted recently in many domains such as combinational optimization and knowledge discovery. The most popular techniques applied in this field is Particle Swarm Intelligence (PSO) and Ant Colony Optimization (ACO). The latter is the technique used in this paper for implementing a knowledge discovery of rule based classifiers.

1.1. Data Mining
Data mining refers to the process of discovering hidden useful patterns in the data. Variety of terms has been used to describe this same process such as knowledge discovery, information extraction, etc. Data mining explores large amount of data trying to find consistent patterns and/or relations between variables, and then validates the findings by applying the detected pattern to a new data. Data mining is useful in tasks as classification, clustering, association, and time series analysis.

Several domains have contributed to accomplish the task of data mining such as Statistics, Machine Learning and Databases. Swarm intelligence was recently used in data mining to help discovering robust rules and hidden relations in the data. AntMiner, the enhanced technique in this paper, exhibits the usage of swarm intelligence technique (ACO) in data mining to construct classification rules.

2. RULE INDUCTION BASED CLASSIFICATION
Classification is a supervised machine learning and is one of the most studied data mining techniques. The main goal is to predict the class \( C_i = f(x_1, \ldots, x_n) \), where \( x_1 \ldots x_n \) are input attributes. There is one distinguished attribute called dependant attribute. The input to the classification algorithm is a data set of training records with several attributes. Each record is labeled by a class \( C_i \). Rule Induction is one of the techniques used to build a classification model. The goal of rule induction is to create rules of the form:

\[
\text{IF} \ < \text{condition} > \ \text{THEN} \ < \text{Class} >
\]

These rules are generally created from a set of data examples, where each example contains a set of attributes, and a set of values for those attributes. Each attribute \( x_i \) has a domain consisting of possible values for that attribute \( V_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\} \).

The constructed rule can be represented in the following form:

\[
\text{IF} \ <V_{i1} \ \text{AND} \ V_{23} \ \text{AND} \ V_{31}, \ldots, V_{ij}> \ \text{THEN} \ C_n
\]

where \( i=\text{attribute number} \), \( j=\text{attribute value} \), \( n=\text{class number} \). In principle, the domain of each attribute does not have to be a discrete set. However, the attribute domains considered for the rule induction in this paper are always assumed to be a discrete set. In the case where an attribute has a real valued domain, the domain is discretised as a preprocessing step. Swarm intelligence was recently used in data mining to help discover robust rules and hidden relations in the data. AntMiner demonstrates the usage of swarm intelligence technique (ACO) in data mining to construct classification rules.
3. **ANT SYSTEM**

Ant Colony Optimization (ACO) technique is inspired by the behavior of real ant colonies. When searching for food, ants initially explore the surrounding area, leaving chemical evidence on the path that it took in order to be followed by other ants. As an ant finds food, it evaluates the quality and the quantity of it and carries some of them back to the nest dropping pheromone in amounts proportional to the quality.
and the quantity of the founded food. The pheromone trial will probabilistically guide the ants to the food source. Eventually, this kind of behavior will lead the convergence of the ant taking the shortest path to the best food source. The main steps of (ACO) algorithms are as follows:

1. Pheromone trail initialization
2. Solution construction using pheromone trail: each ant constructs a complete solution to the problem according to a probabilistic model
3. Solution evaluation: evaluate the quality of the solution based on a problem specific fitness function
4. Pheromone trail update: this is applied in two phases, which are reinforcement, where each ant deposits an amount of pheromone which is proportional to the fitness of its solution, and evaporation, where a fraction of the pheromone evaporates in order to avoid stagnation.

In essence, the design of an ant system implies the specification of the following aspects:
- An environment (usually a graph- see figure 1) that represents the problem domain in such a way that it is suitable for the ants to navigate and construct a solution for the problem.
- A problem dependent heuristic evaluation function (η), which represents a quality factor for the different steps that construct the solution.
- A rule for pheromone updating (τ), which takes into account the evaporation and the reinforcement of the trails.
- A probabilistic transition rule based on the value of the heuristic function (η) and on the strength of the pheromone trail (τ) that is used to iteratively construct a solution.
- A fitness function by which the constructed solution is evaluated.
- A clear specification of when the algorithm converges to a solution.

