Learning Multi-tree Classification Models with Ant Colony Optimization

Khalid M. Salama and Fernando E. B. Otero
School of Computing, University of Kent, Canterbury, U.K.

Keywords: Ant Colony Optimization (ACO), Data Mining, Classification, Decision Trees, Multi-trees.

Abstract: Ant Colony Optimization (ACO) is a meta-heuristic for solving combinatorial optimization problems, inspired by the behaviour of biological ant colonies. One of the successful applications of ACO is learning classification models (classifiers). A classifier encodes the relationships between the input attribute values and the values of a class attribute in a given set of labelled cases and it can be used to predict the class value of new unlabelled cases. Decision trees have been widely used as a type of classification model that represent comprehensible knowledge to the user. In this paper, we propose the use of ACO-based algorithms for learning an extended multi-tree classification model, which consists of multiple decision trees, one for each class value. Each class-based decision trees is responsible for discriminating between its class value and all other values available in the class domain. Our proposed algorithms are empirically evaluated against well-known decision trees induction algorithms, as well as the ACO-based Ant-Tree-Miner algorithm. The results show an overall improvement in predictive accuracy over 32 benchmark datasets. We also discuss how the new multi-tree models can provide the user with more understanding and knowledge-interpretability in a given domain.

1 INTRODUCTION

Data mining is an active research area involving the development and analysis of algorithms for extracting interesting knowledge (or patterns) from real-world datasets. Classification is a central problem in the data mining and machine learning fields, where the goal is to build a model (classifier) using a set of labelled cases (of which the class values are known) that captures the relationships between the input attribute values and the values of the target (class) attribute. The classifier is then used to predict the class of new, unlabelled cases (of cases of which the class values are unknown). While the literature includes several types of classification models, such as classification rules, artificial neural networks, support vector machines and Bayesian network classifiers (Witten and Frank, 2010), in this paper we focus on the popular and widely used classification models, namely decision trees.

Decision trees (Tan et al., 2005; Han and Kamber, 2000) are widely used as a comprehensible representation model, given that they can be easily represented in a graphical form and also be represented as a set of classification rules, which generally can be expressed in the form of IF-THEN rules. In a decision tree, the internal nodes correspond to attribute tests (decision nodes) and leaf nodes correspond to predicted class labels. In order to classify a case, the tree is traversed from the root node towards a leaf node by selecting branches according to internal nodes' tests, until a leaf node (class prediction) is reached.

Top-Down Induction of Decision Trees (TDIDT) is the most common approach in the literature of learning decision trees, which employs a *divide-and-conquer* of approach: 1) select the best attribute to use as internal node of the tree at a certain level; 2) divide the dataset into several subsets according to the branches (values) of the selected node (attribute); and 3) recursively apply the previous steps on each subset until all cases from the subset have the same class label or when another stopping criterion is satisfied, creating a leaf node to represent a class label to be predicted. The well-known ID3, C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984) algorithms follow such approach for learning decision trees.

Nonetheless, the divide-and-conquer approach represents a deterministic, greedy search strategy to create a decision tree; the selection of the best attribute is made locally at each iteration, without taking into consideration its influence over the subsequent iterations. This makes the algorithm prone to get stuck in local optima. While several heuristics (functions) for choosing attributes have been used in the literature, such as Gini Index, Information Gain, and Gain Ratio (Tan et al., 2005), the idea of utiliz-

Salama K. and Otero F..
Learning Multi-tree Classification Models with Ant Colony Optimization.
DOI: 10.5220/0005071300380048
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2014), pages 38-48
ISBN: 978-989-758-052-9

ing stochastic, global search algorithms – that are less likely to get stuck into local optima – to learn decision trees is under-explored; only three related papers can be found in the literature (see Section 2.2).

Ant Colony Optimization (ACO) is a meta-heuristic for solving combinatorial optimization problems, inspired by the behaviour of biological ant colonies. Ant-Tree-Miner (Otero et al., 2012) is a recently introduced ACO-based algorithm for inducing decision trees. In this paper, we extend the Ant-Tree-Miner algorithm to build a new multi-tree classification model. In essence, a multi-tree model consists of several class-based decision trees, where each tree is responsible for discriminating between a specific class value and all other values in the class domain. The multi-tree model can provide the user with a potentially useful representation of the knowledge discovered from the dataset, by focusing on the specific patterns describing each class value and differentiating it from the other ones.

In this context, we propose two new ACO-based algorithms to build multi-tree classifiers: 1) Ant-Tree-Miner$_{ML}$, which learns one class-based tree at a time in a local approach, and 2) Ant-Tree-Miner$_{MI}$, which learns a complete multi-tree model in an integrated approach. Our proposed algorithms are empirically evaluated against the C4.5 and CART decision trees induction algorithms, as well as the original Ant-Tree-Miner. The results show that Ant-Tree-Miner$_{ML}$ and Ant-Tree-Miner$_{MI}$ achieve an overall improvement in predictive accuracy over 32 benchmark datasets.

The rest of the paper is organized as follows. Section 2 provides an overview on ACO and its related work in the classification learning domain, focusing on the Ant-Tree-Miner decision tree induction algorithm. In Section 3, we describe the multi-tree classification model. We introduce our two new ACO algorithms for learning multi-trees in Section 4. The experimental methodology and results are discussed in Section 5, followed by concluding remarks and further research directions in Section 6.

