The Application of the Ant Colony Decision Rule Algorithm on Distributed Data Mining

LinQuan Xie
HongBiao Mei
Application Institute of Jiangxi
University of Science and Technology Beijing

ABSTRACT

Based on the deep research of Ant Colony Algorithm, an Ant Colony decision rule Algorithm is proposed for rules mining based on the thought of ant colony and rule mining of decision tree. Then the algorithm is compared with C4.5 and applied on the rules mining and the results are showed by simulation.

INTRODUCTION

In general, 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 (Quinlan, 1986). There are several data mining tasks, including classification, regression, clustering, dependence modeling, etc. (Quinlan, 1993). 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 recent, there are many mining tools, such as neutral network, gene algorithm, decision trees, rule referring, to predict the future development[1], and they are able to help people to make good decisions. But there are some shortcomings in these methods, such as incomprehensive results, over-fit rules and difficulties in being applied on distributed simulation. In this paper we propose an Ant Colony Optimization (ACO) algorithm (Dorigo, Maniezzo & Colorni, 1996; Stutzle & Hoos, 1997; and de A. Silla & Ramalho, 2001) 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.

In this paper, the ACDR is applied for resolving the distributed database mining. The simulation results show that the ACDR is an available and correct algorithm for distributed mining.

Foundation Item: Shanghai Technology Development Key Item(06DZ12001);
Author Resume: Xie Linquan(19-), man, Fujian, professor, mainly research area is Rough Set Theory.
Mei Hongbiao(1976-),man, Jiangxi, doctor postgraduate, mainly research area is Distributed Simulation.

THE ANT COLONY OPTIMIZATION ALGORITHM

The ant colony optimization technique has emerged recently as a novel meta-heuristic belongs to the class of problem-solving strategies derived from natural (other categories include neural networks, simulated annealing, and evolutionary algorithms). The ant system optimization algorithm is basically a multi-agent system where low level interactions between single agents (i.e., artificial ants) result in a complex behavior of the whole ant colony. Ant system optimization algorithms have been inspired by colonies of real ants, which deposit a chemical substance (called pheromone) on the ground. It was found that the medium used to communicate information among individuals regarding paths, and used to decide where to go, consists of pheromone trails. A moving ant lays some pheromone (in varying quantities) on the ground, thus making the path by a trail of this substance. While an isolated ant moves essentially at random, an ant encountering a previously laid trail can detect it and decide with high probability to follow it, thus reinforcing the trail with its own pheromone.

The collective behavior where that emerges is a form of autocatalytic behavior where the more the ants following a trail, the more attractive that trail becomes for being followed. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path.

Given a set of n cities and a set of distances between them, the Traveling Salesman Problem (TSP) is the problem of finding a minimum length closed path (a tour), which visits every city exactly once. We call d_{ij} the length of the path between dot *i* and *j*. An instance of the TSP is given by a graph (*N*, *E*), where *N* is the set of cities and *E* is the set of edges between cities (a fully connected graph in the Euclidean TSP). Let $b_i(t)$ ($i = 1,\dots,n$) be the number of ants in city *i* at time *t* and let $m = \sum_{i=1}^{n} b_i(t)$ be the total number of ants. Let $\tau_{ij}(t+n)$ be the intensity of pheromone trail on connection (*i*, *j*) at time *t* + *n*, given by

$$\tau_{ij}(t+n) = (1-\rho)\tau_{ij} + \Delta\tau_{ij}(t,t+n) \qquad (1)$$

Where$\rho$is a coefficient such that 1-ρdenotes a coefficient which represents the evaporation of trail between time *t* and t+n, $\Delta\tau_{ij}(t,t+n) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t,t+n)$, where $\Delta\tau_{ij}^{k}(t,t+n)$ is the quantity per unit of length of trail substance (pheromone in real ants) laid on connection (*i*, *j*) by the *k^th^* ant at time *t* + *n* and is given by the following formula:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k}, if(i,j) \in tour \quad described \quad by \quad tabu_k \\ 0, \qquad\qquad otherwise \end{cases} \qquad (2)$$

Where: *Q* denotes a constant and L_k represents the tour length found by the k^{th} ant. For each edge, the intensity of trail at time 0 ($\tau_{ij}(0)$) is set to a very small value.