4. ANTMiner Classification Technique

The following pseudo-code describes the main work of the AntMiner technique. First, a directed acyclic construction graph is built where the ants navigate from node to node to construct a rule (see figure 1). Ant transition from node to another is probabilistic based on the pheromone amount on the edge between the two nodes and the value calculated by a heuristic function applied on the destination node. After a rule is constructed, it is evaluated against the training set, and the pheromone on the rule edges is updated based on the quality of the rule. In each iteration, the best rules generated are selected and added to the result rule set, the covered cases in the training set by the rules are removed and the pheromone in the construction graph is reset. The algorithm runs until percentage of the cases are covered or for a specific number of iterations.

AntMiner+ pseudo-code:

- Construct Graph
- WHILE (Not min. percentage of cases covered OR max. number of iteration)
  - Initialize heuristics, pheromones and probabilities of edges
  - FOR EACH Ant in the swarm
    - Let ant run from source to sink and construct a rule
    - Evaluate the Quality of the rule
    - Update the pheromone based on the quality of the generated rule
  - END FOR EACH
  - Extract the best rule generated
  - Remove the covered cases by the best rule for the training set
- END WHILE
- Evaluate performance on test set

4.1. Construction Graph

A directed acyclic graph represents the solution space for AntMiner algorithm (see figure 1). Each node \( n_{i,j} \) represents a value \( V_j \) for the attribute \( A_i \) for in the attribute vector that defines the case. Direct edges are constructed from node \( n_{i,j} \) to node \( n_{i+1,j} \). So as an ant moves from node to node, it constructs a rule in this shape:

\(<A_i = V_j \; \text{AND} \; A_{i+1} = V_k \; \text{THEN} \; \text{Class} \; C_l>\)

Extra dummy nodes are added for each \( A_i \) to allow some rules to be constructed without involving each attribute. Rule pruning must be done after a rule is constructed to remove any selected dummy values. Continuous value attributes should be discretised before the graph construction.

4.2. Edge Probability

The probability that an ant in node \( n_{i,j} \) will go to node \( n_{i+1,k} \) is calculated as follows:

\[
P(n_{i,j}, n_{i+1,k}) = \frac{[\tau(n_{i,j}, n_{i+1,k})]^\alpha \cdot [\eta n_{i+1,k}]^\beta}{\sum_{l=1}^{n_{i+1}} [\tau(n_{i,j}, n_{i+1,l})]^\alpha \cdot [\eta n_{i+1,l}]^\beta}
\]

(4.2.1)

The probability depends on the heuristic value η and pheromone value τ, where \( \alpha \) and \( \beta \) are parameters representing the weight of each.

4.3. Heuristic Value

The heuristic value is problem specific input that helps in finding a good next step in constructing a solution. As for rule induction, a heuristic value for a specific attribute value can be measured in the number of cases covered by this value in a specific class. This can be calculated as follows:
\[ \eta_{n_i,k} = \frac{|V_i = Value_k \& \text{class} = 1|}{|V_i = Value_k|} \]  
(4.3.1)

4.4. Pheromone Update

Pheromone is updated after each ant constructs a rule. This is done in two steps.

Evaporation:
Pheromone level is diminished according to the following rule:

\[ \tau(n_i,j,n_{i+1,k})(t + 1) = \rho \cdot \tau(n_i,j,n_{i+1,k})(t) \]  
(4.4.1)

where \( \rho \) is the evaporation factor. Typical values for \( \rho \) lie in the range \([0.8, 0.99]\).

Reinforcement:
Pheromone level is increased on the edges of the constructed rule in a proportional way to the quality of the rule, as follows:

\[ \tau(n_i,j,n_{i+1,k})(t + 1) = \rho \cdot \tau(n_i,j,n_{i+1,k})(t) + \Delta \]  
(4.4.2)

where \( \Delta \) represents the reinforcement value that is proportional to the quality of the constructed rules. This quality of a rule is calculated by the sum of its confidence and coverage, as follows:

\[ \frac{|\text{rule}_{ib-ant} \& \text{Class} = 1|}{|\text{rule}_{ib-ant}|} + \frac{|\text{rule}_{ib-ant} \& \text{Class} = 1|}{|\text{Cov} = 0|} \]  
(4.4.3)

Where \(|\text{rule}_{ib-ant}|\) represents the number of the cases described by the this rule, and \(|\text{Cov}=0|\) represents the number of the uncovered rules in the training set.

4.5. Algorithm variations

In this paper seven variations of the original AntMiner are proposed, implemented and tested. The variations are: local search support, Multi-pheromone edges, Moving from exploration to exploitation, quality contrast intensification, early stopping criteria, web-based distribution, and parallel AntMiner variation. In the following sections we will elaborate on each variation.

