A Mixed-Attribute Approach in Ant-Miner Classification Rule Discovery Algorithm

Ayah Helal
School of Computing
University of Kent
Chatham Maritime, UK
amh58@kent.ac.uk

Fernando E. B. Otero
School of Computing
University of Kent
Chatham Maritime, UK
F.E.B.Otero@kent.ac.uk

ABSTRACT

In this paper, we introduce Ant-Miner$_{\mathbf{MA}}$ to tackle mixed-attribute classification problems. Most classification problems involve continuous, ordinal and categorical attributes. The majority of Ant Colony Optimization (ACO) classification algorithms have the limitation of being able to handle categorical attributes only, with few exceptions that use a discretisation procedure when handling continuous attributes either in a preprocessing stage or during the rule creation. Using a solution archive as a pheromone model, inspired by the ACO for mixed-variable optimization (ACO$_{\mathbf{MV}}$), we eliminate the need for a discretisation procedure and attributes can be treated directly as continuous, ordinal, or categorical. We compared the proposed Ant-Miner$_{\mathbf{MA}}$ against *c*Ant-Miner, an ACO-based classification algorithm that uses a discretisation procedure in the rule construction process. Our results show that Ant-Miner$_{\mathbf{MA}}$ achieved significant improvements on computational time due to the elimination of the discretisation procedure without affecting the predictive performance.

CCS Concepts

•Computing methodologies → Supervised learning by classification;

Keywords

ant colony optimization, Ant-Miner, data mining, classification, continuous attributes

1. INTRODUCTION

Data mining is the process of extracting knowledge and patterns from data [1, 22]. One of the major data mining areas is classification. Classification is concerned in finding patterns in data sets, then use those patterns to classify any new (future) data. Classification problems can be viewed as optimization problems, where the aim is to create the best model that represent the predictive patterns in the data. The discovered classification model is then used to classify—predict the class attribute value of—new examples (unseen during training) based on the values of their predictor attributes. Using classification rules gives a more comprehensible classification model compared to black-box models such as SVMs or artificial neural networks [17], which are more difficult to be interpreted.

The first Ant Colony Optimization (ACO) classification algorithm, called Ant-Miner, was proposed in [16]. Ant-Miner and its extensions [7] were used to successfully extract *IF-THEN* classification rules from data. Each ant traverses a construction graph, where each node in the graph consists of a condition that the ant might choose to add to its rule. The majority of ACO-based classification algorithms are limited to cope only with categorical attributes, therefore continuous attributes must be discretised in a pre-processing stage. *c*Ant-Miner [13, 14] was the first Ant-Miner extension to cope with continuous attributes directly. It employs a dynamic discretisation procedure during the rule construction: when an ant select a node representing a continuous attribute, the discretisation procedure is used to determine cutoff value in a dynamic manner. A potential drawback of this approach is its computational time: discretisation requires sorting and the evaluation of multiple candidate cuttoff values, which can significantly increase the computational time of the algorithm when applied to large data sets.

In this paper we propose a new approach to extract *IF-THEN* of classification rules, based on the ACO for mixed-variable optimization (ACO$_{\mathbf{MV}}$) [3]. Our approach handles the mixed attributes directly: attributes are categorized as continuous, ordinal and categorical attributes. This takes full advantage of the mechanisms of ACO$_{\mathbf{MV}}$ to tackle mixed-variable optimization to cope with mixed-attribute classification problems. More importantly, it eliminates the need for a discretisation procedure, which improves the computational time for most of the large datasets. We compared the predictive accuracy and computational time performance of the proposed algorithm against *c*Ant-Miner in 30 publicly available data sets. Our results show that the proposed algorithm significantly improves the computational time when compared to *c*Ant-Miner while maintaining the predictive accuracy.

The remainder of this paper is organized as follows. We begin by reviewing the literature of ACO-based classification algorithms in Section 2. Then, we present an overview of *c*Ant-Miner and ACO$_{\mathbf{MV}}$ in Section 3. We then present our proposed Ant-Miner$_{\mathbf{MA}}$ algorithm in Section 4, com-

GECCO '16, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4206-3/16/07. . . $15.00

DOI: http://dx.doi.org/10.1145/2908812.2908900

putational results are presented in Section 5, and finally, conclusions and directions for future work are discussed in Section 6.

2. RELATED WORK

There are two main approaches to apply ant colony optimization to create classification rules: grammar- and graph-based approaches. In grammar-based approaches, the rule creation is guided by a context-free grammar, which determines the valid structure of rules. The Grammar-Based Ant Programming (GBAP) algorithm [9, 10] was the first implementation of a grammar-based approach. Similar to most of ACO-based classification algorithms, GBAP does not cope with continuous attributes directly and it uses a discretisation procedure in a preprocessing stage.