2 CLASSIFICATION WITH ANT COLONY ALGORITHMS

2.1 Ant Colony Optimization Overview

Inspired by the behaviour of natural ant colonies, Dorigo et al. (Dorigo and Stützle, 2004; Dorigo and Stützle, 2003) have defined Ant Colony Optimization (ACO) as a meta-heuristic that has been widely applied to solve combinatorial optimization problems. The basic principle of ACO is that a population of artificial ants cooperates with each other to find the best path in a graph, representing a candidate solution to the target problem, analogously to the way that natural ants cooperate to find the shortest path between two points like their nest and a food source. The outline of the basic ACO procedures is presented in Algorithm 1.

Algorithm 1: Pseudo-code of basic ACO algorithm.

```
Begin ACO
ConstructionGraph ← Problem_definition;
Initialize();
best ← ϕ; /* best solution found so far */
repeat
    current ← ant.ConstructSolution();
    ApplyLocalSearch(current); /* Optional */
    if Quality(current) > Quality(best) then
        best ← current;
    end if
    ant.UpdatePheromone(current);
until termination_condition
return best;
End
```

In order to design an ACO algorithm to solve a specific type of problem, the following components of the algorithm should be designed:

Construction Graph - This graph defines the search space to be explored by the ACO algorithm, which consists of the various components (nodes or edges) by which an ant constructs a solution to the target problem.

Heuristic Function - This function uses problem-specific information to locally evaluate the quality of each candidate solution component available in the construction graph.

State Transition Rule - This probabilistic transition rule determines how each ant decides which solution component will be visited next in the construction graph, in order to continue the creation of its candidate solution. This rule is based on both the heuristic function value η and the pheromone amount τ associated with the available components.

Quality Evaluation Function - This is a problem-specific function used to evaluate the quality of a candidate solution constructed by an ant. The higher the quality of a solution, the more pheromone will be deposited on the construction graph components used in that solution.

Local Search - An optional procedure to improve the quality of a constructed solution. This can be performed on each constructed candidate solution, or just on the best solution among the colony to reduce com-

putational time.

In ACO, an ant incrementally constructs a candidate solution by visiting nodes and edges in the graph, creating a search path. An ant deposits pheromone on the nodes or edges of the path corresponding to its constructed solution, where the amount of pheromone deposited is proportional to the quality of that solution. The higher the amount of pheromone on a node or edge, the higher the probability that other ants will decide to visit that node or edge when constructing their solution. The iterative process of solution construction, quality evaluation, and pheromone update incorporates an aspect of global search into an ACO algorithm, which makes it less likely to get trapped into local optima than conventional greedy algorithms.

2.2 ACO Related Work

ACO has been successful in tackling the classification problem of data mining. A number of ACO-based algorithms have been introduced in the literature with different classification learning approaches. Ant-Miner (Parpinelli et al., 2002), is the first ant-based classification algorithm, which discovers a list of IF-THEN classification rules.

Ant-Miner is a sequential covering ACO-based rule induction algorithm, where in which iteration the algorithm, the ACO meta-heuristic is utilized to discover the best classification rule using the current training set. The rule is then add the discovered rule list, and all the instances covered by this rule are removed from the training set. The algorithms continuous until all the training instances are covered by the discovered rules, or until the maximum number of iteration is reached.

The Ant-Miner algorithm has been followed by several extensions. Ant-Miner+ (Martens et al., 2007) used the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System (Stützle and Hoos, 1997), the class is selected before the antecedent construction. The Multi-pheromone Ant-Miner (Salama et al., 2011; Salama et al., 2013) introduced the idea of class-based pheromone. That is, the ant select class values prior to the rule antecedent construction process, and the ant is only influenced by the pheromone dropped by previous ant that constructed rules with the same current class value, so that pheromone information is not shared by ant constructing rules with different consequent classes.

*c*Ant-Miner (Otero et al., 2009) is the version of the Ant-Miner algorithm that copes with continues attributes, by dynamically creating intervals that best fits the rule that is currently being constructed. Recently, *c*Ant-Miner$_{\text{PB}}$ (Otero et al., 2013; Otero and Freitas, 2013) was introduced as a new sequential covering strategy for ACO classification algorithms, where the whole rule list is constructed in one ant iteration. The order of the rules is implicitly encoded as pheromone values and the search is guided by the quality of a candidate list of rules.

In addition, ACO has also been employed to learn various types of Bayesian network classifiers. ABC-Miner (Salama and Freitas, 2013d) learns Bayesian Augmented Naïve-Bayes (BAN) classifier, while ABC-Miner+ (Salama and Freitas, 2013c) builds Markov Blanket classifiers, which the algorithms perform embedded feature selection and produces more flexible Bayesian network structures. ACO for learning Bayesian Multi-nets is introduced in (Salama and Freitas, 2013a; Salama and Freitas, 2013b).

ACO-PB (Liu et al., 2006) ia a hybrid of the ant colony and back-propagation algorithms to optimize the network weights. It adopts ACO to search the optimal combination of weights in the solution space, and then uses BP algorithm to obtain the accurate optimal solution quickly. ACO$_{\mathbb{R}}$, an ant colony optimization algorithm for continuous optimization (Socha and Dorigo, 2008), was used to train feed-forward neural networks (Socha and Blum, 2005; Socha and Blum, 2007).

As for learning decision trees, only ACDT (Boryczka and Kozak, 2010; Boryczka and Kozak, 2011), and Ant-Tree-Miner (Otero et al., 2012), were introduced as ACO-based algorithms. Since our algorithms are built upon Ant-Tree-Miner, it is described in more details in the following subsection.