5. EXPERIMENTS

5.1. Experimental Setup

The experiments code is implemented using Microsoft.NET Frame Works 3.5, Microsoft C# language and Visual Studio 2008. Several Datasets from UCI repository [1] were used for the experiments. A detailed description of each dataset is exhibited in the following section, where the algorithm parameters are described in section 5.3.

5.2. Datasets

   - Number of Instances: 1728 (instances completely cover the attribute space)
   - Number of Attributes: 6
   - Attribute Values:
     - buying       v-high, high, med, low
     - maint        v-high, high, med, low
     - doors        2, 3, 4, 5-more
     - seaters      2, 4, more
     - lug_boot     small, med, big
     - safety       low, med, high
   - Missing Attribute Values: none
   - Class Distribution (number of instances per class)
     - unacc     1210      (70.023 %)
     - acc         384        (22.222 %)
     - good      69          ( 3.993 %)
     - v-good   65          ( 3.762 %)

2. Voting Records: Categorical Data
   - Number of instances: 435
   - Number of attributes: 16 (all Boolean valued)
   - Class Distribution: (2 classes)
     - Democrat 267 (54.8%)
     - Republican 168 (45.2%)
   - Missing Values: yes.

5.3. Algorithm Parameters

- Number of ants=1000
- Max number of iterations \( I \)
- Min percentage of cases to be covered to stop.
- Evaporation factor \( \rho = 0.85 \)
- Relative weights \( \alpha \) and \( \beta \) set at respectively 2 and 1

5.4. Proposed Modifications

5.4.1. Local Search

Local search is a cost effective technique that is used in optimization problems, after the best rule of the iteration is extracted, a local search is performed on it. In seeking for more general accurate rule, each attribute in the rule premises is removed and tested to evaluate its quality. If the new rule has a better quality than the original one, it’s taken instead of the original one and more attributes are tried to be removed until the modified rule gives less quality than the original one. The following is the algorithm:

- BRule=IterationBestGeneratedRule
- For each Attribute Att in BRule
  - NRule=BRule.Remove(Att)
  - IF NRule.Quality>BRule.Quality
    - BRule=NRule
  - End IF
- End For each
5.4.2. Multi-Pheromone System
In the original AntMiner algorithm, any ant is affected by the amount of the pheromone on the edge in selecting the next node to visit. Ants drop pheromone based on the quality of the rule that it's constructed. But the quality of the rule constructed to be classified as “ClassA” should not affect the construction of a rule that would be classified as “ClassB”. In other words, an ant constructed a rule that to be classified as “ClassA” should only be affected by the pheromone dropped by previous ants constructed “ClassA” rules. This technique implies that the rule class should be chosen first in the rule construction operation. When ant chooses “ClassA” for a rule, it is only affected by the “ClassA” pheromone dropped on the construction graph edges. And when the ant finishes constructing and evaluating the rule, it drops “ClassA” pheromone on the selected edges. This could be implemented by making the graph edges having as many lanes as the number of the class values (see figure 2). As the ant chooses the rule class first, it wanders in the graph by considering the lane of the per-selected class on the edges. The edge selection and the pheromone update operation are kept the same by making the graph edges having as many lanes as the number of the class values.

5.4.3. Moving from Exploration to Exploitation
A good ant system tries to balance between exploration (wandering in the problem space looking for a good solution) and exploitation (following the previous experience of the ants). For any ant system, the wisdom of the swarm (represented in the amount of the pheromone) is low in the first iterations. As more and more ants go, explore and try different solution, the amount of the wisdom increase in the swarm, and this is when the exploitation should have more weight. On the other hand, it should be ignored in the early stages of the swarm activation. The idea is that at the beginning the selection of the edges is completely random, which enhance the exploration aspect. The probability of selecting the edge randomly deceases with the increase of the number of the ants that went through the problem space, as there will be enough wisdom (pheromone) to exploit.