Graph-based approaches started with Ant-Miner [16], which was limited to discrete datasets only. Several extensions of Ant-Miner have been proposed [7]. Ant-Miner2 [6] and Ant-Miner3 [5] presented a simple heuristic function using density estimation. Ant-Miner+ [8] extended Ant-Miner in several aspects: it uses a class based heuristic, since an ant pre-selects the predicted class value and extracts a rule accordingly; it also employs a different pheromone initialization and update procedure based on $\mathcal{MAX}-\mathcal{MIN}$ ant system ($\mathcal{MMAS}$) [21], where the use of the lower and upper bound values allows the algorithm to avoid early stagnation of the search; and the complexity of the construction graph is reduced, in terms of the number of edges connecting vertices, by defining it as a direct acyclic graph (DAG).

Additionally, Ant-Miner+ employs a distinctive procedure for categorical and ordinal attributes. Categorical attributes have unordered nominal values (e.g., *male* and *female*), which were treated as a tuple ($attribute, =, value$). Ordinal attributes have a natural order (e.g., $poor < acceptable < good$), where the algorithm creates upper and lower bounds on the values chosen by the ant: the first type represents a lower bound of the interval and takes the form ($attribute$, $\leq$, $value_i$); the second type represents an upper bound of the interval and takes the form ($attribute$, $\geq$, $value_j$), where $value_i$ and $value_j$ are values from the attribute domain. Continuous attributes are discretised in a pre-processing stage and then treated as ordinal attributes.

*c*Ant-Miner was proposed in [13], where an entropy-based method is proposed for handling the continuous attributes discretisation during the rule construction process. The use of the minimum description length (MDL) principle in *c*Ant-Miner to allow construction of discrete intervals with lower and upper bounds was proposed in [14]. Further improvements in the *c*Ant-Miner are found in [18], where the authors proposed the use of multiple pheromone levels to extract rules predicting different class values. The latest improvements based on *c*Ant-Miner were presented in [15], where using a new sequential covering strategy, each ant creates and evaluates a rule list (considering interactions among rules) rather than creating and evaluating a single rule like most versions of Ant-Miner. More recently, an extension to discover unordered rules (set of rules) instead of ordered rules (list of rules), with the aim of improving the interpretability of the discovered rules, was proposed in [11, 12].

Despite the Ant-Miner extensions proposed in the literature, extending Ant-Miner to use a archive-based pheromone model to handle continuous and categorical values (i.e., using $\mathrm{ACO}_{\mathbf{MV}}$ model) is a research topic that has not been explored, to the best of our knowledge. Most of the proposed extensions handle continuous attributes in a pre-processing stage, while *c*Ant-Miner employs a dynamic discretisation procedure using a heuristic to define intervals.

3. BACKGROUND

3.1 *c*Ant-Miner

*c*Ant-Miner uses a graph-based approach to extract *IF-THEN* classification rules from data. Let r be a rule, each rule is a n-dimensional vector of terms t that are joined with *AND*, such that *IF* t_1 *AND* t_2 .... *AND* t_n *THEN (class)*, each term t_i consist of a tuple (attribute, operator, value).

The *c*Ant-Miner construction graph consists of a fully connected graph. Let a_i be a nominal attribute and v_{ji} be the j-th value of a_i attribute. For $j = 1, ..., b_i$, where b_i is the number of values of attribute a_i, each v_{ij} is added as a node (a_i, $=$, v_{ij}) to the graph. Let c_i be a continuous attribute, only one node is added with (c_i) to the graph—the operator and value are not defined for a continuous attribute node.

Suppose an ant x is generating a rule r_x. It starts with an empty rule at node i and probabilistically chooses to visit a node j based on the amount of pheromone and heuristic information on the edge E_{ij}, given by

$$P(E_{ij}) = \frac{\tau_{ij}^{\alpha} \cdot \eta_j^{\beta}}{\sum\limits_{l \in \text{ allowed i}} \tau_{il}^{\alpha} \cdot \eta_l^{\beta}} \tag{1}$$

where τ_{ij} is the pheromone value of the edge connecting node i to node j; η_j is the value of the heuristic information for node j; node l is a node in the neighbourhood of node i; the exponents α and β are used to control the influence of the pheromone and heuristic information, respectively.

If a node with a nominal attribute is selected, then a term in the form ($a_i = v_{ij}$) is added to the rule. If a node with a continuous attribute is selected, then a dynamic discretisation procedure based on the entropy measure is used to choose an operator and value to create a term in the form ($a_i \leq v_{ij}$) or ($a_i > v_{ij}$). This is done with a complexity of $O(n \log n)$, where n is the number of the training instances, since the values need to be sorted and multiple cutoff values are evaluated.

An ant keeps adding terms until the rule covers less training examples than a user specified threshold, or all attributes are already added to the rule.

