To Appear in Applied Soft Computing (2012), DOI: 10.1016/j.asoc.2012.05.028

Inducing Decision Trees with an Ant Colony Optimization Algorithm

Fernando E. B. Otero*, Alex A. Freitas, Colin G. Johnson

School of Computing, University of Kent, UK

Abstract

Decision trees have been widely used in data mining and machine learning as a comprehensible knowledge representation. While ant colony optimization (ACO) algorithms have been successfully applied to extract classification rules, decision tree induction with ACO algorithms remains an almost unexplored research area. In this paper we propose a novel ACO algorithm to induce decision trees, combining commonly used strategies from both traditional decision tree induction algorithms and ACO. The proposed algorithm is compared against three decision tree induction algorithms, namely C4.5, CART and *c*ACDT, in 22 publicly available data sets. The results show that the predictive accuracy of the proposed algorithm is statistically significantly higher than the accuracy of both C4.5 and CART, which are well-known conventional algorithms for decision tree induction, and the accuracy of the ACO-based *c*ACDT decision tree algorithm.

Keywords: ant colony optimization, data mining, classification, decision tree

1. Introduction

One of the most studied data mining tasks in the literature is the classification task [15, 29]. In essence, the classification task consists of learning a predictive relationship between input values and a desired output. Each example (data instance or record) is described by a set of features (attributes)—referred to as predictor attributes—and a class attribute. Given a set of examples, a classification algorithm aims at creating a model, which represents the relationship between predictor attributes values and class values (labels), and which is able to predict the class label of a new (unseen) example based on the values of its predictor attributes.

Classification problems can be viewed as optimisation problems, where the goal is to find the best function (model) that represents the predictive relationships in the data. A classification problem can be formally specified as:

*Corresponding author

Email addresses: F.E.B.Otero@kent.ac.uk (Fernando E. B. Otero), A.A.Freitas@kent.ac.uk (Alex A. Freitas), C.G.Johnson@kent.ac.uk (Colin G. Johnson)

Given: $\{(e_1, c_{e_1}), \ldots, (e_n, c_{e_n})\}$ pairs representing the training data D, where each e_i denotes the set of predictor attributes' values of the i-th example ($1 \leq i \leq n$, where n is the total number or examples), and each c_{e_i} denotes the class label associated with the i-th example out of m different class labels available in the set C.

Find: a function $f : D \rightarrow C$ that maps each example e_i in D to its correspondent class label c_{e_i} in C.

The main goal of a classification algorithm is to build a model that maximises the predictive accuracy—the number of correct predictions—in the test data (unseen during training), although in many application domains the comprehensibility of the model plays an important role [15, 8, 19]. For instance, in medical diagnosis the classification model should be validated and interpreted by doctors; in credit scoring the classification model should be interpreted by an expert, improving their confidence in the model; in protein function prediction the classification model should be interpreted to provide useful insights about the correlation of protein features and their functions and ultimately improve the current biological knowledge about protein functions. In these domains, it is crucial to produce comprehensible classification models.

Ant colony optimization (ACO) [11, 12, 13] algorithms involve a colony of ants (agents), which despite the relative simplicity of their individuals' behaviours, cooperate with one another to achieve an unified intelligent behaviour. As a result, the colony produces a system capable of performing a robust search to find high-quality solutions for optimisation problems with a large search space. In the context of the classification task in data mining, ACO algorithms have the advantage of performing a flexible robust search for a good combination of predictor attributes, less likely to be affected by the problem of attribute interaction [10, 17].

Decision trees 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 natural language in the form of *IF-THEN* rules. Most ACO algorithms for classification in data mining have focused on extracting classification rules [22, 18]. In this paper, we propose a novel ACO algorithm for the induction of decision trees. The proposed algorithm—called Ant-Tree-Miner (ant colony optimization-based decision tree induction)—is compared against two well-known decision tree induction algorithms, namely C4.5 [31] and CART [6], and the ACO-based *c*ACDT algorithm [5] in 22 publicly available data sets in terms of both predictive accuracy and size of the induced decision trees.

The remainder of this paper is organised as follows. Section 2 presents the background of this paper, discussing the top-down strategy commonly used to induce decision trees, an overview of Ant Colony Optimization (ACO) and the related work on ACO algorithms for induction of tree structures. The proposed algorithm is described in Section 3. The computational results are presented in Section 4. Finally, Section 5 concludes this paper and presents future research directions.

2. Background

2.1. Top-down Induction of Decision Trees

Decision trees provide a comprehensible graphical representation of a classification model, where the internal nodes correspond to attribute tests (decision nodes) and leaf nodes correspond to the predicted class labels—illustrated in Fig. 1. In order to classify an example, the tree is traversed in a top-down fashion from the root node towards a leaf node, moving down the tree by selecting branches according to the outcome of attribute tests represented by internal nodes until a leaf node is reached. At this point, the class label associated with the leaf node is the class label predicted for the example.

A common approach to create decision trees automatically from data is known as the *divide-and-conquer* approach, which consists of an iterative top-down procedure of selecting the best attribute to label an internal node of the tree. It starts by selecting an attribute to represent the root of the tree. After the selection of the first attribute, a branch for each possible (set of) value(s) of the attribute is created and the data set is divided into subsets according to the examples' values of the selected attribute. The selection procedure is then recursively applied to each branch of the node using the corresponding subset of examples—i.e., the subset with examples which have the attribute's value associated with the branch—and it stops for a given branch when all examples 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 divide-and-conquer approach represents a greedy strategy to create a decision tree, since the selection of an attribute at early iterations cannot be reconsidered at later iterations—i.e., the selection of the best attribute is made locally at each iteration, without taking into consideration its influence over the subsequent iterations.

The problem of inducing a decision tree following the divide-and-conquer approach is divided into smaller problems of selecting an appropriate attribute given a set of examples. Several heuristics for choosing attributes have been used in the literature [23, 24, 33]. The CART algorithm [6] uses the Gini Index as a measure of impurity of an attribute, where the lower the impurity the better is the attribute. Quinlan [30] introduced the ID3 algorithm, which selects attributes based on the information gain measure—a measure derived from the entropy measure commonly used in information theory [7]. ID3 is the precursor of the well-known C4.5 algorithm [31]. More recently, a distance-based measure has been used in the CLUS algorithm [3, 36].

The C4.5 algorithm, probably the best known decision tree induction algorithm, employs an entropy-based criterion in order to select the best attribute to create a node, called the information gain ratio. In essence, the entropy measures the (im)purity of a collection of examples relative to their values of the class attribute, where higher entropy values correspond to more uniformly distributed examples, while lower entropy values correspond to more homogeneous examples (more examples associated with the same class label). The entropy of a collection of examples S is given by

$$Entropy(S) = \sum_{c=1}^{m} -p_c \cdot \log_2 p_c \,, \tag{1}$$

where p_c is the proportion of examples in S associated with the c-th class label and m is the total number of class labels. Using the entropy measure, the information gain of

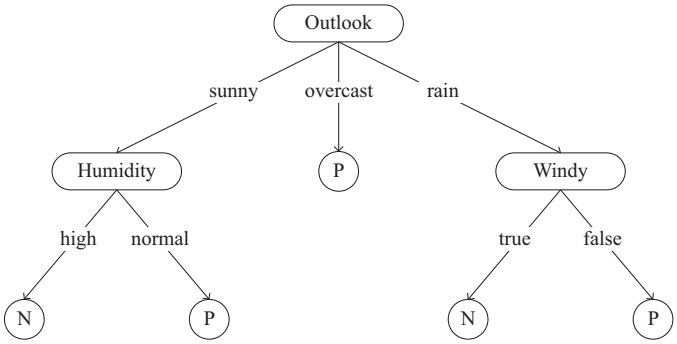

Figure 1: An example of a decision tree, adapted from [30]. Internal nodes (including the root node) are represented by attribute names and branches originating from internal nodes correspond to different values of the attribute in a node; leaf nodes are represented by different class labels. In this example there are three attributes: Outlook {sunny, overcast, rain}, Humidity {high, normal} and Windy {true, false}; and two class labels {N, P}.

an attribute A corresponds to the expected reduction in entropy achieved by dividing the training examples into T subsets, where T is the number of different values in the domain of attribute A, and is defined as

$$Gain(S,A) = Entropy(S) - \sum_{v=1}^{T} \frac{|S_v|}{|S|} \cdot Entropy(S_v)\,, \tag{2}$$

where $|S_v|$ is the number of examples in the subset of S for which the attribute A has the v-th value in the domain of A and $|S|$ is the number of examples in S. The calculation of the information gain ratio includes a penalty for attributes that divide the training examples into very small subsets, called the split information, computed as

$$SplitInformation(S,A) = \sum_{v=1}^{T} -\frac{|S_v|}{|S|} \cdot \log_2 \frac{|S_v|}{|S|}\,. \tag{3}$$

Such a penalty is necessary since attributes that divide the training examples into very small subsets are likely to have a high information gain just because the entropy of each subset is artificially small, given that the very small number of examples in each subset can be easily associated with a single class label without indicating a good generalisation ability. An extreme case would be an attribute that has a different value for each training example. This attribute would have a high information gain, since if we divide the training examples by its values, we would have subsets with examples associated with the same class label, even though the size of each subset is one. Clearly, this attribute represents a poor predictor (no generalisation ability) and would not be useful to classify unseen examples.

Finally, the information gain ratio of an attribute A is derived from the *Gain* and *SplitInformation* measures, and is given by

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitInformation(S,A)}. \quad (4)$$

At each step of the top-down procedure, the C4.5's selection favours the attribute that maximises information gain ratio, which corresponds to the attribute that provides the larger gain in terms of the entropy measure. C4.5 has been successfully applied to a wide range of classification problems and it is usually used on evaluative comparisons of new classification algorithms [37]. Further details of C4.5 can be found in [31, 32].