2.3 The Ant-Tree-Miner Algorithm

The Ant-Tree-Miner algorithm (Otero et al., 2012) follows the traditional divide-and-conquer approach, with the difference that an ACO procedure is used during the tree construction to select the nodes (attributes) of the tree. Instead of applying a greedy deterministic selection, Ant-Tree-Miner uses a stochastic process based on heuristic information and pheromone values. To create a candidate decision tree DT, an ant starts by selecting an attribute from the construction graph. The probability of selecting an attribute is based on both their heuristic function value η and the pheromone amount τ.

Each entry in the pheromone matrix is represented by a triple $[E_{ij}, L, x_k]$, where E_{ij} is the edge representing the j-th attribute condition of the attribute x_i in the construction graph, L is the level of the decision tree where the E_{ij} appears and x_k is its destination attribute vertex. The level information of the edge is associated with an entry to discriminate between multiple

occurrences of the same type of edge (i.e., the same attribute condition) at different levels of the tree, either for occurrences in the same tree path – possible in the case of edges of continuous attribute vertices – or in different tree paths.

Once an attribute is selected to represent a decision node, branches corresponding to each attribute-condition are created. At this point, the training set is divided into one subset for each branch, where each subset contains the training cases satisfying the attribute-condition represented by the branch. When an ant follows a branch, it checks whether a leaf node should be added or if it should recursively add a sub-tree below its current branch. An ant chooses to add a leaf node if one of the following conditions occur:

1. the branch's subset is empty —i.e., no training case satisfy the branch's condition;
2. all cases in the subset have the same class label;
3. the subset size is smaller than or equal to the `min_cases_per_branch` user-defined value;
4. there are no more (unused) attributes to be selected.

If none of the above conditions is observed, the construction procedure is applied recursively to create a sub-tree given the subset of training cases.

When all ants create a decision tree, they are evaluated and the iteration-best tree (DT^{tbest}) is used to update pheromone values. The creation process is repeated until a user-defined maximum number of iterations is reached or the algorithm has converged (i.e., the pheromone values lead the algorithm to create the same solution). The pheromone update procedure is given by:

$$\tau_{(E,L,x_k)} = \begin{cases} \rho.\tau_{(E,L,x_k)}, \ \ if\ (E,L,x_k) \notin DT^{tbest}; \\ \rho.\tau_{(E,L,x_k)} + Q(DT^{tbest}), \\ \qquad\qquad if\ (E,L,x_k) \in DT^{tbest}; \end{cases} \quad (1)$$

where ρ is the `evaporation_factor` parameter, $\tau_{(E,L,x_k)} - E$ is the attribute condition of the edge that this pheromone entry corresponds to, L is the level in which the edge occurs and x_k s the edge's destination attribute vertex – and $Q(DT)$ is the quality of the candidate constructed decision tree DT.

3 MULTI-TREE MODELS

Unlike a decision tree classification model, which represents the induced knowledge from a given dataset as a tree model, the multi-tree model represents the induced knowledge in several decision trees, one from each class value perspective. More precisely, a multi-tree model MT consists of a set of separate local decision trees $\{DT_1, DT_2, ...DT_{|C|}\}$, where $|C|$ is the number of the available class values. DT_l is responsible for discriminating between a class value l and all the other values in the class domain. In other words, each tree in the multi-tree model treats its related class values as the positive class, and all the other class values as one negative class. Hence, DT_l concerned with classifying a case to one of two class values: one natural positive class value ($C = l$), and other artificial negative class value ($C \neq l$). Thus, the knowledge captured in each tree of a multi-tree model only describes the attribute-value relationships that are relevant to specific class value, regardless of the other relationships that might be relevant to the other classes.

Therefore, for each D_l, an induction algorithm will only focus on discovering the specific patterns that describe class value l, and will not get distracted by patterns related to other classes. This should make the induction process of D_l effective in terms of discriminating l from other class values. This is opposite to the conventional decision tree induction algorithms that try to discriminate between all the class values in a single model.

In essence, in order to predict the label of a new test case x using a multi-tree classification model, x is classified using each tree DT_l in the model. Each DT_l produces a classification output: $P(C = l|x)$, which is the probability that x belongs to local positive class l. The outputs of the different trees in a multi-tree can take on of the following forms:
– $P(C = l|x) > 0.5$ in only one tree in the set, then x is assigned to class l;
– $P(C = l|x) > 0.5$ in multiple trees in the set, then x is assigned to class value l of the tree that has the highest probability;
– $P(C = l|x) \leq 0.5$ for all the trees in the set, then x is also assigned to class value l of the tree that has the highest probability as well.

4 LEARNING MULTI-TREES WITH ACO ALGORITHMS

4.1 A Local ACO Approach for Learning Multi-trees

Ant-Tree-Miner$_{\text{ML}}$ is the first ACO algorithm we propose in this paper to build multi-tree classification models. The algorithm employs a local approach to build a multi-tree, in which each tree in the model is

learnt in a one-at-a-time fashion. In other words, there is no dependency between the processes of learning different trees in the model — each tree induction process is considered as a separate optimization problem, carried out by a separate ACO procedure. Algorithm 2 shows the overall process of Ant-Tree-Miner$_{\mathrm{ML}}$.