3.2 $\mathrm{ACO}_{\mathbf{MV}}$

$\mathrm{ACO}_{\mathbf{MV}}$ [3] is applied to mixed variables optimization problems with r real-valued variables, c categorical-valued variables and o ordinal-valued variables. The $\mathrm{ACO}_{\mathbf{MV}}$ uses a solution archive (SA) as a form of pheromone model, instead of a pheromone matrix. The archive structure contains R previously generated solutions. Each solution S_j in the archive, for $j = \{1, 2, ..., R\}$, is a vector containing n-dimensional real-valued components, m-dimensional index-valued components and o-dimensional ordinal-valued components. The archive is sorted by the quality Q of solutions, so that $Q(S_1) \geq Q(S_2) \geq \cdots \geq Q(S_R)$.

Each solution S_j is associated with weight w_j that is related to $Q(S_j)$, where w_j is calculated using the Gaussian function given by

$$w_j = \frac{1}{qR\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2R^2}} \tag{2}$$

where q is a variable that is used to control the extend of the top-ranked solution influence on the construction of new solutions. The weight of solution S_j is used during the creation of new solutions, as an indicator for the level of attractiveness of this solution. The higher the weight of the solution S_j, the higher the probability of sampling a new solution around the values of S_j.

ACO$_{\mathbf{MV}}$ starts by randomly generating R solutions in the archive. The solution construction phase starts by each ant i generating a new S_i candidate solution. When constructing solution S_i, a probabilistic solution construction method is used to sample new values from the solution archive according to each attribute type. At the end of an iteration, all solutions created by the ants in the colony are added to the archive. The archive is sorted and only the best R solutions are kept and the remaining solutions are removed.

3.2.1 Continuous variables

Continuous variables are handled by ACO$_{\mathbf{MV}}$ using ACO$_{\mathbb{R}}$ [19], where each ant i probabilistically chooses one solution from the archive based on

$$P_j = \frac{w_j}{\sum\limits_{l=1}^{R} w_l} \tag{3}$$

where P_j is the probability of selecting the j-th solution from the archive to sample the new continuous variable value around it, R is the size of the archive, and w_j is the weight associated with the j-th solution in the archive. Let S_i denote a new solution sampled by ant i around the chosen solution S_j for continuous attribute a, the Gaussian probability density function (PDF) is given by

$$S_{i,a} \sim N(S_{j,a}, \sigma_{j,a}) \tag{4}$$

$$\sigma_{j,a} = \xi \sum_{r=1, j\neq r}^{R} \frac{|S_{j,a} - S_{r,a}|}{R-1} \tag{5}$$

where $S_{j,a}$ is the value of the variable a in the solution j of the archive, $\sigma_{a,j}$ is the average distance between the value of the variable a in the solution j and the value of a in all the other solutions in the archive and ξ is a user-defined value representing the convergence speed of the algorithm.

3.2.2 Ordinal variables

Ordinal variables are variables whose order have a meaning, e.g., *small* < *medium* < *large*. ACO$_{\mathbf{MV}}$ handles ordinal variables as continuous variables, where the continuous value is the index of the chosen value in the ordered attribute values. Then, a final step is to round up the value generated from Equation (4) to the nearest index. Using this relaxed continuous sampling allows the algorithm to take in consideration the order of the attribute values.

3.2.3 Categorical variables

Categorical variables are treated differently by ACO$_{\mathbf{MV}}$. Suppose a categorical variable i that has t possible values, each ant has to choose v_l where $v_l^i \in \{v_1^i, v_1, .., v_t^i\}$. The probability P_l^i to choose v_l is given by

$$P_l^i = \frac{\alpha_l}{\sum\limits_{l=1}^{t} \alpha_j} \tag{6}$$

where the variable α_l is weight function for each value of the categorical variable, based on the weight of the solution, calculated as

$$\alpha_l = \begin{cases} \frac{w_{ji}}{u_j} + \frac{q}{\kappa}, & \text{if } \kappa > 0, u_j > 0, \\ \frac{w_{ji}}{u_j}, & \text{if } \kappa = 0, u_j > 0, \\ \frac{q}{\kappa}, & \text{if } \kappa > 0, u_j = 0, \end{cases} \tag{7}$$

where α_l represents the weight associated with v_l, w_{ji} is the weight of the first solution that uses the value v_j in the archive, u_j is the number of solutions that use the value v_j in the archive, κ is the number of values in this attribute that are not used in the archive, and q is a variable that is used to control the extend of the top-ranked solution influence on the construction of new solutions.

The categorical sampling procedure allow an ant to consider two components when sampling a new value. The first component biases the sampling towards values that are used in high-quality solutions but do not occur very frequently in the archive. The second component biases the sampling towards unexplored values in that attribute.