5.4.4. Quality Contrast Intensifier
The pheromone is dropped on based on the quality of the trial. The idea is to intensify the contrast between good solution and better solution. Moreover, we want to remove pheromone form the trials that are below a certain threshold of quality. So good solution are rewarded with an amount of pheromone proportional to the quality, and the bad solutions are penalized by removing pheromone from the path.

```plaintext
Function GetSolutionRuleQuality
  Returns Quality
  Begin
  IF Quality > UpperThershold
    o Return Quality * 100
  IF Quality < LowerThershold
    o Return Quality - 10
  Else
    o Return Quality
  End
```

5.5. Experimental Results

| AntMiner Original – Car Evaluation Data Set Results |
|-------|-------|-------|-------|-------|-------|-------|
| I     | Ants=100 | Ants=1000 |
|       | T  | R  | Cov | Acc | T  | R  | Cov | Acc |
| 20    | 0:16 | 20  | 16% | 14% | 0:28 | 19  | 52% | 50% |
| 30    | 0:23 | 30  | 27% | 25% | 0:38 | 27  | 64% | 61% |
| 40    | 0:35 | 40  | 39% | 37% | 0:48 | 36  | 71% | 70% |

| AntMiner Original with Local Search – Car Evaluation Data Set Results |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| I               | Ants=100       | Ants=1000      |
|                 | T  | R  | Cov | Acc | T  | R  | Cov | Acc |
| 20              | 0:12 | 9   | 58% | 54% | 0:11 | 15  | 71% | 68% |
| 30              | 0:20 | 12  | 61% | 59% | 0:26 | 18  | 78% | 76% |
| 40              | 0:25 | 16  | 67% | 64% | 0:30 | 22  | 80% | 78% |

| Multi-pheromone AntMiner – Car Evaluation Data Set Results |
|----------------|----------------|----------------|----------------|----------------|----------------|
| I               | Ants=100       | Ants=1000      |
|                 | T  | R  | Cov | Acc | T  | R  | Cov | Acc |
| 20              | 0:14 | 18  | 37% | 34% | 0:30 | 15  | 65% | 62% |
| 30              | 0:20 | 25  | 44% | 41% | 0:27 | 15  | 68% | 67% |
| 40              | 0:27 | 37  | 53% | 50% | 0:30 | 34  | 79% | 76% |

| Multi-pheromone AntMiner with Local Search – Car Evaluation Data Set Results |
|----------------|----------------|----------------|----------------|----------------|----------------|
| I               | Ants=100       | Ants=1000      |
|                 | T  | R  | Cov | Acc | T  | R  | Cov | Acc |
| 20              | 0:08 | 8   | 63% | 61% | 0:13 | 17  | 88% | 86% |
| 30              | 0:12 | 12  | 68% | 65% | 0:24 | 26  | 95% | 93% |
| 40              | 0:16 | 14  | 73% | 71% |  -   |  -   |  -   |  -   |
Chan & Freitas [9] have proposed a new rule pruning procedure for AntMiner. They have observed that the original Ant-Miner’s pruning procedure processing time increases significantly with a large increase in the number of attributes, which affects the scalability of the method. To overcome this limitation, they proposed a new prune procedure that led to the discovery of simpler (shorter) rules and improved the computational time in datasets with a large number of attributes.

Martens et al. [10] have introduced a new classification algorithm, named AntMiner+, based on AntMiner. It differs from the original AntMiner implementation in several aspects. Firstly, it makes a distinction between nominal and ordinal attributes. Nominal attributes have unordered nominal values (e.g. gender has unordered values “male” and “female”). Ordinal attributes are those categorical or discrete attributes whose values are ordered (e.g. “0”, “1”, “2”, “3” and “4 or more”, which may be the domain of an attribute that represents the number of children in a family). Instead of creating a pair (attribute = value) for each value of an ordinal attribute, AntMiner+ creates two types of bounds that represent the interval of values to be chosen by the ants. The first type represents the lower bound of the interval and takes (attribute ≥ valuei) form, and the second type represents the upper bound of the interval and takes (attribute ≤ valuej) form (valuei and valuej are values from the attribute domain). Moreover, it employs different pheromone initialization and update procedures based on the MAX – MIN ant system (where the pheromone level of the edges is restricted by an upper bound (τmax) and a lower-bound (τmin)). These bounds are dynamically updated.

Swaminathan [10] proposed an extension to Ant-Miner which enables interval conditions in the rules. While it still uses a discretization method to define discrete intervals for continuous attributes in a preprocessing step, the continuous values are not replaced in the dataset. For each discrete interval, a node (e.g. humidity ≤ 75) is added to the construction graph and the pheromone value associated to the node is calculated using a mixed kernel probability density function.

The idea of dividing the dataset into training, validation, and testing subgroups for the purpose of “early stopping” could be easily incorporated into the above experiments.