2.2. Ant Colony Optimization

Ant colonies, despite the lack of centralised control and the relative simplicity of their individuals' behaviours, are self-organised systems which can accomplish complex tasks by having their individual ants interacting with one another and with their environment. The intelligent behaviour of the colony emerges from the indirect communication between the ants mediated by small modifications of the environment, which is called stigmergy.

Many ant species, even with limited visual capabilities or completely blind, are able to find the shortest path between a food source and the nest by using pheromone as a communication mechanism. Ants drop pheromone on the ground as they walk from a food source to the nest, thereby creating a pheromone trail on the used path. The pheromone concentration of a path influences the choices ants make, and the more pheromone the more attractive a path becomes. Given that shorter paths are traversed faster than longer ones, they have a stronger pheromone concentration after a period time, contributing to being selected and reinforced more often. Ultimately the majority of ants will be following the same path, most likely the shortest path between the food source and the nest. Inspired by this behaviour, Dorigo et al. [11, 12, 13] have defined an artificial ant colony metaheuristic that can be applied to solve optimization problems, called Ant Colony Optimization (ACO).

ACO algorithms use a colony of artificial ants, where ants build candidate solutions to optimization problems by iteratively selecting solution components based on their associated pheromone and heuristic information—where the latter corresponds to a measure of how good a solution component is for the problem at hand. The colony cooperates by using pheromone to identify prominent components of a solution and the components with higher concentration of pheromone have a greater chance of being selected by an ant. Components used to create good solutions have their pheromone increased, while components not used will have their pheromone gradually decreased. At the end of the iterative process of building candidate solutions guided by pheromone, the colony converges to optimal or near-optimal solutions. In ACO algorithms, artificial ants construct candidate solutions by traversing a graph, called the construction graph. This is a graph where each vertex represents a potential component of a candidate solution, and the act of traversing an edge means that an ant is adding, to the current candidate solution, the component vertex at the end of the edge. Hence, the problem of finding the best solution corresponds to the problem of finding the best path in the target problem's construction graph. Fig. 2 presents the high-level pseudocode of a basic ACO algorithm. There are four main procedures involved:

Input: problem's construction graph
Output: best solution
1. *Initialise()*;
2. **while** *termination condition not met* **do**
3. *ConstructAntSolutions*();
4. *ApplyLocalSearch*();
5. *UpdatePheromones*();
6. **end while**
7. **return** *best solution*;

Figure 2: High-level pseudocode of a basic ACO algorithm.

- *Initialise*: this procedure sets the parameters of the algorithm, and initialises the amount of pheromone (usually represented as a pheromone matrix) and heuristic information associated with vertices or edges of the construction graph.
- *ConstructAntsSolutions*: this procedure incrementally builds candidate solutions by creating paths on the problem's construction graph, simulating the movement of an artificial ant. Ants traverse the problem's construction graph by applying a stochastic decision policy based on the use of (problem-dependent) heuristic information and pheromone.
- *ApplyLocalSearch*: this (optional) procedure is used to further refine a solution created by an ant. In general, local search operators/algorithms introduce small modifications to a solution in order to explore neighbour solutions. Dorigo and Stützle [13] have shown that for a wide range of optimisation problems, the use of local search operators/algorithms can boost the performance of ACO algorithms. A local search procedure is one example of problem specific or centralised (optional) daemon actions [13]—i.e., actions that cannot be performed by individual ants.
- *UpdatePheromones*: this procedure updates the pheromone associated with the components—vertices or edges—of the problem's construction graph by either increasing the amount of pheromone, as ants deposit pheromone on the path used to build their candidate solutions, or decreasing the amount of pheromone, due to the simulation of pheromone evaporation. The quality of a candidate solution influences on how much pheromone will be deposited on the path that represents the candidate solution.

2.3. Related Work on ACO Algorithms For Inducing Tree Structures

Most of the research using ACO algorithms for the classification task in the context of data mining have been focused on discovering classification rules, as implemented in the Ant-Miner algorithm [28] and its many variations, which have been recently reviewed in [22, 18]. It is important to emphasise that the ACO algorithm proposed in this paper builds a decision tree, rather than a set of rules, and consequently it is very

different from Ant-Miner and its variations. Therefore, in this section we only review related work on ACO for inducing tree-like structures.

Izrailev and Agrafiotis [21] proposed an ant colony-based method for building regression trees. Regression consists of finding a model that maps a given input to a numeric prediction (i.e., the target attribute takes continuous values), while classification consists of finding a model that maps a given example to one of a predefined set of discrete or nominal class labels (i.e., the target attribute takes discrete or nominal values). A regression tree can be viewed as a special case of decision tree, where the value predicted for the target attribute at each leaf node of the tree is a continuous value instead of a discrete or nominal value. In their method, an ant represents a regression tree and the pheromone matrix is represented by a binary reference tree corresponding to the topological union of all created trees; each decision node of a tree is represented by a binary condition $x_i < v_{ij}$ (where v_{ij} is the j-th value of the i-th continuous attribute); pheromone is used to select both the attribute and value to create a decision node. Consequently, their method has two important limitations. First, it uses only continuous attributes on the decision nodes (i.e., it cannot cope with discrete or nominal predictor attributes). Secondly, it does not make use of heuristic information, which is commonly used in ACO algorithms.

Recently, Boryczka and Kozak [4] proposed an ant colony algorithm for building binary decision trees, called ACDT. A candidate decision tree is created by selecting decision nodes—which are represented by binary conditions $x_i = v_{ij}$ (where v_{ij} is the j-th value of the i-th nominal attribute) and consequently have exactly two outgoing edges (i.e., one representing the outcome 'true' when the condition is satisfied and one representing the outcome 'false' when the condition is not satisfied)—according to heuristic information and pheromone values. The heuristic information is based on the Twoing criterion, previously used in the CART decision tree induction algorithm [6], and pheromone values represent the quality of the connection between a parent and child decision nodes. An extension of ACDT, called cACDT, which can cope with continuous attributes is presented in [5]. The cACDT algorithm is included in our experiments to evaluate the performance of the proposed Ant-Tree-Miner algorithm.

While the research in decision tree induction with ACO algorithms remains almost unexplored, evolutionary algorithms (non-ACO) have been applied for decision tree induction [2]. Unlike ACO, evolutionary algorithms do not use a local heuristic and their search is guided only by the fitness function (e.g., global quality of the decision tree). In ACO algorithms, the search is guided by both the overall quality of the solution and a local heuristic information. Additionally, pheromone levels provide a feedback on the quality of the components of the solution and guide the search to the more prominent solutions. More details of evolutionary algorithms for decision tree induction can be found in [2].

3. The Proposed Ant Colony Approach for Decision Tree Induction

The proposed Ant-Tree-Miner algorithm follows the traditional structure of ACO algorithms, as presented in Fig. 3. It starts by initialising the pheromone values and computing the heuristic information for each attribute of the training set. Then, it enters in an iterative loop (*while* loop) where each ant in the colony creates a new decision

Input: training examples, list of predictor attributes
Output: best discovered tree

1. *InitialisePheromones()*;
2. *ComputeHeuristicInformation()*;
3. $tree_{gb} \leftarrow \varnothing$;
4. $m \leftarrow 0$;
5. **while** m < maximum iterations **and** not *CheckConvergence*() **do**
6. $\quad tree_{ib} \leftarrow \varnothing$;
7. $\quad$ **for** n $\leftarrow$ 1 **to** colony_size **do**
8. $\quad\quad tree_n \leftarrow CreateTree(Examples, Attributes, -)$;
9. $\quad\quad Prune(tree_n)$;
10. $\quad\quad$ **if** $Q(tree_n) > Q(tree_{ib})$ **then**
11. $\quad\quad\quad tree_{ib} \leftarrow tree_n$;
12. $\quad\quad$ **end if**
13. $\quad$ **end for**
14. $\quad UpdatePheromones(tree_{ib})$;
15. $\quad$ **if** $Q(tree_{ib}) > Q(tree_{gb})$ **then**
16. $\quad\quad tree_{gb} \leftarrow tree_{ib}$;
17. $\quad$ **end if**
18. $\quad m \leftarrow m + 1$;
19. **end while**
20. **return** $tree_{gb}$;

Figure 3: High-level pseudocode of the Ant-Tree-Miner algorithm.

tree until a maximum number of iterations is reached or the algorithm has converged. An ant creates a decision tree (*for* loop) in a top-down fashion by probabilistically selecting attributes to be added as decision nodes based on the amount of pheromone (τ) and heuristic information (η). The decision tree creation procedure (*CreateTree* procedure) takes three parameters, the set of training examples, the set of predictor attributes and the edge being followed by the ant, which at the start of the procedure corresponds to the default edge (denoted by the symbol '$-$').

Once the tree construction procedure has finished, the created tree is pruned in order to simplify the tree and thus potentially avoid overfitting of the model to the training data. Overfitting is the phenomenon where the model is too adjusted to the training data and as a result does not have a good predictive accuracy on the test set, unseen during training. Since decision nodes are added to the tree while there are attributes available and the set of examples in the current node includes examples of more than on class label, the tree is usually very complex—composed of a large number of nodes—which may affect its generalisation power on unseen data. After the pruning, the tree is evaluated and the iteration-best tree ($tree_{ib}$) is updated, if the quality of the newly created tree is greater than the quality of the current iteration-best tree. Finally, the iteration-best tree constructed by the ants is used to update the pheromone values, the global-best tree ($tree_{gb}$) is stored/updated and a new iteration of the algorithm starts.