4. MIXED-ATTRIBUTES IN ANT-MINER

The proposed algorithm Ant-Miner$_{\mathbf{MA}}$ uses an ACO$_{\mathbf{MV}}$ procedure to handle mixed attributes types, eliminating the need of an entropy-based discretisation when handling a continuous attribute, and also coping with ordinal attributes. The archive is used to sample conditions for the creation of the rules instead of traversing a construction graph. A high-level pseudocode of Ant-Miner$_{\mathbf{MA}}$ is shown in Algorithm 1.

Ant-Miner$_{\mathbf{MA}}$ starts with an empty list of rules (line 1), and iteratively (*outer while loop* in line 2) adds the best rule found along the iterative process to the list of rules (line 20), while the number of uncovered training examples is greater than a maximum uncovered value in a sequential covering fashion.

At each iteration, a single rule is created by an ACO$_{\mathbf{MV}}$ procedure (lines 3–18). It starts by initializing the archive with R random generated rules (line 3). Then, each ant generates a new rule (lines 6-11). Once m new rules have been generated, where m is the number of ants, they are added into the solution archive (line 12). The R and m rules are sorted and the m worst ones are removed from the archive. The procedure to create a new rule is repeated until the maximum number of iterations has been reached, or a restart procedure is once applied.

4.1 Archive and Rule Structure

As aforementioned, the archive consist of R rules. Each rule consist of a vector of n-dimensional terms, where n is the number of attributes in the dataset. Each term t_j in rule R_i contains a flag to indicate if this term is enabled or not, an operator and value(s). For continuous attributes, the operator could be either `LESS THAN OR EQUAL`, `GREATER THAN` or `IN RANGE`; categorical attributes' operator is always

	Continuous attribute (A_r)						Categorical attribute (A_c)					Ordinal attribute (A_o)						
	Flag	Op	Value1	Value2	. . .	. . .	Flag	Op	Value	. . .	. . .	Flag	Op	Value	. . .	. . .	f(S)	w
S_1	T	$>$	v_1	-	. . .	. . .	T	$=$	v_2	. . .	. . .	F	-	-	. . .	. . .	f(S_1)	w_1
S_2	T	InR	v_3	v_4	. . .	. . .	F	-	-	. . .	. . .	T	$\leq$	v_5	. . .	. . .	f(S_2)	w_2
S_3	F	_	_	_	. . .	. . .	T	$=$	v_6	. . .	. . .	T	$\geq$	v_7	. . .	. . .	f(S_3)	w_3
⋮	⋱						⋱					⋱					⋮	⋮
S_R	T	$\leq$	v_8	-	. . .	. . .	F	-	-	. . .	. . .	T	$\leq$	v_9	. . .	. . .	f(S_R)	w_R
	Continuous attributes						Categorical attributes					Ordinal attributes						

Figure 1: Archive Structure: example of 3 solutions of the archive, each solution showing a single example of each attribute type. A_r is a real-valued (continuous) attribute, A_c is a categorical attribute and A_o is an ordinal attribute.

Algorithm 1: High-level pseudocode of Ant-Miner$_{\mathbf{MA}}$

```
Data: training data
Result: list of rules
RuleList ← {}
while |TrainingData| < MaxUncovered do
    A ← Generate Random Rules
    while t < MaxIterations and not Restarted do
        $A_t$ ← {}
        while i < number of ants do
            $R_i$ ← Create New Rule
            $R_i$ ← Prune($R_i$)
            $i ← i + 1$
            $A_t ← R_i$
        end
        A ← UpdateArchive($A_t$)
        $t ← t + 1$
        if stagnation() then
            Restart(A)
            Restarted ← True
        end
    end
    $R_{best}$ ← BestRule(A)
    RuleList ← RuleList + $R_{best}$
    TrainingData ← TrainingData − covered($R_{best}$)
end
return RuleList
```

EQUAL; and ordinal attributes have an operator of either LESS THAN OR EQUAL or GREATER THAN OR EQUAL.

Figure 1 illustrates a solution archive with 3 solutions, each solution showing a single example of each attribute type. The solutions are stored according to their quality in the archive, where the best is stored at the top (highest ranking) and the worse at the bottom (worst ranking). The archive stores the quality of each rule in $f(S)$ and the weight in w calculated by Equation (2).

In Figure 1, A_r is an example of a real-valued (continuous) attribute in the dataset, A_c is an example of a categorical attribute in the dataset and A_o is an example of an ordinal attribute in the dataset. The solution S_1 is an example in which the continuous attribute A_r is enabled (Flag = T), the operator is GREATER THAN and value v_1 is set—representing the term $A_r > v_1$. The categorical attribute A_c is enabled (Flag = T), the default operator is EQUAL and value v_2 is set—representing the term $A_c = v_2$. The ordinal attribute A_o is not enabled (Flag = F) and so it has no values for operator and value.