Algorithm 2 : Pseudo-code of Ant-Tree-Miner$_{\mathrm{ML}}$.

```
1:  Begin
2:  D = training_set; MT ← ϕ;
3:  for l = 1 to |C| do
4:      D_l = GetBinaryDataset(D, l);
5:      InitializeConstructionGraph();
6:      DT_l^gbest = ϕ;
7:      Q^gbest = 0;
8:      t = 1;
9:      repeat
10:         DT_l^tbest = ϕ; Q^tbest = 0;
11:         for i = 1 to colony_size do
12:             DT_l^i = ant^i.CreateTree(D_l);
13:             Q^i = EvaluateQuality(DT_l^i, D_l);
14:             if Q^i > Q(tbest) then
15:                 BN_l^tbest = BN_l^i;
16:                 Q^tbest = Q^i;
17:             end if
18:         end for
19:         PruneTree(DT_l^tbest);
20:         UpdatePheromone(DT_l tbest);
21:         if Q^tbest > Q^gbest then
22:             DT_l^gbest = DT_l^tbest;
23:             Q^gbest = Q^tbest;
24:         end if
25:         t ← t + 1;
26:     until t = max_iterations or Convergence()
27:     append DT_l^gbest to MT;
28: end for
29: return MT;
30: End
```

The following course of actions are repeated for each class label l available in the class domain C (lines 3 to 28), in order to produce a decision tree for each class. For class l, a new modified binary class training dataset copy DT_l is generated from the original training dataset D (line 4). In each new dataset D_l, cases labelled with class l are considered positive cases; all the cases that are labelled with class values other than l are considered the negative cases, and re-labelled to class l^-. Hence, cases in each D_l will be either labelled with l or l^-.

At this point, an ACO procedure is responsible inducing a decision tree on the new binary dataset. In essence, a construction graph (similar to the one discussed in Section 2.3) is constructed and the pheromone amounts are initialized as is done in Ant-Tree-Miner (Otero et al., 2012). Now, each ant^i in the colony creates a candidate decision tree DT_l^i (line 12), using the decision tree construction procedure described in Section 2.3. Then, the predictive accuracy of DT_l^i is evaluated, however, using its related new training dataset D_l (line 13). That is, a correct classification occurs when DT_l^i correctly classify a case to either positive (l) or negative (l^-) class (regardless of the specific negative classes). Predictive accuracy (the quality Q^i of the current local decision tree DT_l^i) is calculated as the number of correctly classified instances by DT_l^i over the total number of instance in the binary training dataset D_l.

The iteration-best DT_l^{tbest} (best tree constructed in the colony during the iteration) is selected to undergo pruning and perform pheromone update, using the same error-based pruning procedure and level-based pheromone update strategy used in the Ant-Tree-Miner algorithm, with respect to D_l. The global best solution DT_l^{gbest} is updated – if possible – at the end of each iteration (lines 21 to 23).

This set of steps is considered one iteration of the outer repeat-until loop (lines 9 to 26) and it is iteratively performed is repeated until the same DT is generated for a number of consecutive trials specified by the `conv_iterations` parameter (indicating convergence) or until `max_iterations` is reached (see Table 1 for parameter settings). The induced decision tree DT_l^{gbest} for class label l is appended to the multi-tree model MT and then the algorithm moves to build the next decision tree for class label $l+1$.

4.2 An Integrated ACO Approach for Learning Multi-trees

The second algorithm proposed in this paper follows an integrated approach. Ant-Tree-Miner$_{\mathrm{MI}}$ works in a different way from its local counterpart. A single run of ACO is used to build the whole solution; each ant builds a *complete* multi-tree classification model as a candidate solution at once. This is accomplished by building a candidate local DT_l for each class value l in each single ant trial, appending them to current candidate multi-tree model.

Unlike Ant-Tree-Miner$_{\mathrm{ML}}$, which has to completely finish building the local tree DT_l before starting to build DT_{l+1}, in Ant-Tree-Miner$_{\mathrm{MI}}$ the whole multi-tree model is constructed before performing the quality evaluation and pheromone update. In this case, the integrated approach is not concerned with the quality of each individual decision tree DT_l—in classifying the artificial training dataset D_l—per se,

rather it is concerned with the quality of the complete multi-tree model when used to classify the original training set D.

Algorithm 3 : Pseudo-code of Ant-Tree-Miner$_{\mathrm{MI}}$.

```
1:  Begin
2:  D = training_set; MT^gbest = ϕ;
3:  Q^gbest = 0;
4:  t = 1;
5:  InitializeConstructionGraphs();
6:  repeat
7:      MT^tbest = ϕ; Q^tbest = 0;
8:      for i = 1 to colony_size do
9:          MT^i ← ϕ;
10:         for l = 1 to |C| do
11:             D_l = GetBinaryDataset(D, l);
12:             DT_l^i = ant^i.CreateTree(D_l);
13:             append DT_l^i to MT^i;
14:         end for
15:         Q^i = EvaluateQuality(MT_l^i, D);
16:         if Q^i > Q^tbest then
17:             MT^tbest = MT^i;
18:             Q^tbest = Q^i;
19:         end if
20:     end for
21:     PruneMultitree(MT^tbest);
22:     UpdatePheromone(MT^tbest);
23:     if Q^tbest > Q^gbest then
24:         MT^gbest = MT^tbest;
25:         Q^gbest = Q^tbest;
26:     end if
27:     t = t + 1;
28: until t = max_iterations or Convergence()
29: return MT^gbest;
30: End
```

As shown in Algorithm 3, each ant^i in the colony creates a candidate solution MT^i—i.e., a complete multi-tree model (lines 10 to 13). Then, the quality of the constructed MT^i is evaluated on the original training set D (line 15). The best solution MT^{tbest} produced in the colony at iteration t is selected to undergo pruning before the ant updates the pheromone trail according to the classification quality Q^{tbest} of the multi-tree MT^{tbest} (line 21 and 22). Finally, MT^{tbest} is compared with the global best solution MT^{gbest} to keep track of the best multi-tree found so far (lines 23 to 26). This is iteratively repeated until the termination conditions are met. At the end, MT^{gbest} is considered the final induced multi-tree classification model.