While building a tour, the transition probability that ant *k* in city *i* visits city *j* is:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{0}\ {}_{k \in allowed_k} \tau_{ik}^{\alpha}(t)\eta_{ik}^{\beta}(t)}, if \quad j \in allowed_k(t) \\ 0, \qquad\qquad else \end{cases} \qquad (3)$$

Where: allowed*k*(*t*) is the set of cities not visited by ant *k* at time *t*, and η_{ij} denotes a local heuristic which equal to 1/*d* (and it is called 'visibility'). The parameter α and β control the relative importance of pheromone trail versus visibility. Hence, the transition probability is a trade-off between visibility, which says that closer cities should be chosen with a higher probability, and trail intensity, which says that if the connection (*i*, *j*) enjoys a lot of traffic then is it highly profitable to follow it.

A data structure, called a tabu list, is associated to each ant in order to avoid that ants visit a city more than once. This list tabu*k*(*t*) maintains a set of visited cities up to time *t* by the k^{th} ant. Therefore, the set allowed*k*(*t*) can be defined as follows: $allowed_k(t) = \{j \mid j \notin tabu_k(t)\}$, When a tour is completed, the tabu*k*(*t*) list (*k* = 1,…, *m*) is emptied and every ant is free again to choose an alternative tour for the next cycle.

By using the above definitions, we can describe the ant colony optimization algorithm as follows:

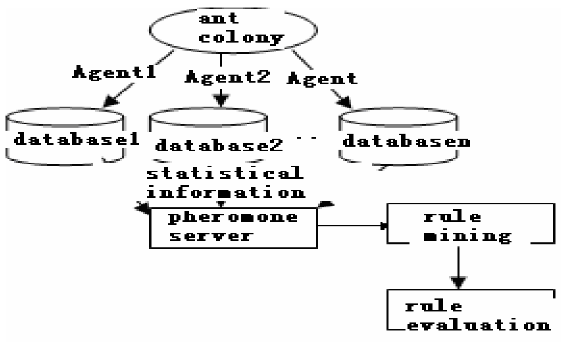
Figure 1: The ACO algorithm.

```
procedure ACO algorithm for TSP
    Set parameters, initialize pheromone trails
    while (termination condition not met) do
        Construct Solutions
        Apply Local Search
        Local_Pheromone_Update
    End
    Global_Pheromone_Update
end ACO algorithm for TSPs
```

THE ANT COLONY DECISION RULE ALGORITHM (ACDR) AND ITS' MINER

Classification is a research branch of ant colony theory. The ant miner was firstly proposed for the classification by the reference (Parpinelli, Lopes, & Freitas (2002). In this paper, we use the ACDR to resolve distributed database mining. The structure is described as figure 2.

Figure 2. The structure of distributed mining.



The flows of ACDR as follows:

```
Initial the parameters;
For i=1:m // the cycles
Train list TS=[ $TS_1$,$TS_2$,…,$TS_n$], $TS_i$={all training cases}; initial Rule list RL;
While ($TS_i$ >Max_Uncovered_Case)
Initial the amount of pheromone;
```

```
For n agents in parallel
Stat. every database relative information;
End
Calculate the heuristic function value;
t=1// the ant number of rule
j=1// No.of ant
Repeat
Calculate the other value;
$Ant_t$ 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 ($R_j$=$R_{j-1}$)
   T=T+1;
Else
   T=1;
End
j=j+1;
UNTIL (i □□No_of_ants) OR (j □□No_rules_converg)
Choose the best rule Rbest among all rules Rt constructed by all the ants;
Add rule Rbest to DiscoveredRuleList;
$TS_i$ = $TS_i$ - {set of cases correctly covered by $R_{best}$};
END WHILE
Calculate the average density of rules;
Choose the minimal density rule as the $RL_{best}$;
End
```

The details of the algorithm as follows:

COMPRISING RULE CONSTRUCTION

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. 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)……”.