The solution S_2 is an example in which the continuous attribute A_r is enabled (Flag = T), the operator is IN RANGE and two values v_3 and v_4 are set—representing the term $v_3 \leq A_r < v_4$. The categorical attribute A_c is not enabled showing (Flag = F) and so it has no values for operator and value. The ordinal attribute A_o is enabled (Flag = T), the operator is LESS THAN OR EQUAL and value v_5 is set—representing the term $A_o \leq v_5$. Solution S_3 follows a similar representation.

4.2 Archive Initialization

In the archive initialization procedure, each rule is randomly initialized. Rule initialization starts with an unbiased random probability to enable each the term, then according to its type it continues the rule initialization. If it is a continuous term, it uses an unbiased random probability to set the operator from the set {LESS THAN OR EQUAL, GREATER THAN, IN RANGE}. The value of the continuous attribute is generated using a random value from a normal distribution in the range of values of the attribute. In case of the IN RANGE operator, two values are generated and the values will only be accepted if they make the operator valid.

Ordinal terms also use an unbiased random probability to choose the operator from the set {LESS THAN OR EQUAL, GREATER THAN OR EQUAL}, then an unbiased random value for the index is generated. For categorical terms, a default EQUAL operator is added, then an unbiased random value for the index is generated.

After the random initialization of each rule, a rule is pruned to remove irrelevant terms enabled by the stochastic nature of the initialization. Then, if the number of of covered instances is greater or equal to the user-defined minimum limit, it will be added to the archive. Finally, the rules in the archive are ordered according to their quality, so that $f(S_1) \geq f(S_2) \geq \cdots \geq f(S_R)$.

4.3 Rule Creation

Rule creation follow the following steps:

1. For each term, we start by considering the probability to include the term or not. The decision is handled as a categorical variable, since we are dealing with boolean {`TRUE`, `FALSE`} values. We use the same procedure in Section 3.2.3 to probabilistic sample a boolean value.

2. If the term is enabled, we set the operator according to the attribute type. If the attribute type is categorical, we set the operator to `EQUAL`. If it is continuous, the decision is handled as a categorical attribute of three values {`LESS THAN OR EQUAL`, `GREATER THAN`, `IN RANGE`}, using the same procedure in Section 3.2.3 to probabilistic sample an operator based on the subset of the archive rules that have this term enabled. While for ordinal attributes, the decision is handled as a categorical attribute of two values {`LESS THAN OR EQUAL`, `GREATER THAN OR EQUAL`}, using the same procedure in Section 3.2.3 to probabilistic sample an operator based on the subset of the archive rules that have this term enabled.

3. The value of each attribute is then sampled. If the attribute type is continuous, we use the continuous sampling procedure in Section 3.2.1 based on the subset of the archive rules that have this term enabled and using the same operator. If the attribute is ordinal, we use the continuous sampling procedure in Section 3.2.2 to benefit from the order of the attribute, based on the subset of the archive rules that have this term enabled and using the same operator. While for categorical terms we use the discrete sampling procedure in Section 3.2.3 based on the subset of the archive terms that are enabled.

4. After the creation of each term, we apply the rule to the training set. If the rule covered less than the minimum instances, the term is disabled. If it covers more than the minimum instances, the term is added and the process is repeated for the next term, until all terms are considered.

4.4 Rule Pruning

Ant-Miner$_{\mathbf{MA}}$ applies different heuristics for the rule refinement and rule selection following a similar approach proposed in [20]. We use the m-estimate measure in the rule selection and archive ordering, given by

$$Q = \frac{TP + m \cdot (\frac{P}{P+N})}{TP + FP + m} \tag{8}$$

where TP and FP are number of the correct classified and misclassified instances by the rule, respectively; P and N are the total number of instances that are in the positive and negative class in training dataset, respectively.[1] The value $m = 22.466$ used in our approach has been determined experimentally in [2] to be optimum value for the m-estimate measure.

For the pruning function, we use the sensitivity specificity function, as employed in Ant-Miner, given by

Table 1: Parameters: Ant-Miner$_{\mathbf{MA}}$ uses the first three parameters in table, while remaining are used by both Ant-Miner$_{\mathbf{MA}}$ and *c*Ant-Miner.

Parameters	Value
q	0.025495
ξ	0.6795
R	90
Minimum Covered	10
Max Uncovered	10
Max Iterations	1500
Number of Ants	60
Stagnation Test	10

$$Q_{Pruning} = \frac{TP}{TP + FN} \cdot \frac{TN}{FP + TN} \tag{9}$$

where TP, FP, TN and FN stand for true positives, false positives, true negatives and false negatives, respectively.

The pruning function starts by disabling the last term in the rule and if the quality of the rule does not decrease, it permanently removes the term; this process is repeated to disable the next one until disabling a term decreases the quality of the rule or the rule has only one term enabled. In this case, the pruning stops, returning the rule without removing the last term. This is the same pruning procedure used by *c*Ant-Miner [14].