In order to apply such an integrated approach using ACO, we use several construction graphs for a given dataset. More precisely, we use $|C|$ construction graphs, one for each class value $l \in C$. Pheromone amounts and heuristic information initialization are performed on all the used construction graphs. As for solution creation, an ant uses each construction graph to build each local DT_l, one for each class label l. Once an ant creates the local DT_l, it proceeds to the construction of the local DT_{l+1}, using the $(l+1)th$ construction graph and the D_{l+1}.

The tree pruning procedure is applied on each tree DT_l in the the iteration-best MT^{tbest} multi-tree model. As for pheromone update, it is performed on all construction graphs according to the quality of the constructed MT^{tbest} as a whole.

5 EXPERIMENTS AND RESULTS

We compare the predictive performance of the two proposed ACO algorithms for learning multi-tree classification models against the well-known CART and C4.5 decision trees induction algorithms. In our experiments, we used the WEKA (Witten and Frank, 2010) implementations for these algorithms, SimpleCART and J48, respectively.

In addition, we implemented a greedy algorithm for learning multi-tree models, C4.5-MTree, as another baseline for our comparisons. In essence, several binary datasets are generated for a given dataset, one for each available class value (as described in Section 4.1). Then, the C4.5 algorithm is used to induce a decision tree from each binary dataset. The final multi-tree model would be the set of the induced decision trees. We also compare the predictive performance of the multi-tree models produced by the proposed algorithms against the decision trees produced by the Ant-Tree-Miner algorithm.

The experiments were carried out using the *stratified* ten-fold cross validation procedure. In essence, a dataset is divided into ten mutually exclusive partitions (folds), with approximately the same number of cases in each partition. Then each classification algorithm is run ten times, where each time a different partition is used as the test set and the other nine partitions are used as the training set. The results (accuracy rate on the test set) are then averaged and reported as the accuracy rate of the classifier. For the ACO algorithms, we run them ten times – using a different random seed to initialize the search each time – for each of the ten iterations of the cross-validation procedure (i.e., 100 runs in total, for each dataset). In the case of the deterministic algorithms, each one is run just once for each iteration of the cross-validation procedure.

The parameter configuration used in our experiments is shown in Table 1. Note that the

max_iterations parameter refers to the maximum number of iterations used to build a single local tree in the Ant-Tree-Miner$_{ML}$ algorithm. The value of this parameter is multiplied by the number of class values when used with Ant-Tree-Miner$_{MI}$. However, this maximum number is not fully tilized, since the algorithms converge earlier. The WEKA default parameter settings were used for CART and C4.5.

Table 1: ACO Parameter settings used in experiments.

Parameter	Value
max_iterations	500
colony_size	50
conv_iterations	10
min_cases_per_branch	3
evaporation_factor	0.9

5.1 Predictive Accuracy Results

The experimental evaluation was performed using 32 public-domain datasets from the University of California at Irvine (UCI) dataset repository (Asuncion and Newman, 2007). The main characteristics of the datasets are shown in Table 2

Table 3 reports the average predictive accuracy values obtained by ten-fold cross validation for the 32 datasets, where the highest accuracy value for each dataset is shown in bold face. The last row shows the average rank of each algorithm in terms of predictive accuracy according to the non-parametric Friedman test (Garca and Herrera, 2008). The average rank for a given algorithm *g* is obtained by first computing the rank of *g* on each dataset individually. The individual ranks are then averaged across all datasets to obtain the overall average rank. Note that the lower the value of the rank, the better the algorithm performance.

As shown in Table 3, the proposed Ant-Tree-Miner$_{MI}$—which uses the ACO integrated approach to learn multi-tree models—obtained the best overall average ranking of 2.64 and achieved the highest predictive accuracy in 8 datasets. In the second place is Ant-Tree-Miner$_{ML}$—the local approach ACO algorithm for learning multi-tree models—obtained an overall average ranking of 2.89 and achieved the highest predictive accuracy in 6 datasets. Ant-Tree-Miner—the ACO algorithm for learning decision trees—came in the third place with overall average ranking of 2.95 and achieved the highest predictive accuracy in 5 datasets. The C4.5 And CART decision tree induction algorithms came in the fourth and the fifth place with overall average rankings of 4.05 and 4.1, and achieved the highest predictive accuracy in 6 and 9 datasets, respectively. The multi-tree C4.5-MTree obtained an overall average ranking of 4.28 and performed last.

Table 2: Description of datasets.