Rule pruning

Rule pruning is a commonplace technique in data mining, 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 usually easier to be understood by the user than a longer one. In this paper, the basic idea of rule pruning is to iteratively remove one-term-at-a-time from the rule while this process improves the quality of the rule. And this process is keeping until the rule has just one term or until no term removal will improve the quality of the rule. The quality of a rule, denoted by *Q*, is computed by the formula:

$$Q = \left(\frac{TruePos}{TruePos + FalseNeg}\right) \bullet \left(\frac{TrueNeg}{FalsePos + TrueNeg}\right) \qquad (4)$$

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 $0 < Q < 1$ and, the larger the value of Q, the higher the quality of the rule.

In the rule pruning, 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 statistical information of all databases

(1) k: the number of class in all cases;

(2) $term_{ij}$,that is $Ai=V_{ij}$;

(3) $|T_{ij}|$: the cases belong to $term_{ij}$;

(4) $freqT_{ij}^{w}$: the cases belong to $term_{ij}$, and the class is w.

(5) a: the amount of attributes in database;

(6) b_i: the amount of possible values of attribute i;

(7) $\tau_{ij}(t)$:the amount of pheromone of $term_{ij}$ when t , and the initial value is :

$$\tau_{ij}(t=0) = \frac{1}{\sum_{i=1}^{a} b_i} \qquad (5)$$

Sum all the statistical information from all databases

The pheromone server and other relative servers which calculate all databases calculate the $freqT_{ij}^{w}$, and compute the follows:

(1) Heuristic Function η_{ij}:

$$\eta_{ij} = \frac{\max(\sum_n freqT_{ij}^{1}, \sum_n freqT_{ij}^{2}, \cdots \sum_n freqT_{ij}^{k})}{\sum_n |T_{ij}|} \quad (6)$$

(2) $R_{ij}(t)$: the relative value when t, and compute as:

$$R_{ij}(t) = \frac{\tau_{ij}(t) \bullet \eta_{ij}}{\sum_{i=1}^{a} \bullet \sum_{j=1}^{b_i} \tau_{ij}(t) \bullet \eta_{ij}}, \forall i \in I \quad (7)$$

Where: I is the attributes that are not used.

(3) θ:the average density of term, and compute as:

$$\theta = \frac{\sum_{i=1}^{a} \bullet \sum_{j=1}^{b_i} \eta_{ij}}{\sum_{j=1}^{a} b_i}, \forall i \in I \quad (8)$$

In the formula, θdenotes the threshold value ant agents select terms.

Construct the classification rule

The key for a rule constructing is to choose one item for some class. The possibility that one item is chosen to the current rule is donated by:

$$P_{ij}(t) = \frac{R_{ij}(t)^2}{R_{ij}(t)^2 + \theta^2} \quad (9)$$

Updating pheromone

When an agent has constructed one rule, the pheromone of the terms of the rule is updated by the formula (10), and the pheromone of the other terms not belonging to the rule is updated by standardization (Haibin, 2005).

$$\tau_{ij}(t) = (1 - \rho(\tau_{ij}(t)))\tau_{ij}(t) + (1 - \frac{1}{1+Q})\tau_{ij}(t) \quad (10)$$

Where :

$\tau_{ij}(t)$is the sum of the pheromone of $term_{ij}$ when time is t;

$$\rho(\tau_{ij}(t)) = \begin{cases} 1-e^{-\tau_{ij}(t)} \cdots\cdots\cdots\cdots \tau_{ij}(t) \le 0.2 \\ 0.05 \\ 1-e^{-\tau_{ij}(t)} \cdots\cdots\cdots\cdots \tau_{ij}(t) \ge 0.75 \end{cases} \quad (11)$$

The rule density RD

The RD is the criterion of the best rule list, and denoted as :

$$RD(k) = {Terms(k)} \Big/ {Rules(k)} \quad (12)$$

Where: Terms(k) is the sum of terms in DRlist during the kth loop; Rules(k) is the sum of terms in DRlist during the kth loop.