4.5 Restart Strategy

Ant-Miner$_{\mathbf{MA}}$ uses a simple restart strategy to avoid stagnation. The restart strategy is triggered if the best rule of the current iteration is exactly the same as the best rule constructed by a user defined number of previous iterations, which works as stagnation test. When the restart strategy is triggered, all the rules in the archive are reinitialized except the best-so-far rule. The restart is performance only once.

5. COMPUTATIONAL RESULTS

Our computational results are calculated using 30 publicly available dataset from the UCI Machine Learning Repository [4], presented in Table 2. Ant-Miner$_{\mathbf{MA}}$ uses the first three parameters in Table 1 for the archive settings, while the rest of the parameters are used by both Ant-Miner$_{\mathbf{MA}}$ and *c*Ant-Miner. The archive settings parameters were empirical chosen based on preliminary experiments. The remaining parameters were based on *c*Ant-Miner default values [14]. The *c*Ant-Miner implementation used in the experiments is *c*Ant-Miner2$_{MDL}$ [14][2], which can create intervals with lower and upper bounds for continuous attributes.

We ran both algorithms for 15 times in a tenfold cross-validation settings, the average results (over the 150 individual runs) are shown in Table 3. We ran Wilcoxon rank sum test on the 30 datasets to show if there are statistically significant differences in terms of both predictive accuracy and computational time. For a fair comparison, both algorithms are implemented in Java running on the same environment.

[1] An instance is considered negative if it is from a class different than the class predicted by the rule

[2] We used the *c*Ant-Miner2$_{MDL}$ variation available from https://github.com/febo/myra.

Table 2: Summary of the datasets used in the experiments: datasets from 1 to 18 are considered small datasets, while the remaining ones are considered large datasets due to either the large number of attributes or the large number of instances.

				Attributes			
#	Dataset	Size	#Classes	Total	#Ordinal	#Categorical	#Continuous
1	breast-tissue	106	6	9	0	0	9
2	iris	150	3	4	0	0	4
3	wine	178	3	13	0	0	13
4	parkinsons	195	2	22	0	0	22
5	glass	214	7	9	0	0	9
6	breast-l	286	2	9	4	5	0
7	heart-h	294	5	13	3	3	7
8	heart-c	303	5	13	3	3	7
9	liver-disorders	345	2	6	0	0	6
10	ionosphere	351	2	34	0	0	34
11	dermatology	366	6	34	33	0	1
12	cylinder-bands	540	2	35	2	14	19
13	breast-w	569	2	30	0	0	30
14	balance-scale	625	3	4	4	0	0
15	credit-a	690	2	14	4	4	6
16	pima	768	2	8	0	0	8
17	annealing	898	6	38	0	29	9
18	credit-g	1000	2	20	11	2	7
19	MiceProtein	1080	8	80	0	3	77
20	HillValley	1212	2	100	0	0	100
21	Magic	19020	2	10	0	0	10
22	Nomao	34465	2	118	0	29	89
23	bank-additional	41188	2	20	0	10	10
24	eb	45781	31	3	0	1	2
25	adult	48842	2	14	0	8	6
26	connect4	67557	3	42	0	42	0
27	diabetes	101766	3	47	2	34	11
28	SkinNonSkin	245057	2	3	0	0	3
29	ForestType	581012	7	54	0	44	10
30	PokerHand	1025010	10	10	5	0	5

Table 3 shows that Ant-Miner$_{\mathbf{MA}}$ has a higher average rank of 1.43 for the predictive accuracy, where *c*Ant-Miner has an average rank of 1.57. In terms of computational time, Ant-Miner$_{\mathbf{MA}}$ has a rank of 1.10, while *c*Ant-Miner has a rank of 1.87. Most of the datasets show an order of magnitude improvement of Ant-Miner$_{\mathbf{MA}}$ over *c*Ant-Miner in terms of computational time. Considering both number of attributes and instances size, the largest datasets are *forest type*, *poker hand* and *diabetes*, respectively. Most notably, *c*Ant-Miner running a single *diabetes* execution takes up to 3.5 days, while Ant-Miner$_{\mathbf{MA}}$ takes just over 1 hour. Ant-Miner$_{\mathbf{MA}}$ would take 45 minutes for a single run in *poker hand*, while *c*Ant-Miner almost 8 hours. These results show that the use of the solution archive as pheromone model does not affect the accuracy, while improving the computational time since the discretisation procedure can be eliminated.

Our approach seems to show a small limitation when the number of attributes increases over 50, where we could observe an improvement in computational time of only around 25% in *forest type* dataset. In cases where there is a large number of attributes but a smaller number of instances—which means that the discretisation overhead is less noticeable—such as in the datasets *Nomao* (119 attributes, 34465 instances), *Hill Valley* (101 attributes, 1212 instances) and *Mice Protein* (81 attributes, 1080 instances), Ant-Miner$_{\mathbf{MA}}$ running time increases in relation to *c*Ant-Miner. We hypothesised that this is due to the use of heuristic information in *c*Ant-Miner, which allows the algorithm to quickly identify irrelevant attributes and not use them in the rule creation process.