Dataset	Instances	Attributes	Classes
annealing	896	38	6
audiology	200	70	24
balance	625	4	3
breast-l	286	9	2
breast-p	198	32	2
breast-tissue	106	9	2
breast-w	569	30	2
car	1,728	6	4
credit-a	690	14	2
credit-g	1,000	20	2
dermatology	366	33	6
heart-c	303	12	3
heart-h	293	13	5
hepatitis	155	19	2
horse	356	22	2
ionosphere	351	34	2
lymphography	148	18	4
monks	432	6	2
parkinsons	197	23	2
pima diabetes	768	8	2
pop	90	8	3
s-heart	267	22	2
segmentation	2,310	19	7
soybean	307	35	19
thyriod	215	5	3
transfusion	722	4	2
tic-tac-to	958	9	2
voting	435	16	2
wine	178	13	3
zoo	101	17	7

The non-parametric Friedman statistical test (Garca and Herrera, 2008) with the Holm's post-hoc test was applied to the predictive accuracy results shown in Table 3. The test showed that both Ant-Tree-Miner$_{ML}$ and Ant-Tree-Miner$_{MI}$ are statistically better than CART, C4.5 and C4.5-MTree with a significance level of 5%. Besides that, Ant-Tree-Miner is shown to be statistically better than CART, C4.5 and C4.5-MTree with a significance level of 10%. Table 4 shows the results of non-parametric Friedman Statis-

Table 3: Average predictive accuracy for each algorithm in the 32 datasets. The columns annotated with ATM correspond to Ant-Tree-Miner and the proposed multi-tree extensions.

Dataset	CART	C4.5	C4.5-MTree	ATM	ATM_{ML}	ATM_{MI}
annealing	94.1	91.1	93.49	95.2	**95.31**	94.61
audiology	80	**88.33**	81.67	80	82.17	**88.33**
automobile	75.91	81.4	77.48	77.17	79.57	**82.95**
balance	68.39	63.83	65.67	74.5	78.5	**80.67**
breast-l	72.39	72.86	72.86	72.91	73	**73.31**
breast-p	**76.29**	64	64	76.29	76.29	76.29
breast-tissue	62.46	59.6	60.55	64.45	64.55	**69.18**
breast-w	92.61	**94.56**	94.56	91.38	92.1	94.2
car	**97.63**	93.27	93.74	93.8	93.39	93.8
credit-a	84.93	86.38	86.38	86.64	**86.65**	86.2
credit-g	**75.5**	70.9	70.9	71.3	69.9	69.9
cylinder	71.48	73.75	73.75	73.92	**75.08**	72.98
dermatology	94.01	93.5	91.79	**95.35**	95.35	94.53
heart-c	51.46	51.2	53.86	54.46	52.82	**54.81**
heart-h	63.97	66.7	61.65	**67.45**	67.38	66.13
hepatitis	78.04	79.38	79.38	81.29	**83.83**	83.29
horse	**85.02**	85.02	85.02	83.57	82.71	82.98
ionosphere	89.18	89.67	89.67	90.21	90.23	**90.26**
lymphography	74.43	77.71	78.57	77.8	**79.19**	78.8
monks	**66.55**	60.73	60.73	61.64	63.64	62.73
parkinsons	85	83.58	83.58	84.58	**87.63**	85.58
pima	72.92	**74.73**	74.73	71.23	69.92	69.54
pop	73.75	**75**	75	75	73.75	75
s-heart	**77.78**	74.81	74.81	75.56	77.41	77.78
segmentation	95.37	**96.76**	95.02	96.23	95.39	95.67
soybean	86.32	85.86	83.1	**86.55**	86.21	86.55
thyroid	91.58	91.6	89.76	91.6	91.69	**92.14**
transfusion	71.27	**74.56**	74.56	72.6	71.72	71.93
tic-tac-to	**92.28**	85.58	85.58	87.05	89.68	92.11
voting	**95.41**	94.48	94.48	94.91	94.94	94.91
wine	90.98	93.3	92.75	**93.82**	93.3	93.82
zoo	90.09	95	97.5	**100**	97.5	97.5
Average rank	**4.1**	**4.05**	**4.28**	**2.95**	**2.89**	**2.64**

Table 4: The results of non-parametric Friedman Statistical test. *p*-values are shown in bold-face, while Holm values are shown between brackets.

Algorithm	ATM	ATM_{ML}	ATM_{MI}
CART	**0.017** (0.141)	**0.012** (0.122)	**0.002** (0.026)
C4.5	**0.019** (0.141)	**0.013** (0.122)	**0.002** (0.028)
C4.5-MTree	**0.005** (0.061)	**0.003** (0.043)	**4.5E-4** (0.006)

tical test.

One notice regarding the results of C4.5-MT, which uses the C4.5 in a local approach to build multi-tree models, is that it did not performed, overall, better than C4.5 decision tree induction algorithm. One important reason is the function used to select the best attribute to create an internal tree node, and its attribute values become branches to split the training set into more pure data subset relative to the class values. C4.5 uses Information Gain Ration (Quinlan, 1993), which is proper for reducing the entropy (impurity) of the data subsets with respect to *all* the class values. However, a multi-tree induction algorithm may need to use a different function that focus on the quality

Table 5: Average size (in terms of the total number of rules' terms extracted from each model) results. The ratio results represent the size results ratio of a multi-tree induction algorithm to the size results of its corresponding decision tree induction algorithm.

