*c*Ant-Miner: An Ant Colony Classification Algorithm to Cope with Continuous Attributes

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

Computing Laboratory, University of Kent, Canterbury, UK
{febo2,A.A.Freitas,C.G.Johnson}@kent.ac.uk

Abstract. This paper presents an extension to Ant-Miner, named *c*Ant-Miner (Ant-Miner coping with continuous attributes), which incorporates an entropy-based discretization method in order to cope with continuous attributes during the rule construction process. By having the ability to create discrete intervals for continuous attributes "on-the-fly", *c*Ant-Miner does not requires a discretization method in a preprocessing step, as Ant-Miner requires. *c*Ant-Miner has been compared against Ant-Miner in eight public domain datasets with respect to predictive accuracy and simplicity of the discovered rules. Empirical results show that creating discrete intervals during the rule construction process facilitates the discovery of more accurate and significantly simpler classification rules.

1 Introduction

Data mining is a multi-disciplinary field which aims to extract knowledge from databases [1]. The data mining task addressed in this paper is the classification task, where the goal is to predict the class of an example, given the values of a set of attributes for that example. In essence, the classification task consists of inducing a model from the data by observing relationships between predictor attributes and classes, which can be used later to classify new examples. The discovered knowledge is often represented in the form of *IF* (*conditions*) *THEN* (*class*) classification rules, which has the advantage of representing a comprehensible model to the user [2].

In the context of discovering classification rules in data mining, Ant Colony Optimization (ACO) [3] algorithms have been successfully applied to different classification problems [4]. Ant-Miner [5], [6], the first implementation of an ACO algorithm for the classification task of data mining, has been shown to be competitive with the well-known C4.5 [7] and CN2 [8] classification-rule discovery algorithms.

Although real-world classification problems are often described by nominal (with a finite number of nominal or discrete values) and continuous (real-valued) attributes, Ant-Miner has the limitation of being able to cope only with nominal attributes in its rule construction process. In order to overcome this limitation, a commonly used approach is to discretize continuous attributes in a preprocessing step. In essence, a discretization method aims at converting continuous

attributes into nominal (discrete) attributes by creating interval boundaries (e.g. a continuous attribute *age* might be discretized into "0−14", "15−24", "25−64" and "65+" intervals). A potential disadvantage of this approach is that less information will be available to the classifier – since the discrete intervals have a coarser granularity – which can have a negative impact on the accuracy of the discovered knowledge.

This paper proposes an extension to Ant-Miner, named *c*Ant-Miner (Ant-Miner coping with continuous attributes), which incorporates an entropy-based discretization method in order to cope with continuous attributes during the rule construction process. *c*Ant-Miner has the ability to create discrete intervals for continuous attributes "on-the-fly", taking advantage of all continuous attributes information, rather than requiring that a discretization method be used in a preprocessing step. Note that, although many Ant-Miner variations have been proposed – as reviewed in [4] – none of them can discretize attributes "on-the-fly" (during the rule construction process) as proposed in this paper.

The remainder of this paper is organized as follows. Section 2 presents an overview of Ant-Miner. Section 3 discusses some Ant-Miner variations proposed in the literature. In Section 4, the proposed *c*Ant-Miner algorithm is introduced. Section 5 presents the computational results evaluating *c*Ant-Miner. Finally, Section 6 presents the conclusion of the paper and future research directions.

2 Ant-Miner Overview

The goal of Ant-Miner is to extract *IF-THEN* classification rules of the form *IF* $(term_1)$ *AND* $(term_2)$ *AND ... AND* $(term_n)$ *THEN* $(class)$ from data. Each term in the rule is a triple $(attribute, operator, value)$, where *operator* represents a relational operator and *value* represents a value of the domain of *attribute* (e.g. $Sex = male$). The *IF* part corresponds to the rule's antecedent and the *THEN* part is the rule's consequent, which represents the class to be predicted by the rule. An example that satisfies the rule's antecedent will be assigned the class predicted by the rule. As the original Ant-Miner only works with nominal (categorical or discrete) attributes, the only valid relational operator is "=" (equality operator). Continuous attributes need to be discretized in a preprocessing step.

Algorithm 1 presents a high level pseudo-code of Ant-Miner [6]. In essence, Ant-Miner works as follows. It starts with an empty rule list and iteratively adds one rule at a time to that list while the number of uncovered training examples is greater than a user-specified maximum value (*while* loop). In order to construct a rule, a single ant starts with an empty rule (no terms in its antecedent) and adds one term at a time to the rule antecedent (*repeat-until* loop). It probabilistically chooses a term to be added to the current partial rule based on the values of the amount of pheromone (τ) and a problem-dependent heuristic information (η) associated with the term. A pheromone value and a heuristic value are associated with each possible term – i.e. each possible triple $(attribute, operator, value)$. As usual in ACO, heuristic values are fixed (based on an information theoretical measure of the predictive power of the term), while