When the maximum number of iterations is reached or the algorithm has converged (*CheckConvergence* procedure), the global-best tree is returned as the discovered decision tree.

3.1. Construction Graph

The construction graph consists of N vertices that represent the predictor attributes, one vertex per attribute. These vertices are connected by edges corresponding to different conditions involving values from the domain of the attribute from where the edge originates. Additionally, the construction graph has a virtual '*start*' vertex. An ant always starts the creation of a candidate decision tree from the '*start*' node and there are N edges connecting the '*start*' node to each x_i attribute vertices of the construction graph. Fig. 4 presents an example of a construction graph.

For nominal attributes, each edge represents the condition where the attribute x_i has the value v_{ij}—i.e., the term $x_i = v_{ij}$. Note that an ant cannot select the same nominal attribute vertex x_i multiple times in the same path of the tree—a path in this context is defined as the list of attribute vertices between a node and the root node of the tree—to avoid inconsistencies, such as the selection of the condition '$outlook = sunny$' and '$outlook = rain$'. Hence, there are $N-1$ edges for each $x_i = v_{ij}$ pair connecting attribute vertex x_i to the remaining $N-1$ attribute vertices—i.e., for each value v_{ij} in the domain of the attribute x_i, there is an edge $x_i = v_{ij}$ connecting attribute vertex x_i to the attribute vertex x_k, where $i \neq k$.

Continuous attributes represent a special case of attribute vertices in the construction graph. Given that a continuous attribute does not have a (predefined) set of fixed intervals to define attribute conditions and therefore represent the edges originating from its vertex, a dynamic discretisation procedure is employed to create discrete intervals to be used as attribute conditions. The discretisation procedure follows a similar approach as the one employed in the ACO-based *c*Ant-Miner rule induction algorithm [25, 26]. We have used two different approaches to discretise continuous attributes values:

1. The first approach is the same as the one employed in C4.5, where the threshold value selected corresponds to the value that produces the highest information gain. Each continuous attribute first has its values sorted in increasing order. The candidate threshold values are the values in the domain of the continuous attribute that occur on adjacent examples—according to the sorted values—associated with different class labels. For instance, consider a hypothetical data set with 3 examples with attribute Age values $\{25, 26, 27\}$ associated with class labels $\{yes, yes, no\}$, respectively. A candidate threshold value for those examples is 26, but not 25, since the examples with Age values 25 and 26 have the same class label. A threshold value t divides the training examples into two subsets in relation to the values of the continuous attribute x_i, those with a value of x_i less than the threshold ($x_i < t$) and those with a value greater than or equal to the threshold ($x_i \geq t$) respectively. This approach has also been used in the ACO-based *c*Ant-Miner rule induction algorithm [25].
2. The second approach is based on the minimum description length (MDL) principle, and it was proposed by Fayyad and Irani [14] and used in the context

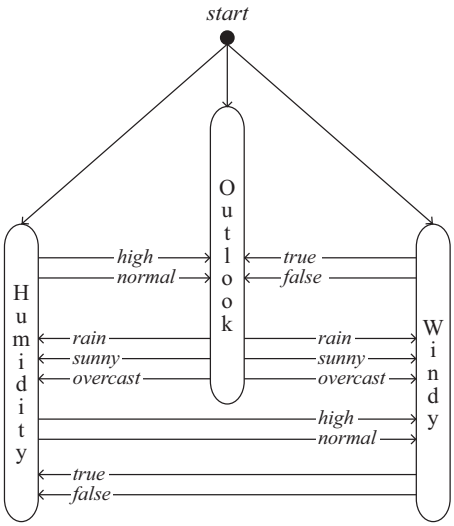

Figure 4: An example of a construction graph that can be used to generate the decision tree presented in Fig. 1. In this example, the construction graph is composed by three vertices representing nominal attributes: Outlook {sunny, overcast, rain}, Humidity {high, normal} and Windy {true, false}; and a virtual '*start*' vertex.

of inducing decision trees. In the MDL-based discretisation, multiple discrete intervals can be extracted by applying a binary discretisation procedure—e.g., the information gain-based discretisation procedure used in C4.5—recursively, selecting the best threshold value at each iteration, and using the MDL principle as a stopping criterion to determine whether more threshold values should be introduced or not. This approach has also been used in an extension of the ACO-based *c*Ant-Miner rule induction algorithm [26].

Note that this makes the construction graph dynamic, since the discretisation of continuous attributes is tailored for the current path being followed by an ant and the current subset of training examples is used in the discretisation procedure. Additionally, a continuous attribute can be selected multiple times in the same path of the decision tree, giving the algorithm a chance to refine the discrete interval selection. For each discrete interval of a continuous attribute x_i generated by the dynamic discretisation procedure, N edges connecting the attribute vertex x_i to every other attribute vertex x_k (including attribute vertex x_i) are added to the construction graph. Since the discretisation procedure is deterministic, when an ant follows the same path to an attribute vertex representing a continuous attribute, the same edges will be used.

3.2. Heuristic Information

The heuristic information associated with each attribute vertex x_i of the construction graph corresponds to its estimated quality with respect to its ability to improve

the predictive accuracy of the decision tree. Ant-Tree-Miner uses the same heuristic information of the well-known C4.5 decision tree induction algorithm, namely the *information gain ratio* of the attributes [31], given by

$$\eta_{x_i} = GainRatio(S, x_i), \tag{5}$$

where x_i corresponds to the i-th attribute vertex and S corresponds to the set of training examples. The *GainRation* function is described in Eq. 4.

In order to calculate the information gain ratio of continuous attributes, it is necessary to dynamically select threshold values to define discrete intervals and subsequently divide the training example into subsets. Once the threshold values of a continuous attribute are generated and its discrete intervals defined, the information gain ratio given by Eq. 4 can be computed assuming that each discrete interval represents a different value and consequently a different subset of training examples. The discretisation of a continuous attribute is a temporary transformation with the goal of computing its information gain ratio and the continuous attribute values are not replaced by the discrete intervals.

Although both Ant-Tree-Miner and C4.5 algorithms use the information gain ratio measure as a heuristic to select attributes, Ant-Tree-Miner selects attributes using the information gain ratio in combination with pheromone, while C4.5 uses only the information gain ratio. The use of pheromone provides an accurate feedback of the quality of an attribute considering its effect in an entire decision tree ('global' attribute evaluation), and so it can compensate for imprecisions of the greedy information gain ratio measure ('local' attribute evaluation).

3.3. Solution Construction

The construction of a candidate decision tree follows a divide-and-conquer approach, with the difference that attributes are chosen stochastically based on heuristic information and pheromone values, instead of deterministically like in a conventional decision tree induction algorithm. At each iteration of the construction process, an ant applies a probabilistic rule to decide which attribute vertex to visit based on the amount of pheromone and the heuristic information. The probability p_i of an ant to visit the attribute vertex x_i is given by

$$p_i = \frac{\tau_{(E,L,x_i)} \cdot \eta_i}{\sum_{i \in \mathcal{F}} \tau_{(E,L,x_i)} \cdot \eta_i}, \ \forall i \in \mathcal{F}, \tag{6}$$

where:

- $\tau_{(E,L,x_i)}$ is the amount of pheromone associated with the entry (E,L,x_i)—E is the attribute condition represented by the edge being followed or '$-$' at the start of the construction procedure, L is the ant's current level in the decision tree or 0 at the start of the construction procedure, x_i is the i-th attribute vertex of the construction graph—in the pheromone matrix;
- η_i is the heuristic information of the i-th attribute;

- $\mathcal{F}$ is the set of available (feasible) attributes for selection.

The exponents α and β commonly used to control the influence of the pheromone and heuristic information, respectively, during the selection of vertices are set to 1 and therefore omitted from Eq. 6.

Fig. 5 presents the high-level pseudocode of the decision tree construction procedure used in Ant-Tree-Miner. An ant begins the construction of a candidate decision tree following the edge '$-$' originating from the virtual '*start*' node, with the complete set of training examples and the complete set of predictor attributes. The selected attribute is used to create a decision node. Depending on the type of the selected attribute, two different set of instructions are executed. If the selected attribute is a nominal attribute, it is removed from the set of predictor attributes and the list of conditions is filled with attribute conditions $A = v_i$ for all values v_i in the domain of attribute A. If the selected attribute is a continuous attribute, the list of conditions is represented by the discrete intervals generated by the discretisation procedure. The discretisation of a continuous attribute is tailored to the current set of training examples, as explained earlier. In contrast to nominal attributes, continuous attributes are not removed from the set of predictor attributes since they can be selected multiple times, in the same tree path, by an ant.

Once the list of attribute conditions is defined, each condition is used to create a branch of the decision node. Then, the set of training examples is divided into one subset of examples for each attribute condition (branch), where each subset contains the training examples satisfying the corresponding attribute condition. At this point, the construction procedure checks whether a leaf node should be added below the current branch or if it should recursively add a subtree below the current branch. The decision to add a leaf node to the candidate decision tree is deterministic and it is based on the following conditions:

1. the current subset of training examples is empty, which corresponds to the case that none of the training examples satisfied the attribute condition represented by the current branch;
2. all examples in the current subset are associated with the same class label;
3. the number of examples in the subset is below a user-defined threshold;
4. the set of available predictor attributes is empty.

If any of the above conditions is observed, a leaf node representing a class label prediction is added below the current branch. Otherwise, the construction procedure is applied recursively to create a subtree given the subset of training examples, the current set of predictor attributes and the current branch. Finally, the root node of the candidate decision tree is returned at the end of the construction procedure.