Analysis of ACDR's Computational Complexity

(1) The values of all η_{ij} are pre-computed, as a preprocessing step, and kept fixed throughout the algorithm. So the time complexity of this step is O(n·a), where n is the number of cases and a is the number of attributes.

(2) Computational complexity of a single iteration of the WHILE loop of Algorithm ACDR is composed of two sides: ① 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. ② we have to consider the REPEAT loop. 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<a (since each attribute can occur at most once in a rule). Hence, rule construction takes O(k • a). ③Rule Evaluation – This step consists of measuring the quality of a rule, as given by Equation (4). This requires matching a rule with k conditions with a training set with N cases, which takes O(k • n). ④The entire rule pruning process is repeated at most k times, so rule pruning takes at most: n • (k-1) • k+ n • (k-2) • (k-1) + n • (k-3) • (k-2) + ... + n • (1) • (2), this is, O(k^3 • n).⑤Since k < a, pheromone update takes O(a). Hence, a single iteration of the WHILE loop takes: O(T • [k • a + n • k^3]). In order to derive the computational complexity for the entire WHILE loop we have to multiply O(T • [k • a +n • k^3]) by r, the number of discovered rules, which is highly variable for different data sets.

(3) Therefore, the computational complexity of one loop as a whole is: O(r • T • [k • a + k^3 • n] + a • n). where: r denotes the discovered rules, T is the number of ant of each agent.

(4) if m is the number of loop, the computational complexity of the whole ACDR algorithm is O(m·(r • T • [k • a + k^3 • n] + a • n)). The computational complexity of the whole ACDR depends on r and k. So, in this paper, we set the rate=sum(k)/sum(r) as the evaluation function.

SIMULATION OF THE ACDR

Data Set

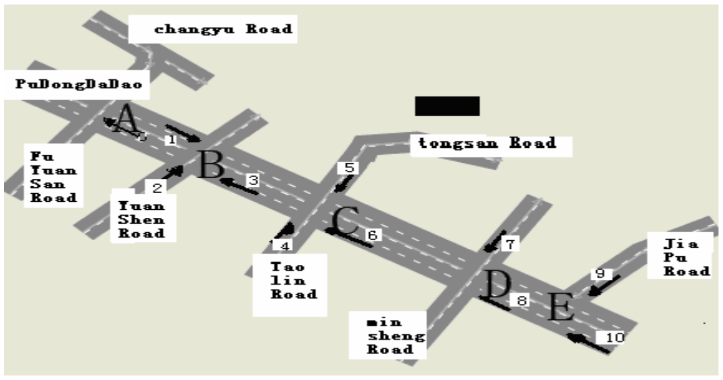
The performance of ACDR was evaluated using data sets from the SHTIC (Shanghai Traffic Information Centre) repository. Set the Pu Dong Da Dao Road as the application. And we choose five crosses noted as A,B,C,D - showed in figure3. We set the traffic flow of A as the predicted object. Note CE0 as the entrance 0 (Dougherty, 1996;

Smith & Demetsky, 1997), and its' upriver entrances are CE1, CE2, CE3, CE4, CE5, CE6, CE7, CE8, CE9, CE10. the time-lapse number of each cross shows as label1. So the attributes of training set are 34. The data is divided into 10 sets after it is dealt with- this process data preprocessing will be not narrated. The 10 sets are input in 10 databases. Every set has 20,000 records. We choose one set as testing set, and the others are training set.

Table 1: The time-lapse of entrances.

Entrance	0	1	2	3	4	5	6	7	8	9	10
Time-lapse	2	2	3	4	3	3	5	3	2	2	5

Figure 3: PuDongDaDao Crosses.



COMPUTATION RESULTS

We set the parameters as follows:

Max_uncovered_case=10, No_of_Agents=10, No_of_Ants=1800, α =1, β =2, min_case_Per_rule=10, No_Rules_Converg=10. The computational results show in tabel2. It shows that the rules mined by ACDR are simple and small.