Table 4 indicates that there is no statistically significantly differences between Ant-Miner$_{\mathbf{MA}}$ and *c*Ant-Miner in term of predictive accuracy ($p = 0.24200$). In the case of computational time, Ant-Miner$_{\mathbf{MA}}$ improvement is statistically significant ($p = 0.00036$). Overall, we consider these results positive. The use of a rule creation process inspired by ACO$_{\mathbf{MV}}$ led to a statistically significant computational time improvement in Ant-Miner$_{\mathbf{MA}}$ compared to *c*Ant-Miner, without affecting the predictive accuracy.

6. CONCLUSION

We introduced Ant-Miner$_{\mathbf{MA}}$ to tackle mixed-attribute classification problems, based on ACO$_{\mathbf{MV}}$. The use of a solution archive allows the algorithm to deal with categor-

Table 3: Average predictive accuracy and computational time (*average* [*standard deviation*]) of *c*Ant-Miner and Ant-Miner$_{\mathbf{MA}}$ over 15 runs of tenfold cross-validation. The last row of the table shows the average rank of the algorithm. The best value of a given dataset is shown in bold.

	Accuracy		Computational time (seconds)	
Dataset	Ant-Miner$_{\text{MA}}$	*c*Ant-Miner	Ant-Miner$_{\text{MA}}$	*c*Ant-Miner
breast-tissue	60.24 [0.97]	**64.24 [0.24]**	**0.38 [0.01]**	0.67 [0.02]
iris	93.6 [0.31]	**94.27 [0.11]**	**0.28 [0.00]**	0.49 [0.00]
wine	90.78 [0.40]	**93.52 [0.07]**	**0.33 [0.01]**	0.56 [0.04]
parkinsons	**86.29 [0.64]**	85.22 [0.40]	**0.78 [0.01]**	2.87 [0.25]
glass	**63.24 [0.50]**	59.18 [0.32]	**0.50 [0.00]**	2.66 [0.42]
breast-l	71.46 [0.34]	**76.17 [0.11]**	**0.54 [0.01]**	1.28 [0.14]
heart-h	64.37 [0.29]	**64.81 [0.33]**	**0.72 [0.00]**	12.61 [0.62]
heart-c	56.94 [0.54]	**57.42 [0.32]**	**0.76 [0.02]**	10.91 [0.64]
liver-disorders	**63.13 [0.49]**	62.26 [0.18]	**0.47 [0.01]**	1.81 [0.08]
ionosphere	**89.28 [0.37]**	88.84 [0.25]	**1.27 [0.03]**	5.97 [0.65]
dermatology	**89.42 [0.47]**	89.02 [0.25]	**1.31 [0.02]**	16.67 [1.49]
cylinder-bands	69.32 [0.35]	**70.28 [0.22]**	**3.15 [0.08]**	29.54 [1.31]
breast-w	93.53 [0.22]	**94.28 [0.11]**	**2.31 [0.04]**	5.40 [0.29]
balance-scale	**80.10 [0.22]**	68.34 [0.08]	**0.50 [0.01]**	5.95 [0.38]
credit-a	85.19 [0.22]	**85.74 [0.11]**	**1.10 [0.01]**	11.57 [0.79]
pima	**75.30 [0.21]**	67.45 [0.07]	**0.93 [0.01]**	3.69 [0.42]
annealing	96.68 [0.14]	**97.02 [0.09]**	**4.79 [0.16]**	10.76 [0.77]
credit-g	**74.19 [0.14]**	69.39 [0.17]	**1.69 [0.02]**	39.1 [2.11]
MiceProtein	62.57 [0.37]	**99.07 [0.43]**	26.79 [0.61]	**8.53 [0.65]**
HillValley	**52.65 [0.19]**	51.35 [0.11]	37.93 [0.71]	**19.12 [0.91]**
Magic	**82.67 [0.05]**	70.41 [0.01]	**25.56 [0.61]**	155.07 [2.01]
Nomao	87.56 [0.05]	**90.66 [0.02]**	2308.57 [98.25]	**779.77 [28.15]**
bank-additional	89.49 [0.02]	**89.87 [0.01]**	**185.27 [2.49]**	1970.72 [60.55]
eb	**65.00 [0.05]**	64.58 [0.01]	**25.38 [0.36]**	321.58 [1.11]
adult	**84.74 [0.03]**	79.73 [0.04]	**236.54 [3.83]**	5048.44 [165.33]
connect4	**68.18 [0.02]**	67.83 [0.01]	**1273.66 [7.97]**	14380.04 [535.72]
diabetes	**55.83 [0.09]**	54.23 [0.13]	**4008.11 [141.34]**	∗ 388023.6 [3580.67]
SkinNonSkin	**98.91 [0.02]**	97.54 [0.00]	**125.41 [1.56]**	1484.52 [5.27]
ForestType	**68.55 [0.07]**	63.09 [0.07]	**30649.92 [695.20]**	53336.53 [3946.86]
PokerHand	**51.97 [0.04]**	50.2 [0.00]	**2577.19 [43.07]**	27872.59 [2286.18]
Rank	**1.43**	1.57	**1.10**	1.87

∗ Result of a single tenfold execution due to high computational time.