3.4. Pruning

After a candidate decision tree is created, it undergoes a pruning procedure. The aim of the pruning procedure is to improve the decision tree's generalisation power and consequently its predictive accuracy by removing unnecessary decision nodes from the tree. In general, a candidate tree created by the the construction procedure is overly

Input: training examples (*Examples*), list of predictor attributes (*Attributes*), current edge (*Edge*)
Output: root node of the decision tree

1. $A \leftarrow$ probabilistically selects an attribute from *Attributes* to visit given the current *Edge*;
2. *root* $\leftarrow$ creates a new decision node representing attribute A;
3. *conditions* $\leftarrow \varnothing$;
4. **if** A is a nominal attribute **then**
5. $\quad$ *Attributes* $\leftarrow$ *Attributes* $- \{A\}$;
6. $\quad$ **for all** value v_i in domain of A **do**
7. $\quad\quad$ *conditions* $\leftarrow$ *conditions* $+ \{A = v_i\}$;
8. $\quad$ **end for**
9. **else**
10. $\quad$ *conditions* $\leftarrow$ *Discretise(A, Examples)*;
11. **end if**
12. **for all** attribute condition T in conditions **do**
13. $\quad$ $branch_i \leftarrow$ new branch representing T of *root*;
14. $\quad$ $subset_i \leftarrow$ subset of *Examples* that satisfies T;
15. $\quad$ **if** $subset_i$ is empty **then**
16. $\quad\quad$ Add a leaf node with the majority class label of *Examples* below $branch_i$;
17. $\quad$ **else if** all examples in $subset_i$ have the same class label **then**
18. $\quad\quad$ Add a leaf node with the class label of $subset_i$ below $branch_i$;
19. $\quad$ **else if** number of examples in $subset_i$ is below a threshold **then**
20. $\quad\quad$ Add a leaf node with the majority class label of $subset_i$ below $branch_i$;
21. $\quad$ **else if** Attributes is empty **then**
22. $\quad\quad$ Add a leaf node with the majority class label of $subset_i$ below $branch_i$;
23. $\quad$ **else**
24. $\quad\quad$ Add the subtree returned by *CreateTree(*$subset_i$*, Attributes,* $branch_i$*)* below $branch_i$;
25. $\quad$ **end if**
26. **end for**
27. **return** *root*;

Figure 5: High-level pseudocode of the *CreateTree(Examples, Attributes, Edge)* decision tree construction procedure used in Ant-Tree-Miner.

complex, since it is expanded until there are no available attributes or training examples, or all training examples are associated with the same class label. The main drawback of having a complex tree is that the tree is likely to overfit the training examples and its predictive accuracy will be poor on unseen examples. Additionally, the decision tree creation procedure is stochastic and the generated tree usually does not represent the best fit to the training data. This is the main difference between an unpruned tree generated by C4.5 and Ant-Tree-Miner. An unpruned decision tree of C4.5 represents the best fit to the training data using the deterministic (greedy) creation procedure based on the information gain ratio. Therefore, the pruning procedure should be applied in order to increase the generalisation power of the decision tree. In Ant-Tree-Miner, the pruning procedure has two tasks and it is divided into two steps: the first step prunes the tree to fit the training data, which potentially leads to overfitting the training data, and the second step prunes the tree in order to increase its generalisation power.

The first step consists of replacing a decision node by either its most common used branch or by a leaf node that leads to a higher classification accuracy on the training data. The aim of this step is to remove decision nodes that have been added by the stochastic creation procedure that have a negative effect on the classification accuracy of the decision tree. This step is repeated until none of the replacements leads to an improvement in accuracy or the tree consists of just a leaf node.

The second step consists of the same pruning procedure used in C4.5, called *error-based pruning* [31]. In essence, it consists of replacing a decision node by either its most common used branch or by a leaf node. If the replacement leads to a lower estimated error rate (details of the error rate estimation can be found in [31, p. 41]), the decision tree is pruned accordingly. By contrast with the previous step, this decision is based on the estimated error rate (a measure that gives a higher penalty for errors than the accuracy measure) rather than the classification accuracy in training examples, since the use of the classification accuracy can lead to overfitting—the case where the decision tree is too tailored to the training data and does not generalise well, i.e., it has a lower predictive accuracy in the test data. The rationale behind this replacement is that the overall (estimated) error rate of the tree decreases if the error rate of any subtree is reduced. The aim of this step is to improve the generalisation power of the tree by reducing its estimated error rate on the training data. Note that a lower estimated error rate is not related to a higher classification accuracy on the training data, and a decision node that leads to a lower estimated error rate will be replaced by this step even though the accuracy on the training data decreases—the accuracy on the training data is not computed by this step.

3.5. Pheromone Matrix Representation

In order for an ant to create a decision tree, it has to follow several paths from the root node towards leaf nodes. The decision nodes of a tree are represented by attribute vertices of the construction graph and each branch originating from a decision node is represented by an edge of the construction graph. Recall that each edge of the construction graph is represented by an attribute condition and it has a destination vertex that is the vertex that the edge leads to. Since the choices made by an ant can be expressed by the edges that it has followed during the creation of a decision tree, each entry in the pheromone matrix is represented by a triple $[edge_{ij}, level, x_k]$, where $edge_{ij}$

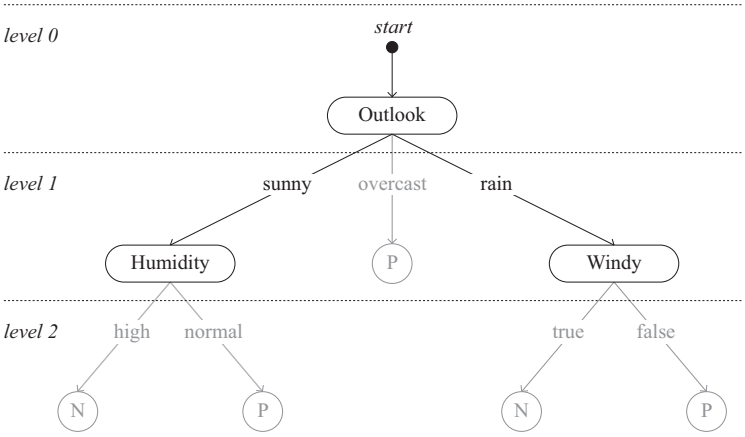

Figure 6: An example of a decision tree created by an ant; the '*start*' node represents the starting vertex. The highlighted branches—the branches that lead to a decision node, including the root node—are the ones that eventually will be used to update the pheromones. Branches whose terminal node is a leaf node, such as branches at level 2 in the figure, are the result of the deterministic pruning procedure or the case when all examples following the branch are associated with the same class label, therefore they are not represented in the pheromone matrix.

is the edge representing the j-th attribute condition of the attribute x_i, *level* is the level of the decision tree where the $edge_{ij}$ 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. Given that edges for continuous attributes' vertices are dynamically created, the pheromone matrix is an incremental structure that represent the mapping of a triple [$edge_{ij}$, *level*, x_k] (*key* of the mapping) and a pheromone value (*value* of the mapping).

Revisiting the example of a decision tree presented in Fig. 1, Fig. 6 illustrates the correspondent candidate decision tree created by an ant. In this example, the highlighted branches—the branches that lead to a decision node, including the root node—are the ones that eventually will be used to update the pheromones and they are represented by the entries:

- `[−,0,Outlook]` - This is the entry in the pheromone matrix for the edge connecting the virtual '*start*' node to the '*Outlook*' node, the root node of the tree. The '*start*' node is referred to as virtual since it is not included in the final decision tree. It is used to associate pheromone on the edge that leads to the root node of a candidate decision tree. The edges originating at the '*start*' node do not represent an attribute condition. They are referred to as $edge_{start}$ and represented by the symbol '$-$';

- `[Outlook = sunny,1,Humidity]` - This is the entry in the pheromone matrix for the edge located at level 1 representing the condition that the '*Outlook*'

attribute has the value equal to '*sunny*', which connects the '*Outlook*' attribute vertex to the '*Humidity*' attribute vertex;

- `[Outlook = rain,1,Windy]` - This is the entry in the pheromone matrix for the edge located at level 1 representing the condition that the '*Outlook*' attribute has the value equal to '*rain*', which connects the '*Outlook*' attribute vertex to the '*Windy*' attribute vertex.

The remaining edges in the example on Fig. 6 are not used during the pheromone update nor represented in the pheromone matrix, since they directly lead to a leaf node representing a class label prediction and they are deterministically introduced in the decision tree, either during the tree creation process or by the pruning procedure.

3.6. Pheromone Update

The pheromone values update in Ant-Tree-Miner are governed by the same approach as the $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [34, 35]. In $\mathcal{MM}$AS, pheromone trail values are limited to the interval $[\tau_{min}, \tau_{max}]$. These limits are dynamically updated each time a new global-best solution is found, as detailed in [34].[1] They are also used to determine the stagnation of the search. When all entries in the pheromone matrix used by an ant to create the iteration-best tree are associated with τ_{max} and the remaining entries are associated with τ_{min}, the search is considered stagnant and the algorithm stops.