As a future extension to the current work is the implementation of a distributed version of the AntMiner using the newly suggested Web Application server in [11]. The idea is to run the AntMiner as a distributed application to access data stored in a distributed database instead of a centralized database. Within such setup, we can simultaneously run the several variations of the ACO algorithms simultaneously (with each maintaining its own problem graph, pheromone levels and heuristic values) and let them exchange the best ants frequently (similar to the idea of parallel genetic algorithms). This way, the various ACO algorithms can reinforce each other and lead to better overall performance. This arrangement can also prevent over-fitting of the training data.

### Table 1: AntMiner Original – Voting Records Data Set Results

<table>
<thead>
<tr>
<th>I</th>
<th>Ants=100</th>
<th>Ants=1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>20</td>
<td>0.12</td>
<td>13</td>
</tr>
<tr>
<td>30</td>
<td>0.20</td>
<td>18</td>
</tr>
<tr>
<td>40</td>
<td>0.29</td>
<td>24</td>
</tr>
</tbody>
</table>

### Table 2: AntMiner Original with Local Search – Voting Records Data Set Results

<table>
<thead>
<tr>
<th>I</th>
<th>Ants=100</th>
<th>Ants=1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>20</td>
<td>0.08</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>0.11</td>
<td>8</td>
</tr>
<tr>
<td>40</td>
<td>0.20</td>
<td>9</td>
</tr>
</tbody>
</table>

### Table 3: Multi-pheromone AntMiner – Voting Records Data Set Results

<table>
<thead>
<tr>
<th>I</th>
<th>Ants=100</th>
<th>Ants=1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>20</td>
<td>0.14</td>
<td>11</td>
</tr>
<tr>
<td>30</td>
<td>0.21</td>
<td>14</td>
</tr>
<tr>
<td>40</td>
<td>0.28</td>
<td>21</td>
</tr>
</tbody>
</table>

### Table 4: Multi-pheromone AntMiner with Local Search – Voting Records Data Set Results

<table>
<thead>
<tr>
<th>I</th>
<th>Ants=100</th>
<th>Ants=1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>20</td>
<td>0.07</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>0.11</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>0.16</td>
<td>4</td>
</tr>
</tbody>
</table>

I represent the number of iterations.

### 6. Analysis of the Results

Form the experimental results, we see that the number of ants affects the quality of the generated rules, because ants after ants trying to find a good rule to generate, the wisdom of the swarm increases by the amount of the pheromone left by previous ants that leads to the good solution. Local search has enhanced the quality of the generated rules in a significant way as less number of rules is generated with more coverage and accuracy. This can even help in avoiding over fitting by generating more general rules. Finally, multi-pheromone AntMiner has shown better results than the original one that we only need 30 iterations with local search to converge and generate 25 rules and cover more than 95% of the training data set.

### 7. Further Variations

Several variations were proposed involving different pruning and pheromone update procedures, new rule quality measures and heuristic functions, discovering fuzzy classification rules and discovering rules for multi-label classification problems.
REFERENCES

APPENDIX
The amount of the pheromone on the swarm represents the current swarm wisdom in choosing a good path to solve the problem. This wisdom is gained by the previous ants that tried different paths to solve the problem and left their experience in the form of pheromones. In the AntMiner problem, not all ants should follow the experience of a previous ant who tried to construct any rule, they only should follow the experience of the ants tried to construct a rule with the same class. For example, an ant trying to construct a rule which class is “A”, should only be influenced by the pheromones dropped by ants that constructed rules with class “A”. That led to the idea of Multi-pheromone Ant System, where ant drops a different kind of pheromone according to the rule class. And the next coming ants are influenced only by the kind of the pheromone corresponding to their current rule class.

The following is the source code of probabilistically selecting an edge:

```csharp
float[] edgesProbability=
new float[currentNode.Edges.Count];
float totalEdgesValue = 0;
int i = 0;
foreach (Edge edge in currentNode.Edges)
{
    //Compute the edge probability based in the amount of the pheromone on the lane of the current class
    edgesProbability[i] = edgeValue;
    totalEdgesValue += edgeValue;
    i++;
}
//Rolette Wheele for probabilistic selection
int randomNumber = randomGenerator.Next(0, 1000);
for (i = 0; i < edgesProbability.Length; i++)
{
    edgesProbability[i] = (int)(edgesProbability[i] / totalEdgesValue) * 1000;
    if (i != 0)
    edgesProbability[i] += edgesProbability[i - 1];
    if (randomNumber <= edgesProbability[i])
    {
        selectedEdgeIndex = i;
        break;
    }
}
```