Table 4: Wilcoxon rank sum tests (at the $\alpha = 0.05$ level) on predictive accuracy and computational time. Statistically significant differences are shown in bold.

	Size	W+	W-	Z	*p*
Accuracy	30	289.5	175.5	-1.1724	0.24200
Time	30	59	406	-3.5686	**0.00036**

ical, continuous and ordinal attributes directly, without required a discretisation procedure. The rule creation then uses ACO$_{\text{MV}}$ strategies to sample values for each attribute to create the antecedent of a rule.

We compared Ant-Miner$_{\text{MA}}$ against *c*Ant-Miner using publicly available datasets, an ACO-based classification algorithm capable of dealing with continuous attributes using a dynamic discretisation procedure. Our results show that the proposed Ant-Miner$_{\text{MA}}$ statistically significantly improves the computational time of the algorithm with no negative effects on its accuracy—in most cases, an order of magnitude improvements were observed. This would enable Ant-Miner$_{\text{MA}}$ to be applied to much larger datasets, mitigating the restriction on computational time.

There are several interesting directions for future research. First, it would be interesting to further investigate the effect of the number of attributes in the rule creation process. Our current results indicates that a large number of attributes might affect Ant-Miner$_{\text{MA}}$ computational time. Second, ACO$_{\text{MV}}$ does not provide an explicit mechanism to incorporate heuristic information, which can be used to quickly identify irrelevant attributes. Therefore, incorporating heuristic information into Ant-Miner$_{\text{MA}}$ can potentially improve its computational time by allowing the algorithm quickly identify irrelevant attributes. In addition, an extension to create a complete list of rules instead of one rule at a time is a research direction worth further exploration.

7. REFERENCES

[1] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth.

Advances in knowledge discovery and data mining. chapter From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[2] F. Janssen and J. Fürnkranz. On the quest for optimal rule learning heuristics. *Machine Learning*, 78(3):343–379, 2010.

[3] T. Liao, K. Socha, M. Montes de Oca, T. Stützle, and M. Dorigo. Ant colony optimization for mixed-variable optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(4):503–518, 2014.

[4] M. Lichman. UCI Machine Learning Repository, 2013. Irvine, CA: University of California, School of Information and Computer Science. [http://archive.ics.uci.edu/ml].

[5] B. Liu, H. Abbas, and B. McKay. Classification rule discovery with ant colony optimization. In *IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, pages 83–88, Oct 2003.

[6] B. Liu, H. A. Abbass, and B. McKay. Density-Based Heuristic for Rule Discovery with Ant-Miner. In *The 6th Australia-Japan Joint Workshop on Intelligent and Evolutionary System*, pages 180–184, 2002.

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

[8] D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens. Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11(5):651–665, Oct 2007.

[9] J. Olmo, J. Romero, and S. Ventura. A grammar based ant programming algorithm for mining classification rules. In *2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, July 2010.

[10] J. Olmo, J. Romero, and S. Ventura. Classification rule mining using ant programming guided by grammar with multiple pareto fronts. *Soft Computing*, 16(12):2143–2163, 2012.

[11] F. Otero and A. Freitas. 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, 2013. ACM Press.

[12] F. Otero and A. Freitas. Improving the Interpretability of Classification Rules Discovered by an Ant Colony Algorithm: Extended Results. *Evolutionary Computation (in press)*, pages 1–25, 2015.

[13] F. Otero, A. Freitas, and C. Johnson. cAnt-Miner: An Ant Colony Classification Algorithm to Cope with Continuous Attributes. In *Ant Colony Optimization and Swarm Intelligence*, volume 5217 of *Lecture Notes in Computer Science*, pages 48–59. Springer Berlin Heidelberg, 2008.

[14] F. Otero, A. Freitas, and C. Johnson. Handling continuous attributes in ant colony classification algorithms. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM '09)*, pages 225–231, March 2009.

[15] F. Otero, A. Freitas, and C. Johnson. A new sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions Evolutionary Computation*, 17(1):64–76, 2013.

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

[17] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

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

[19] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173, 2008.

[20] J. Stecher, F. Janssen, and J. Fürnkranz. Separating rule refinement and rule selection heuristics in inductive rule learning. In T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8726 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin Heidelberg, 2014.

[21] T. Stützle and H. H. Hoos. Max-min ant system. *Future Gener. Comput. Syst.*, 16(9):889–914, June 2000.

[22] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.