Given the best candidate decision tree of an iteration (the iteration-best candidate solution), pheromone update is performed in two steps. Firstly, pheromone evaporation is accomplished by decreasing the amount of pheromone of each entry in the pheromone matrix by a factor ρ (a user-defined parameter). Secondly, the amount of pheromone of the entries corresponding to the branches used in the iteration-best candidate solution are increased based on the quality of the candidate solution. The quality of a decision tree is based on the same measure used during the pruning procedure and is given by

$$Q = \frac{N - Error}{N}, \tag{7}$$

where N is the number of training examples and *Error* is the estimated number of classification errors of the tree—the lower the number of errors, the better the quality of the tree. The quality of a tree given by Eq. 7 is bound to the interval $[0, 1]$. Finally, the pheromone update rule is given by

$$\tau_{(E,L,x_i)} = \begin{cases} \rho \cdot \tau_{(E,L,x_i)}, & \text{if } (E,L,x_i) \notin tree_{ib}; \\ \rho \cdot \tau_{(E,L,x_i)} + Q(tree_{ib}), & \text{if } (E,L,x_i) \in tree_{ib}; \end{cases} \tag{8}$$

where ρ is the $\mathcal{MAX}-\mathcal{MIN}$ evaporation factor, $\tau_{(E,L,x_i)}$ is the amount of pheromone associated with the entry (E,L,x_i)—E is the attribute condition of the edge that this

[1] In Ant-Tree-Miner, the pheromone trail limits are chosen using the default $\mathcal{MM}$AS $p_{best} = 0.05$ parameter.

entry corresponds to, L is the level in which the edge occurs and x_i is the edge's destination attribute vertex—and $tree_{ib}$ is the iteration-best decision tree.

4. Computational Results

The proposed Ant-Tree-Miner algorithm was compared against two well-known decision tree induction algorithms implemented in the Weka workbench [37], namely C4.5 [31] (Weka's J48 algorithm) and CART [6] (Weka's SimpleCART algorithm), and against the ACO-based *c*ACDT decision tree algorithm [5]. We have tested four variations of the Ant-Tree-Miner algorithm:

1. using a binary entropy-based discretisation procedure and the two-step tree pruning, denoted by Ant-Tree-Miner;
2. using a binary entropy-based discretisation procedure and only C4.5's error-based pruning procedure, denoted by Ant-Tree-Miner^{-P} (where '$-P$' denotes the algorithm without the accuracy-based pruning step);
3. using a MDL-based discretisation procedure and the two-step tree pruning, denoted by Ant-Tree-Miner$_{MDL}$;
4. using a MDL-based discretisation procedure and only C4.5's error-based pruning procedure, denoted by Ant-Tree-Miner$^{-P}_{MDL}$.

The motivation for testing these different variations is to evaluate both discretisation and pruning strategies. In terms of discretisation, we are evaluating whether it is best to use a binary discretisation (i.e., a discretisation that creates a single binary split) or the multiple interval MDL-based discretisation. In terms of pruning, we are evaluating whether it is best to first adjust a candidate tree to best fit the training data (two-step pruning) or use the simplified (one-step) pruning procedure—only C4.5's error-based pruning procedure.

A comparison between Ant-Tree-Miner and *c*ACDT is interesting, since Ant-Tree-Miner follows a different strategy to create a decision tree. First, *c*ACDT only creates binary trees (i.e., trees with decision nodes with two outgoing edges), while Ant-Tree-Miner is not restricted to create binary trees (i.e., for decision nodes representing nominal attributes, there is a branch for each value in the domain of the attribute). Second, Ant-Tree-Miner employs a 3-dimensional pheromone matrix to differentiate multiple occurrences of edges (connections between attributes) of a decision tree. Third, *c*ACDT uses the Twoing criterion (based on the CART algorithm) as heuristic information and to dynamically discretise continuous attributes, while Ant-Tree-Miner uses entropy-based criterion (based on the C4.5 algorithm) as heuristic information and to dynamically discretise continuous attributes.

4.1. Experimental Setup

The experiments were carried out using 22 publicly available data sets from the UCI Machine Learning repository [16]. A summary of the data sets used in the experiments is presented in Table 1. We performed tenfold cross-validation for each data set. A tenfold cross-validation procedure consists of dividing the data set into ten stratified partitions of examples, wherein each partition has a similar number of examples

Table 1: Summary of the data sets used in the experiments.

Abbr.	Description	Attributes		Classes	Size
		Nominal	Continuous		
auto	automobile	10	15	7	205
balance	balance scale	4	0	3	625
blood-t	blood transfusion	0	4	2	748
breast-l	breast cancer ljubljana	9	0	2	286
breast-t	breast tissue	0	9	6	106
breast-w	breast cancer wisconsin	0	30	2	569
credit-a	credit approval	8	6	2	690
derm	dermatology	33	1	6	366
ecoli	ecoli	0	7	8	336
glass	glass	0	9	7	214
heart-c	heart cleveland	6	7	5	303
heart-h	heart hungarian	6	7	5	294
horse	horse colic	15	7	2	368
hep	hepatitis	13	6	2	155
ionos	ionosphere	0	34	2	351
iris	iris	0	4	3	150
park	parkinsons	0	22	2	195
s-heart	statlog heart	7	6	2	270
soybean	soybean	35	0	19	307
voting	voting records	16	0	2	435
wine	wine	0	13	3	178
zoo	zoo	16	0	7	101

and class distribution. For each partition, the classification algorithm is run using the remaining nine partitions as the training set and its performance is evaluated using the unseen (hold-out) partition. For the stochastic Ant-Tree-Miner algorithm and its variations, the algorithm is run fifteen times using a different random seed to initialise the search for each partition of the cross-validation. In the case of deterministic C4.5 and CART algorithms, the algorithm is run just once for each partition of the cross-validation.

The user-defined parameters of Ant-Tree-Miner variations were selected from values commonly used in the literature. Although we have made no attempt to tune the parameter values for individual data sets, we have performed a systematic parameter tuning in order to determine a suitable combination of values that work well across a set of tuning data sets.[2] We have tested three different values for the parameters *colony size* $= \{50, 100, 200\}$, each of them with three different values of $\mathcal{MAX}-\mathcal{MIN}$ *evaporation factor* $= \{0.85, 0.90, 0.95\}$—i.e., nine combinations in total. We have left

[2]The tuning data sets used in this step comprises 8 data sets from the UCI repository not used in our final experiments, namely: cylinder-bands, lymphography, monk-1, pima-indians-diabetes, thyroid, tic-tac-toe, vertebral-column-2c and vertebral-column-3c.

fixed the *maximum iterations* = 500, given that we observed that the algorithm converges in less than 200 iterations on average, and the minimum number of examples per branch *threshold* = 3. The parameter tuning did not show significant differences between the nine possible combinations of values—i.e., the findings concerning statistically significant differences between the algorithms was approximately the same across all nine combinations of *colony size* and *evaporation factor*. The combination of *colony size* = 50 and *evaporation factor* = 0.90 was used in the experiments reported in this section, since it performed slightly better than the other combinations, overall, in the tuning data sets. The C4.5, CART and *c*ACDT algorithms were used with their default parameters.

The results concerning the predictive accuracy are summarised in Table 2 and the results concerning the size of the classification model, measured as the average number of leaf nodes in the discovered decision tree, are summarized in Table 3. Each value in those tables represents the average value obtained by the cross-validation procedure followed by the standard error (*average* $\pm$ *standard error*) for the corresponding algorithm and data set pair. Table 4 shows the results of the statistical tests according to the non-parametric Friedman test with the Hommel's post-hoc test [9, 20], for predictive accuracy and discovered model size. For each algorithm, the table shows its average rank—the lower the average rank the better the algorithm's performance—in the first column and the adjusted p-value of the statistical test when that algorithm's average rank is compared to the average rank of the algorithm with the best rank (control algorithm) according to the Hommel's post-hoc test. When statistically significant differences between the average ranks of an algorithm and the control algorithm at the 5% level ($p \leq 0.05$) are observed, the line is tabulated in bold face.

4.2. Discussion

Considering the predictive accuracy (Table 2), Ant-Tree-Miner achieves the highest performance with an average rank of 2.52 across all data sets, which is statistically significantly better than the average ranks obtained by C4.5, Ant-Tree-Miner$_{MDL}$, CART and *c*ACDT algorithms according to the non-parametric Friedman test with the Hommel's post-hoc test. The Ant-Tree-Miner^{-P} and Ant-Tree-Miner$^{-P}_{MDL}$ variations achieved an average rank of 3.20 and 3.93, respectively, ant there are no statistically significant differences between their average ranks and the average rank of Ant-Tree-Miner. C4.5, Ant-Tree-Miner$_{MDL}$ and CART have similar performances, with an average rank of 4.04, 4.11 and 4.49, respectively. *c*ACDT has the worst performance, with an average rank of 5.68. Ant-Tree-Miner is also the most accurate algorithm in 8 of the 22 data sets, followed by C4.5 in 5 data sets, *c*ACDT in 3 data sets, Ant-Tree-Miner$^{-P}_{MDL}$ and CART in 2 data sets each, and both Ant-Tree-Miner^{-P} and Ant-Tree-Miner$_{MDL}$ in 1 data set each.

In terms of size of the classification model (Table 3), measured as the number of leaf nodes of the discovered decision tree, CART is the algorithm that discovers the decision tree with the lowest number of leaves, obtaining an average rank of 1.39 across all data sets. The average rank of CART is statistically significant better than the average ranks obtained by all the other six algorithms according to the non-parametric Friedman test with the Hommel's post-hoc test. Ant-Tree-Miner is second with an average rank of 3.68; *c*ACDT is third with an average rank of 4.00; Ant-Tree-Miner^{-P} is fourth with an

Table 2: Average predictive accuracy (*average* $\pm$ *standard error*) in %, measured by tenfold cross-validation. The columns denoted by ATM correspond to Ant-Tree-Miner variations. The value of the most accurate algorithm for each data set is shown in bold.