Table 2: The results of ACDR for traffic flow.

Algorithm	Accuracy(%)	Simulation time(s)	Average density	rules
ACDR	94.72± 0.71	136	2.53	9.5± 0.30

COMPARING WITH C4.5

We have evaluated the performance of Ant-Miner by comparing it with C4.5 (Quinlan, 1986 & 1993), a well-known classification-rule discovery algorithm. The heuristic function used by ACDR, the entropy measure, is the same kind of heuristic function used by C4.5. The main difference between C4.5 and ACDR, with respect to the heuristic function, is that in C4.5 the entropy is computed for an attribute as a whole, since an entire attribute is chosen to expand the tree, whereas in ACDR 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 C4.5 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). In this case, we can know the priority of ACDR and C4.5 in table 3.

Table 3: Comparing with C4.5.

Algorithm	Accuracy(%)	Simulation time(s)	Average density	rules
ACDR	94.72± 0.71	136	1.71	9.5± 0.30
C4.5	90.38 ± 1.66	128	2.79	42.40 ± 0.71

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

(1) Concerning classification accuracy, ACDR obtained results somewhat better than C4.5.

(2) Concerning the simplicity of discovered rules, ACDR discovered rule lists much simpler (smaller) than the rule lists discovered by C4.5.

(3) Concerning the time of simulation, ACDR's time is more than C4.5's. 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 time.

CONCLUSIONS AND FUTURE WORK

This work has proposed an algorithm for rule discovery called ACDR. The goal of ACDR is to discover classification rules in data sets of distributed database. 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 ACDR and the well-known C4.5 algorithm in SHTIC data sets. The results showed that, ACDR is better than C4.5 in accuracy and simplicity, but worse in time.

Future research is in two important directions as follows. First, it would be interesting to extend ADR to cope with continuous attributes, rather than requiring that this kind of attribute be discrete in a preprocessing step. Second, it would be interesting to study the parameters for improving the quality of the ACDR.

REFERENCES

Chiou, J.P., Chang, C. F., & Su, C. T., (2004). Ant Direction Hybrid Differential Evolution for Solving Large Capacitor Placement Problems, *IEEE Transactions on Power Systems,* 19 (4).

de A.Silla, R. M., & Ramalho, G. L., (2001). Ant System for the Set Covering Problem, *IEEE Transactions on Evolutionary Computation,* 1, 3129 -3134.

Dorigo, M., Di Caro, G., & Gambardella, L. M., (2002). Ant Algorithms for solving weapon-target at assignment problem. *Applied Soft Computing ,* 2,39-47.

Dorigo, M., Maniezzo, V., & Colorni, A., (1996). *IEEE Transactions On Systems, Man, And Cybernetics-Part B Cybernetics*, 26 (1).

Dougherty, M., (1996). A Review of Neural Network Applied to Transport, *Journal of Transportation Research ,* 151 – 1661.

Fayyad, U. M., & .Irani, K. B., (1992). On the handling of continuous valued attributes in decision tree generation. *Machine Learning,* 8, 872102.

Haibin, D., (2005). Theory and Application of Ant Colony Algorithm [M], *Science Publish*, 12.

Meyer, B., (2005). Constraint Handling and stochastic Ranking in ACO, *IEEE Transactions on Evolutionary Computation,* 5, 2683-2690.

Parpinelli R. S., Lopes, H. S., & Freitas A. A., (2006). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321~332.

Quinlan, J. R., (1986). Induction of decision tree. *Machine Learning,* I (1), 812106.

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

Smith, B. L., & Demetsky, M. J., (1997). Traffic Flow Forecasting: Comparison of Modeling Approaches *Journal of Transportation Engineering* , 261 – 2661.

Stutzle.T, & Hoos, H., (1997). Max-Min Ant System and Local Search for the Traveling Salesman Problem, *IEEE International Conference on Evolutionary Computation,* 309-314.