Dataset	C4.5	C4.5-MTree		ATM	ATM_{ML}		ATM_{MI}	
	size	size	ratio to C4.5	size	size	ratio to ATM	size	ratio to ATM
annealing	300.7	210.5	**0.7**	388.1	280.6	**0.72**	271.8	**0.7**
audiology	234.8	139.8	**0.6**	251.6	130.1	**0.52**	141	**0.56**
automobile	214.8	102.2	**0.48**	325	169.5	**0.52**	204.7	**0.63**
balance	84.2	71.8	**0.85**	110.3	91.2	**0.83**	108.9	**0.99**
breast-l	16.4	16.4	**1**	116.6	88	**0.75**	126.6	**1.09**
breast-p	43.1	99.9	**2.32**	2	2	**1**	2	**1**
breast-tissue	66	36.1	**0.55**	104.7	69.4	**0.66**	93.3	**0.89**
breast-w	47.7	50.9	**1.07**	82.9	77.3	**0.93**	81.1	**0.98**
car	605.4	492	**0.81**	612.8	499.8	**0.82**	524.9	**0.86**
credit-a	103.3	90.6	**0.88**	267.1	272.3	**1.02**	252.6	**0.95**
credit-g	507	543.4	**1.07**	775.2	856.4	**1.1**	812.7	**1.05**
cylinder	489.2	449.4	**0.92**	575.3	617.3	**1.07**	587.7	**1.02**
dermatology	118	73.3	**0.62**	101.4	96.6	**0.95**	79	**0.78**
heart-c	260.5	121.1	**0.46**	302.9	126.6	**0.42**	72.7	**0.24**
heart-h	156.7	14.5	**0.09**	172.2	52.5	**0.3**	50.1	**0.29**
hepatitis	37.9	39.5	**1.04**	37.1	39.6	**1.07**	46.8	**1.26**
horse	11.6	16.4	**1.41**	49.5	39.4	**0.8**	47	**0.95**
ionosphere	75.7	88.8	**1.17**	85.9	83.9	**0.98**	103.7	**1.21**
lymphography	67.8	49.6	**0.73**	60.9	58.4	**0.96**	63.7	**1.05**
monks	18.9	18.9	**1**	8.5	1.1	**0.13**	2.5	**0.29**
parkinsons	51.8	52	**1**	67.1	64.5	**0.96**	61.5	**0.92**
pima	123.9	237.3	**1.92**	748.3	676.9	**0.9**	686.8	**0.92**
pop	2	2.4	**1.2**	2	3.6	**1.8**	3.6	**1.8**
s-heart	82.9	89.1	**1.07**	130.8	126.6	**0.97**	121.1	**0.93**
segmentation	315.9	250.4	**0.79**	442.1	535.2	**1.21**	458.8	**1.04**
soybean	330.8	80.4	**0.24**	353.5	130.2	**0.37**	155.3	**0.44**
thyroid	30.3	26.6	**0.88**	36.1	30.9	**0.86**	40.3	**1.12**
transfusion	24.9	36.6	**1.47**	112.8	93.2	**0.83**	109.3	**0.97**
ttt	396.5	396.5	**1**	536	507.2	**0.95**	539.8	**1.01**
voting	18.2	18.2	**1**	25.7	18.3	**0.71**	27.1	**1.05**
wine	12.8	19.4	**1.52**	27.7	24.2	**0.87**	26.6	**0.96**
zoo	43.8	28.8	**0.66**	66.7	19.1	**0.29**	23.8	**0.36**
Average size/ratio	152.9	123.8	**0.95**	218.1	183.8	**0.82**	1852	**0.88**

of the split with respect to the *specific* (positive) class value for which the current local tree is built. This can also be beneficial in class imbalance problems, where one (or more) class values has a low occurrence in the dataset compared to other class values. Precision and Recall measures are candidate functions for this task, yet we leave this for future investigation.

On the other hand, the ACO algorithms do not have the same problem, since ACO perform a global search to build the tree model as a whole, rather than local decision of selecting the next attribute to add to the tree model.

5.2 Model Comprehensibility Results

Although predictive accuracy improvements that multi-tree induction ACO-based algorithms made over the Ant-tree-Miner decision tree induction algorithm are not statistically significant according to the used test, and therefore are regarded as similar, we

claim that our proposed algorithm have advantages regarding comprehensibility of their produced model. It is controversial to state which one is more easily comprehensible from the user perspective: one relatively large decision tree model (produced by a decision tree induction algorithm) or several relatively small class-based trees in the multi-tree models.

Therefore, for fair size comparison, we extracted a set of rules from the constructed models and we compared the total number of the terms in these rules. In essence, for decision tree models, each path from the root node to a leaf node represents a rule, where the rule terms (antecedent) are the node-branch conditions in the path, and the rule consequent is the majority class predicted by the leaf node. As for the multi-tree models, the rules are extracted as follows. First, we extract the rules from each local tree in the same aforementioned way. However, for each tree D_l, we only keep the rules where the consequent is the tree's positive class l and discard the rules where the consequent is the tree's negative class l^-. Second, since there may be two (or more) rules—with different consequent classes—that can classify the same instance (i.e., one of them contains all the antecedent terms of the other), we only keep the rule with the highest consequent class probability.

The size results are in Table 5, which shows that, overall, the total terms in the rules extracted from the multi-tree model is smaller than the total terms in the rules extracted from the single decision tree model built for the same dataset, supporting our claim that the multi-tree model provides a model that is simpler to interpret. This is shown in the ratio of each multi-tree induction algorithm's size results to its corresponding decision tree induction algorithm's size results.

6 CONCLUDING REMARKS

In this paper, we have introduced two new ACO-based algorithm for learning multi-tree classification model. A multi-tree model consists of several class-based decision trees, each is responsible for discriminating between a specific class value and all other values in the class domain. We empirically evaluated the performance of our algorithms over 32 benchmark UCI datasets and compared their results against the results of well-known CART and C4.5 decision trees induction algorithms, as well as the Ant-Tree-Miner ACO-based algorithm. In addition, we compared our results to C4.5-MT, a greedy implementation for learning multi-tree models.