Data Set	C4.5	CART	*c*ACDT	ATM	ATM^{-P}	ATM$_{MDL}$	ATM$^{-P}_{MDL}$
auto	**81.4 ± 2.5**	76.6 ± 2.6	66.0 ± 0.6	77.2 ± 0.4	77.4 ± 0.6	74.8 ± 0.5	76.2 ± 0.5
balance	63.7 ± 2.2	79.3 ± 1.0	**79.4 ± 0.2**	62.9 ± 0.1	62.9 ± 0.1	62.9 ± 0.1	62.9 ± 0.1
blood-t	73.8 ± 1.3	**74.2 ± 1.2**	71.7 ± 0.0	72.4 ± 0.1	72.3 ± 0.1	72.1 ± 0.1	72.4 ± 0.1
breast-l	72.9 ± 2.3	71.7 ± 1.5	71.5 ± 0.4	73.5 ± 0.1	73.5 ± 0.1	**73.6 ± 0.1**	73.2 ± 0.1
breast-t	59.6 ± 5.7	64.7 ± 5.7	47.8 ± 0.8	**65.2 ± 0.6**	64.5 ± 0.6	62.9 ± 0.5	62.1 ± 0.4
breast-w	**94.9 ± 0.7**	93.8 ± 0.8	93.2 ± 0.2	94.0 ± 0.2	94.1 ± 0.2	93.5 ± 0.1	93.8 ± 0.1
credit-a	85.8 ± 1.0	85.2 ± 1.4	85.2 ± 0.2	85.9 ± 0.1	85.8 ± 0.1	85.9 ± 0.1	**86.0 ± 0.1**
derma	93.5 ± 1.1	94.0 ± 1.4	**95.7 ± 0.2**	94.4 ± 0.1	94.5 ± 0.1	94.4 ± 0.1	94.6 ± 0.1
ecoli	83.7 ± 1.7	81.3 ± 1.9	81.2 ± 0.3	**83.9 ± 0.1**	83.1 ± 0.2	83.2 ± 0.2	83.4 ± 0.2
glass	68.5 ± 1.8	68.4 ± 2.5	65.1 ± 0.5	71.0 ± 0.4	71.2 ± 0.5	77.1 ± 0.4	**77.2 ± 0.4**
heart-c	51.2 ± 1.6	**55.1 ± 1.4**	54.9 ± 0.5	51.9 ± 0.4	52.3 ± 0.4	51.5 ± 0.4	51.8 ± 0.3
heart-h	**66.7 ± 3.0**	62.7 ± 2.1	62.3 ± 0.4	65.9 ± 0.2	66.2 ± 0.2	65.4 ± 0.3	65.6 ± 0.2
hep	80.7 ± 2.5	77.5 ± 2.1	80.4 ± 0.7	**82.5 ± 0.3**	82.2 ± 0.3	80.6 ± 0.2	81.1 ± 0.2
horse	**84.7 ± 1.5**	83.5 ± 1.6	83.1 ± 0.3	83.8 ± 0.1	83.6 ± 0.2	83.0 ± 0.2	83.1 ± 0.2
ionos	90.2 ± 1.2	89.4 ± 1.8	88.0 ± 0.4	**90.8 ± 0.2**	90.7 ± 0.1	89.0 ± 0.4	88.9 ± 0.2
iris	93.9 ± 1.6	93.8 ± 1.3	94.9 ± 0.4	**96.2 ± 0.1**	95.9 ± 0.1	95.1 ± 0.1	94.9 ± 0.1
park	88.2 ± 1.5	86.7 ± 2.2	86.2 ± 0.5	**92.4 ± 0.1**	92.2 ± 0.1	89.2 ± 0.3	89.4 ± 0.2
s-heart	75.6 ± 3.1	78.1 ± 2.3	**79.5 ± 0.3**	77.3 ± 0.3	76.9 ± 0.3	77.3 ± 0.2	77.1 ± 0.2
soybean	85.6 ± 1.7	87.6 ± 1.9	86.4 ± 0.3	**87.4 ± 0.2**	87.3 ± 0.2	87.4 ± 0.3	87.2 ± 0.2
voting	94.5 ± 1.2	93.3 ± 1.2	93.1 ± 0.2	**94.9 ± 0.1**	94.7 ± 0.1	94.8 ± 0.1	94.8 ± 0.1
wine	93.3 ± 2.4	91.0 ± 1.5	89.2 ± 0.5	96.2 ± 0.1	**96.4 ± 0.1**	94.3 ± 0.3	94.9 ± 0.2
zoo	**89.6 ± 5.2**	87.6 ± 5.9	82.4 ± 0.7	88.2 ± 0.2	88.6 ± 0.3	88.8 ± 0.4	88.8 ± 0.4

average rank of 4.27; Ant-Tree-Miner$^{-P}_{MDL}$ and Ant-Tree-Miner$_{MDL}$ have similar results, with average ranks 4.70 and 4.75, respectively; C4.5 is the algorithm that consistently discover trees with a greater number of leaf nodes (in 14 out of the 22 data sets) and performs last, with an average rank of 5.20.

Our results show that CART is consistently the algorithm that discovers decision trees with the smallest number of leaf nodes. This is likely due to the way that CART handles nominal attributes. While C4.5 and Ant-Tree-Miner create a branch for each value of a nominal attribute (i.e., each branch representing a condition $x_i = v_{ij}$), CART can group different values into the same branch (i.e., each branch can represent a condition x_i *in* $\{v_{i1}, v_{i2}, \ldots\}$). This effectively reduces the dimensionality of nominal attributes with large number of values and allows the algorithm to group the values into a small number of subsets, and consequently, reduces the complexity of the tree and the number of leaf nodes. On the other hand, it has a very significant negative impact on the predictive accuracy of the discovered decision trees, since CART achieves the second worst rank in terms of predictive accuracy.

Overall, the results obtained by the Ant-Tree-Miner algorithm are clearly positive. It achieved the best rank in terms of predictive accuracy and outperformed well-known

Table 3: Average number of leaf nodes (*average* ± *standard error*) of the decision trees, measured by tenfold cross-validation. The columns denoted by ATM correspond to Ant-Tree-Miner variations. The value of the algorithm with the lowest average number of leaf nodes for each data set is shown in bold.

Data Set	C4.5	CART	*c*ACDT	ATM	ATM^{-P}	ATM$_{MDL}$	ATM$^{-P}_{MDL}$
auto	46.6 ± 2.3	23.3 ± 0.9	**12.4 ± 0.5**	31.1 ± 0.6	29.5 ± 0.6	27.3 ± 0.3	27.7 ± 0.3
balance	34.6 ± 1.2	**25.4 ± 5.2**	47.4 ± 0.4	34.5 ± 0.1	34.6 ± 0.0	34.5 ± 0.1	34.5 ± 0.1
blood-t	6.6 ± 0.7	**6.4 ± 0.7**	8.0 ± 0.3	14.8 ± 0.1	14.8 ± 0.1	14.9 ± 0.1	14.6 ± 0.1
breast-l	8.2 ± 2.6	**2.9 ± 0.4**	10.7 ± 0.2	10.0 ± 0.2	10.3 ± 0.2	10.5 ± 0.3	10.5 ± 0.2
breast-t	12.9 ± 0.5	**7.8 ± 0.8**	15.3 ± 0.4	12.0 ± 0.1	11.9 ± 0.1	13.6 ± 0.1	13.7 ± 0.1
breast-w	11.5 ± 0.5	**7.1 ± 0.5**	10.9 ± 0.2	9.0 ± 0.1	9.0 ± 0.1	11.1 ± 0.1	11.1 ± 0.1
credit-a	19.8 ± 2.2	**2.8 ± 0.4**	14.7 ± 0.3	29.6 ± 0.2	29.8 ± 0.2	29.8 ± 0.3	29.6 ± 0.3
derma	28.0 ± 1.3	**7.5 ± 0.2**	14.6 ± 0.2	20.7 ± 0.1	21.1 ± 0.1	20.9 ± 0.1	20.9 ± 0.1
ecoli	17.9 ± 0.9	**10.1 ± 1.2**	13.9 ± 0.3	15.9 ± 0.1	16.0 ± 0.1	16.8 ± 0.1	16.5 ± 0.1
glass	23.8 ± 0.5	14.7 ± 2.1	**5.6 ± 0.2**	20.2 ± 0.1	20.2 ± 0.1	19.5 ± 0.1	19.4 ± 0.1
heart-c	44.7 ± 1.6	8.8 ± 2.3	**8.4 ± 0.3**	35.8 ± 0.2	36.4 ± 0.3	35.8 ± 0.2	36.4 ± 0.3
heart-h	27.6 ± 0.9	**3.1 ± 0.7**	4.8 ± 0.2	21.8 ± 0.1	21.9 ± 0.1	22.1 ± 0.2	22.0 ± 0.1
hep	9.7 ± 0.6	**2.9 ± 0.8**	10.2 ± 0.3	7.6 ± 0.1	7.5 ± 0.1	8.1 ± 0.1	8.1 ± 0.1
horse	5.2 ± 0.5	**4.0 ± 0.4**	19.7 ± 0.3	10.2 ± 0.1	10.0 ± 0.1	12.7 ± 0.2	13.0 ± 0.2
ionos	13.4 ± 0.7	**8.6 ± 1.6**	8.9 ± 0.2	12.4 ± 0.1	12.2 ± 0.1	14.3 ± 0.1	14.4 ± 0.1
iris	4.5 ± 0.2	4.4 ± 0.3	4.8 ± 0.1	**4.2 ± 0.0**	4.4 ± 0.0	4.4 ± 0.0	4.3 ± 0.0
park	10.7 ± 0.4	**6.2 ± 0.7**	7.9 ± 0.2	8.0 ± 0.0	8.1 ± 0.0	8.3 ± 0.1	8.4 ± 0.1
s-heart	20.8 ± 1.6	**10.1 ± 1.5**	27.4 ± 0.3	18.1 ± 0.2	18.0 ± 0.2	18.4 ± 0.2	18.0 ± 0.2
soybean	55.9 ± 1.6	35.4 ± 2.0	**2.4 ± 0.2**	50.0 ± 0.3	50.3 ± 0.3	50.0 ± 0.3	50.3 ± 0.2
voting	5.7 ± 0.1	**4.8 ± 0.7**	10.8 ± 0.2	6.1 ± 0.0	6.1 ± 0.0	6.1 ± 0.0	6.0 ± 0.0
wine	**5.2 ± 0.1**	5.5 ± 0.3	7.5 ± 0.1	5.6 ± 0.0	5.6 ± 0.0	7.1 ± 0.0	7.1 ± 0.0
zoo	11.2 ± 0.8	**7.3 ± 0.1**	7.4 ± 0.0	10.7 ± 0.0	10.7 ± 0.1	10.5 ± 0.0	10.8 ± 0.1

C4.5 and CART algorithms with statistically significant differences, and also the ACO-based *c*ACDT algorithm. While Ant-Tree-Miner is not the algorithm that discovers decision trees with the lowest number of leaf nodes and it is outperformed by CART with a statistically significant difference, Ant-Tree-Miner achieves the second best rank in terms of size of the classification model; although CART achieves the best ranking in size of the classification model, it achieves the second worst rank in predictive accuracy. Therefore, Ant-Tree-Miner presents a better trade-off between both accuracy and model size, overall, than C4.5, CART—two very popular conventional decision tree induction algorithms—and the ACO-based *c*ACDT algorithm.

Comparing the performance of Ant-Tree-Miner variations we can draw the following conclusions about the discretisation and pruning strategies. The binary (single interval) discretisation performs best in terms of both predictive accuracy and size of the classification model, given that Ant-Tree-Miner and Ant-Tree-Miner^{-P} achieved better average ranks than the corresponding Ant-Tree-Miner$_{MDL}$ and Ant-Tree-Miner$^{-P}_{MDL}$ variations using the MDL-based (multi-interval) discretisation. The difference between the pruning strategies is not as clear as between the discretisation strategies. While the use of the two-step pruner helps the Ant-Tree-Miner with binary discretisation to

Table 4: Statistical test results according to the non-parametric Friedman test with the Holm's post-hoc test for $\alpha = 0.05$. Statistically significant differences between the average ranks of an algorithm and the control algorithm are shown in bold.

Algorithm	average rank	adjusted p_{Homm}
(i) Predictive Accuracy		
Ant-Tree-Miner (control)	2.52	–
Ant-Tree-Miner^{-P}	3.20	0.2952
Ant-Tree-Miner$^{-P}_{MDL}$	3.93	0.0610
C4.5	**4.04**	**0.0458**
Ant-Tree-Miner$_{MDL}$	**4.11**	**0.0437**
CART	**4.49**	**0.0120**
*c*ACDT	**5.68**	**7.4E-6**
(ii) Model Size		
CART (control)	1.39	–
Ant-Tree-Miner	**3.68**	**4.2E-4**
*c*ACDT	**4.00**	**1.2E-4**
Ant-Tree-Miner^{-P}	**4.27**	**2.8E-5**
Ant-Tree-Miner$^{-P}_{MDL}$	**4.70**	**1.4E-6**
Ant-Tree-Miner$_{MDL}$	**4.75**	**9.7E-7**
C4.5	**5.20**	**2.7E-8**

achieve a higher predictive accuracy and smaller number of leaf nodes in the discovered decision trees than the Ant-Tree-Miner^{-P} variation, the same effect is not observed in the MDL-based discretisation variations. The Ant-Tree-Miner$^{-P}_{MDL}$—which uses the one-step pruner (C4.5's error-based pruning procedure)—achieved a better average rank in terms of both predictive accuracy and size of the classification model, although the differences are relatively small. This is likely due to the differences in the structure of the trees when using the multiple interval discretisation, where in this case both pruning strategies have a similar effect.

Considering the average computation time for inducing a decision tree, shown in Fig. 7, it is no surprise that the Ant-Tree-Miner variations using the two-step pruning procedure require more time than the ones using only C4.5's error-based pruning. While the Ant-Tree-Miner algorithm achieved the highest predictive accuracy overall, it is a factor of 4.2 slower than the Ant-Tree-Miner$^{-P}_{MDL}$ (the fastest Ant-Tree-Miner variation). The Ant-Tree-Miner^{-P} presents a trade-off between accuracy, model size and computational time: while its predictive accuracy is slightly lower than the Ant-Tree-Miner variation, it discovers simpler decision trees requiring less computational time. The MDL-based discretisation helps to decrease the computational time at the cost of a lower predictive accuracy and decision trees with more leaf nodes.

When compared to the computational time for inducing a decision tree of C4.5 and CART, Ant-Tree-Miner variations are a factor of 120 to 504 slower. This is expected, given that both C4.5 and CART create and prune a single candidate decision tree using a deterministic procedure, while in Ant-Tree-Miner variations multiple candidate

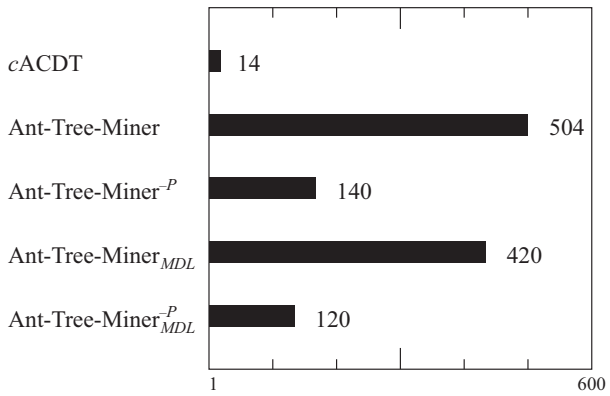

Figure 7: Average computation time (over all data sets) in seconds to complete a fold of the cross-validation for each of the ACO-based algorithms on a 3.33 GHz Intel Xeon computer. The deterministic C4.5 and CART algorithms take on average 1 second to complete a fold of the cross-validation.

decision trees are created and pruned before finding the best decision tree. Comparing the time taken by *c*ACDT, Ant-Tree-Miner variations are a factor of 8.5 to 36 slower, while the predictive accuracy of *c*ACDT is statistically significantly lower than Ant-Tree-Miner's accuracy. It should be noted that in many data mining applications (e.g., medical diagnosis, bioinformatics and credit scoring) the computational time taken by the algorithm to induce a classification model has a relatively minor importance, since they represent off-line applications and the time spent collecting and preparing the data is usually much greater than the time required to run the classification algorithm—although in applications that require parameter tuning, the number of parameter settings evaluated could be limited by the computational time of the algorithm. In addition, ACO algorithms can be easily parallelised since each ant builds and evaluates a candidate decision tree independent from all the other ants. Therefore, a large speed up could be obtained by running a parallel version of Ant-Tree-Miner on a computer cluster or another parallel computing system in applications where Ant-Tree-Miner's processing time becomes a significant issue.

4.3. Comparing the Ant-Tree-Miner and cAnt-Miner$_{\mathrm{PB}}$ *rule induction algorithms*

In this section we compare the performance of Ant-Tree-Miner against the ACO-based *c*Ant-Miner$_{\mathrm{PB}}$ rule induction algorithm [27]. Such a comparison is particularly interesting since both algorithms employ an ACO-based procedure to create a comprehensible classification model, while using different solution representations and search strategies. Additionally, decision trees can be converted to a set of classification rules (e.g., as discussed in [31]) and *vice-versa* (e.g., as discussed in [1]). Table 5 presents the results of Ant-Tree-Miner and *c*Ant-Miner$_{\mathrm{PB}}$ in terms of both predictive accuracy and size of the classification model, using the same tenfold cross-validation procedure described in subsection 4.1. In the case of *c*Ant-Miner$_{\mathrm{PB}}$, the size corresponds to the total number of rules in the discovered lists, while the size in the case of Ant-Tree-Miner corresponds to the number of leaf nodes in the discovered trees—given that the

Table 5: Average predictive accuracy (*average* ± *standard error*) in % and average model size (*average* ± *standard error*), measured by tenfold cross-validation. The value of the best performing algorithm for each data set is shown in bold.

	Predictive Accuracy		*Model Size*	
Data Set	Ant-Tree-Miner	*c*Ant-Miner$_{\text{PB}}$	Ant-Tree-Miner	*c*Ant-Miner$_{\text{PB}}$
auto	**77.2 ± 0.4**	65.5 ± 0.9	31.1 ± 0.6	**14.3 ± 0.1**
balance	62.9 ± 0.1	**76.8 ± 0.2**	34.5 ± 0.0	**13.6 ± 0.0**
blood-t	**72.4 ± 0.1**	72.3 ± 0.1	14.8 ± 0.1	**9.8 ± 0.0**
breast-l	**73.5 ± 0.1**	72.3 ± 0.3	**10.0 ± 0.2**	10.4 ± 0.1
breast-t	65.2 ± 0.6	**67.1 ± 0.5**	12.0 ± 0.1	**6.7 ± 0.0**
breast-w	94.0 ± 0.2	**94.3 ± 0.2**	9.0 ± 0.1	**8.3 ± 0.1**
credit-a	**85.9 ± 0.1**	85.7 ± 0.1	29.6 ± 0.2	**12.3 ± 0.1**
derma	**94.4 ± 0.1**	92.5 ± 0.3	20.7 ± 0.1	**19.2 ± 0.1**
ecoli	**83.9 ± 0.1**	79.9 ± 0.2	15.9 ± 0.1	**9.1 ± 0.0**
glass	71.0 ± 0.4	**73.9 ± 0.2**	20.2 ± 0.1	**9.4 ± 0.1**
heart-c	51.9 ± 0.4	**55.5 ± 0.4**	35.8 ± 0.2	**12.8 ± 0.1**
heart-h	**65.9 ± 0.2**	64.7 ± 0.3	21.8 ± 0.1	**11.0 ± 0.1**
hep	**82.5 ± 0.3**	80.1 ± 0.5	**7.5 ± 0.1**	**7.6 ± 0.1**
horse	**83.8 ± 0.1**	83.5 ± 0.3	**10.2 ± 0.1**	10.8 ± 0.2
ionos	**90.8 ± 0.2**	89.6 ± 0.3	12.4 ± 0.1	**9.2 ± 0.1**
iris	**96.2 ± 0.1**	93.2 ± 0.2	**4.2 ± 0.0**	4.8 ± 0.0
park	**92.4 ± 0.1**	87.0 ± 0.5	8.0 ± 0.0	**7.1 ± 0.1**
s-heart	**77.3 ± 0.3**	77.0 ± 0.4	18.1 ± 0.2	**8.6 ± 0.1**
soybean	**87.4 ± 0.2**	80.0 ± 0.3	50.0 ± 0.3	**22.3 ± 0.1**
voting	**94.9 ± 0.1**	93.3 ± 0.2	**6.1 ± 0.0**	**6.4 ± 0.1**
wine	**96.2 ± 0.1**	93.6 ± 0.3	5.6 ± 0.0	**5.4 ± 0.1**
zoo	**88.2 ± 0.2**	78.6 ± 0.9	10.7 ± 0.0	**6.0 ± 0.0**

path from a leaf node to the root node of the tree can be viewed as a rule.