The results showed that the two proposed ACO algorithms are statistically significantly better than the three greedy algorithms and produces comparable predictive accuracy results compared to Ant-Tree-Miner. An interesting aspect of the multi-tree models induced by our ACO algorithms is that they can provide the user with a potentially useful view of the knowledge discovered from the dataset by producing several classification trees, each one focusing on the specific patterns describing one class value and differentiating it from the other classes. Moreover, predictions made by the multi-tree model involve a smaller number of terms, which contributes to the comprehensibility of the model. Therefore, the user needs to analyze a smaller number of attribute-value conditions in order understand a prediction.

In the future, an interesting direction is to explore different measures, other than accuracy, to evaluate the quality of the candidate constructed decision trees in the local approach, where the focus might be increasing the true positives rate of each tree rather than improving the its general predictive accuracy. We can also explore more sophisticated quality evaluation measures in both approaches, such as Quadratic Loss Function and Bayesian Information Reward.

REFERENCES

Asuncion, A. and Newman, D. (2007). UCI Machine Learning Repository. URL: http://www.ics.uci.edu/ mlearn/MLRepository.html.

Boryczka, U. and Kozak, J. (2010). Ant Colony Decision Trees. In *4th International Conference on Computational Collective Intelligence: Technologies and Applications (ICCCI'11)*, pages 4373–382, Berlin, Heidelberg. Springer.

Boryczka, U. and Kozak, J. (2011). An Adaptive Discretization in the ACDT Algorithm for Continuous Attributes. In *3rd International Conference on Computational Collective Intelligence: Technologies and Applications (ICCCI'11)*, pages 475–484, Berlin, Heidelberg. Springer.

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984). *Classification and Regression Trees*. Chapman and Hall.

Dorigo, M. and Stützle, T. (2003). The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In *Handbook of Metaheuristics*, volume 57 of *OPRMS*, pages 250–28, New York, NY, USA. Springer.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA.

Garca, S. and Herrera, F. (2008). An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research*, 9:2677–2694.

Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2nd edition.

Liu, Y.-P., Wu, M.-G., and Qian, J.-X. (2006). Evolving neural networks using the hybrid of ant colony optimization and bp algorithms. In *3rd International Conference on Advances in Neural Networks (ISNN'06)*, pages 714–722, Berlin, Heidelberg. Springer-Verlag.

Martens, D., Backer, M. D., Haesen, R., Vanthienen, J., Snoeck, M., and Baesens, B. (2007). Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11:651–665.

Otero, F. and Freitas, A. (2013). Improving the Interpretability of Classification Rules Discovered by an Ant Colony Algorithm. In *Genetic and Evolutionary Computation Conference (GECCO-2013)*, pages 73–80, New York, NY, USA. ACM Press.

Otero, F., Freitas, A., and Johnson, C. (2009). Handling continuous attributes in ant colony classification algorithms. In *IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*, pages 225–231, New York, NY, USA. IEEE Press.

Otero, F., Freitas, A., and Johnson, C. (2013). A New Sequential Covering Strategy for Inducing Classification Rules with Ant Colony Algorithms. *IEEE Transactions on Evolutionary Computation*, 17(1):64–74.

Otero, F. E. B., Freitas, A. A., and Johnson, C. G. (2012). Inducing Decision Trees with an Ant Colony Optimization Algorithm. *Applied Soft Computing*, 12(11):3615–3626.

Parpinelli, R. S., Lopes, H. S., and Freitas, A. A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332.

Quinlan, J. (1993). *Programs for Machine Learning*. Morgan Kaufmann.

Salama, K., Abdelbar, A., and Freitas, A. (2011). Multiple Pheromone Types and Other Extensions to the Ant-Miner Classification Rule Discovery Algorithm. *Swarm Intelligence*, 5(3-4):149–182.

Salama, K., Abdelbar, A., Otero, F., and Freitas, A. (2013). Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery. *Applied Soft Computing*, 13(1):667–675.

Salama, K. and Freitas, A. (2013a). Ant Colony Algorithms for Constructing Bayesian Multi-net Classifiers. *Machine Learning - (Under Review)*.

Salama, K. and Freitas, A. (2013b). Clustering-based Bayesian Multi-net Classifier Construction with Ant Colony Optimization. In *IEEE Congress on Evolutionary Computation (IEEE CEC) (2013)*, pages 3079–3086, New York, NY, USA. IEEE Press.

Salama, K. and Freitas, A. (2013c). Extending the ABC-Miner Bayesian Classification Algorithm. In *6th International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO'13)*, volume 512 of *SCI*, pages 1–12, Berlin. Springer.

Salama, K. and Freitas, A. (2013d). Learning Bayesian Network Classifiers Using Ant Colony Optimization. *Swarm Intelligence*, 7(2-3):229–254.

Socha, K. and Blum, C. (2005). Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In *5th International Conference on Hybrid Intelligent Systems (HIS '05)*, pages 233–238, Washington, DC, USA. IEEE Computer Society.

Socha, K. and Blum, C. (2007). An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training. *Neural Computing & Applications*, 16:235–247.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185:1155–1173.

Stützle, T. and Hoos, H. (1997). MAX-MIN Ant System and local search for the traveling salesman problem. *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314.

Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Addison Wesley, 2nd edition.

Witten, I. H. and Frank, E. (2010). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 3rd edition.