The performance of Ant-Tree-Miner in terms of predictive accuracy is statistically significantly better than the performance of *c*Ant-Miner$_{\text{PB}}$, according to the Wilcoxon signed-rank test [9] at the $\alpha = 0.05$ level (*p-value* $= 0.0461$). On the other hand, *c*Ant-Miner$_{\text{PB}}$ is statistically significantly better than Ant-Tree-Miner in terms of the size of the classification model, according to the Wilcoxon signed-rank test at the $\alpha = 0.05$ level (*p-value* $= 0.0001$). These results show that the decision trees created by Ant-Tree-Miner are more accurate than the rule lists created by *c*Ant-Miner$_{\text{PB}}$, at the expense of being bigger in size. As discussed in [31], it is possible to further reduce the complexity of the decision tree representation by converting a tree to a set of classification rules, where each rule can be individually simplified, while maintaining the same level of predictive accuracy. The application of such procedure is an interesting direction for future research.

5. Conclusions

This paper has proposed a novel ant colony algorithm, called Ant-Tree-Miner, for the induction of decision trees in the context of the classification task in data mining. When used as a classification model, decision trees have the advantage of representing a comprehensible model and being easily expressed in a graphical form or in a set of classification rules. The Ant-Tree-Miner algorithm follows a top-down approach by probabilistically selecting attributes to be added as decision nodes based on the amount of pheromone and heuristic information. The pheromone matrix representation takes into account the level in the decision tree that a particular branch—(attribute, value) pair—appears to discriminate between multiples occurrences of the same branch. Four variations of the Ant-Tree-Miner algorithm were compared against two well-known decision tree induction algorithms, namely C4.5 (Weka's J48 algorithm) and CART (Weka's SimpleCART algorithm), and the ACO-based *c*ACDT decision tree algorithm in 22 publicly available data sets in terms of both predictive accuracy and number of leaf nodes of the discovered decision trees.

The results have shown the Ant-Tree-Miner algorithm outperforms C4.5, CART and *c*ACDT in terms of predictive accuracy, achieving a statistically significantly better average rank across all data sets. It also presents a good trade-off between predictive accuracy and size of the classification model (number of leaf nodes in the discovered trees): Ant-Tree-Miner achieved the best rank in predictive accuracy and second best rank in number of leaf nodes; C4.5 achieved the fourth best rank in predictive accuracy and sixth rank in number of leaf nodes; CART achieved the sixth rank in predictive accuracy and first rank in number of leaf nodes; *c*ACDT achieved the seventh rank in predictive accuracy and the third rank in number of leaf nodes. The evaluation of different variations of the Ant-Tree-Miner algorithm have shown that the binary (single interval) discretisation in combination with the two-step (classification accuracy and C4.5's error-based) pruner provides the best performance in terms of both predictive accuracy and size of the classification model.

There are several potential avenues for future research. It would be interesting to investigate different heuristic information measures, as well as different discretisation procedures, which can improve the selection of attributes in decision nodes and consequently the structure of the decision trees. The use of vertices to represent leaf nodes in the construction graph, allowing the ants to choose when to add a leaf node, could potentially avoid the need for a pruning procedure. This would not only simplify the algorithm, but also provide an elegant solution for the problem of overfitting the training data during the construction of a candidate decision tree.

Software Availability The documentation, source-code and binaries of the Ant-Tree-Miner algorithm implementation used in this paper can be found at http://sourceforge.net/projects/myra.

Acknowledgement

We thank Jan Kozak for providing a binary version of the *c*ACDT algorithm and for his help in the preparation of the *c*ACDT's data sets.

References

[1] A. Abdelhalim, I. Traore, and B. Sayed. Rbdt-1: A New Rule-Based Decision Tree Generation Technique. In *Proceedings of the 2009 International Symposium on Rule Interchange and Applications*, pages 108–121, 2009.

[2] R.C. Barros, M.P. Basgalupp, A.C.P.L.F. de Carvalho, and A.A. Freitas. A Survey of Evolutionary Algorithms for Decision-Tree Induction. To appear in *IEEE Transactions Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 2011.

[3] H. Blockeel, L. Schietgat, J. Struyf, S. Džeroski, and A. Clare. Decision Trees for Hierarchical Multilabel Classification: A Case Study in Functional Genomics. In *Proceedings of the ECML PKDD*, pages 18–29, 2006.

[4] U. Boryczka and J. Kozak. Ant Colony Decision Trees – A New Method for Constructing Decision Trees Based on Ant Colony Optimization. In *Proceedings of the ICCCI*, pages 373–382, 2010.

[5] U. Boryczka and J. Kozak. An Adaptive Discretization in the ACDT Algorithm for Continuous Attributes. In *Proceedings of the ICCCI*, pages 475–484, 2011.

[6] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall, 1984.

[7] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley-Blackwell, 2nd edition, 2006.

[8] M. Daud and D. Corne. Human Readable Rule Induction In Medical Data Mining: A Survey Of Existing Algorithms. In *Proceedings of the WSEAS European Computing Conference*, 2007.

[9] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Machine Learning Research*, 7:1–30, 2006.

[10] V. Dhar, D. Chou, and F. Provost. Discovering Interesting Patterns for Investment Decision Making with GLOWER – A Genetic Learner Overlaid With Entropy Reduction. *Data Mining and Knowledge Discovery*, 4(4):251–280, 2000.

[11] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32, 1999.

[12] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[13] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[14] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Thirteenth International Joint Conference on Artifical Inteligence*, pages 1022–1027. Morgan Kaufmann, 1993.

[15] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smith. From data mining to knowledge discovery: an overview. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery & Data Mining*, pages 1–34. MIT Press, 1996.

[16] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[17] A.A. Freitas. Understanding the crucial role of attribute interaction in data mining. *Artificial Intelligence Review*, 16(3):177–199, 2001.

[18] A.A. Freitas, R.S. Parpinelli, and H.S. Lopes. Ant colony algorithms for data classification. In *Encyclopedia of Information Science and Technology*, volume 1, pages 154–159. 2 edition, 2008.

[19] A.A. Freitas, D.C. Wieser, and R. Apweiler. On the Importance of Comprehensible Classification Models for Protein Function Prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(1):172–182, 2010.

[20] S. García and F. Herrera. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Machine Learning Research*, 9:2677–2694, 2008.

[21] S. Izrailev and D. Agrafiotis. A Novel Method for Building Regression Tree Models for QSAR Based on Artificial Ant Colony Systems. *Journal of Chemical Information and Computer Sciences*, 41:176–180, 2001.

[22] D. Martens, B. Baesens, and T. Fawcett. Editorial survey: swarm intelligence for data mining. *Machine Learning*, 82(1):1–42, 2011.

[23] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.

[24] T. Mitchell. *Machine Learning*. McGraw Hill, 1st edition, 1997.

[25] F.E.B. Otero, A.A. Freitas, and C.G. Johnson. *c*Ant-Miner: an ant colony classification algorithm to cope with continuous attributes. In *Proc. of the 6th International Conf. on Swarm Intelligence (ANTS 2008), LNCS 5217*, pages 48–59. Springer-Verlag, 2008.

[26] F.E.B. Otero, A.A. Freitas, and C.G. Johnson. Handling continuous attributes in ant colony classification algorithms. In *Proc. of the 2009 IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*, pages 225–231. IEEE, 2009.

[27] F.E.B. Otero, A.A. Freitas, and C.G. Johnson. A New Sequential Covering Strategy for Inducing Classification Rules with Ant Colony Algorithms. To appear in *IEEE Transactions on Evolutionary Computation*, 2012.

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

[29] G. Piatetsky-Shapiro and W. Frawley. *Knowledge Discovery in Databases*. AAAI Press, 1991.

[30] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[31] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[32] J.R. Quinlan. Improved Use of Continuous Attributes in C4.5. *Artificial Intelligence Research*, 7:77–90, 1996.

[33] L. Rokach and O. Maimon. Top-Down Induction of Decision Trees Classifiers – A Survey. *IEEE Transactions on Systems, Man and Cybernetics*, 35(4):476–487, 2005.

[34] T. Stützle and H.H. Hoos. Improvements on the Ant System: Introducing $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. In *Proceedings of the International Conference Artificial Neural Networks and Genetic Algorithms*, 1997.

[35] T. Stützle and H.H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[36] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

[37] H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2nd edition, 2005.