Algorithm 1: High level pseudo-code of Ant-Miner.

```
begin Ant-Miner
    training_set ← all training examples;
    rule_list ← ∅;
    while |training_set| > max_uncovered_training_examples do
        τ ← initializes pheromones;
        rule_best ← ∅;
        i ← 1;
        repeat
            rule_i ← CreateRule();
            ComputeConsequent(rule_i);
            Prune(rule_i);
            UpdatePheromones(τ, rule_i);
            if Q(rule_i) > Q(rule_best) then
                rule_best ← rule_i;
            end
            i ← i + 1;
        until i ≥ max_number_rules OR convergence ;
        rule_list ← rule_list ∪ rule_best;
        training_set ← training_set \ CorrectlyCoveredExamples(rule_best);
    end
end
```

pheromone values are iteratively updated based on the quality of the rules built by the ants. The ant keeps adding a term to the partial rule until any term added to the antecedent would make the rule cover less training examples than a user-specified threshold, which would make the rule too specific and unreliable, or all attributes have already been used by the ant. The latter rule construction stopping criterion is necessary because an attribute can only occur once in the antecedent of a rule, in order to avoid inconsistencies such as $(Sex = male)$ AND $(Sex = female)$. Once this process of rule construction has finished, first the rule constructed by the ant is pruned to remove irrelevant terms from the rule antecedent. Then, the consequent of the rule is chosen to be the class value most frequent among the set of training examples covered by the rule. Finally, pheromone trails are updated and another ant starts to construct a new rule. The process of constructing a rule is repeated until a user-specified number of rules has been reached, or the current ant has constructed a rule that is exactly the same as rules constructed by a predefined number of previous ants, which works as a rule convergence test. The best rule, based on a quality measure Q, found along this iterative process is added to the rule list and the correctly classified training examples are removed from the training set. An example is considered correctly classified if it satisfies the rule antecedent and has the class predicted by the rule consequent.

In [5], [6], Ant-Miner was compared against the well-known C4.5 [7] and CN2 [8] rule induction algorithms. In terms of predictive accuracy, the results have shown that Ant-Miner is competitive with both C4.5 and CN2. The biggest

difference found was related to the complexity of the discovered rules. Ant-Miner was able to find significant simpler rules, both in terms of a smaller number of rules and a smaller number of terms (conditions) per rule, than C4.5 and CN2.

3 Related Work on Ant-Miner Variations

Following the introduction of Ant-Miner, several variations were proposed [4]. They involve different pruning and pheromone update procedures, new rule quality measures and heuristic functions, discovering fuzzy classification rules and discovering rules for multi-label classification problems.

Chan & Freitas [9] have proposed a new rule pruning procedure for Ant-Miner. They have observed that the original Ant-Miner's pruning procedure processing time increases significantly with a large increase in the number of attributes, which affects the scalability of the method. To overcome this limitation, it was proposed a new prune procedure that led to the discovery of simpler (shorter) rules and improved the computational time in datasets with a large number of attributes.

Martens et al. [10] have introduced a new classification algorithm, named AntMiner+, based on Ant-Miner. It differs from the original Ant-Miner implementation in several aspects. Firstly, it makes a distinction between nominal and ordinal attributes. Nominal attributes have unordered nominal values (e.g. gender has unordered values "male" and "female"). Ordinal attributes are those categorical or discrete attributes whose values are ordered (e.g "0", "1", "2", "3" and "4 or more", which may be the domain of an attribute that represents the number of children in a family). Instead of creating a pair ($attribute = value$) for each value of an ordinal attribute, AntMiner+ creates two types of bounds that represent the interval of values to be chosen by the ants. The first type represents the lower bound of the interval and takes ($attribute \geq value_i$) form, and the second type represents the upper bound of the interval and takes ($attribute \leq value_j$) form ($value_i$ and $value_j$ are values from the attribute domain). Moreover, it employs different pheromone initialization and update procedures based on the $\mathcal{MAX} - \mathcal{MIN}$ ant system ($\mathcal{MM}$AS) [11]. For additional details refer to [10].

Galea & Chen [12] presented an ACO approach for the induction of fuzzy rules, named FRANTIC-SRL (Fuzzy Rules from ANT-Inspired Computation - Simultaneous Rule Learning). FRANTIC-SRL runs several ACO algorithm instances in parallel, where each instance generates rules for a particular class. By having separate ACO instances, separate pheromone matrices are maintained for each class.

Swaminathan [13] proposed an extension to Ant-Miner which enables interval conditions in the rules. While it still uses a discretization method to define discrete intervals for continuous attributes in a preprocessing step, the continuous values are not replaced in the dataset. For each discrete interval, a node (e.g. $humidity \leq 75$) is added to the construction graph and the pheromone value associated to the node is calculated using a mixed kernel probability density function (PDF).

Chan & Freitas [14] proposed a new ACO algorithm, named MuLAM (Multi-Label Ant-Miner) for the multi-label classification task. In a nutshell, MuLAM differs from the original Ant-Miner in three aspects. First, a classification rule can predict one or more class attributes, as in multi-label classification problems an example can belong to more than one class. Second, at each iteration, each ant constructs a set of rules instead of a single rule as in the original Ant-Miner. Third, it uses a pheromone matrix for each class attribute and pheromone updates only occur on the matrix of class attributes that occur in the rule's consequent.

Despite the Ant-Miner variations proposed in the literature, to the best of our knowledge, extending Ant-Miner to discretize continuous attributes "on-the-fly" (during the rule construction process) is a research topic that has not yet been explored. We believe that extending Ant-Miner to cope with continuous attributes "on-the-fly" would enhance its predictive accuracy given that the use of a discretization method in a preprocessing step can lead to loss of predictive power – since less information is available to the classication algorithm.

4 Handling Continuous Attributes in Ant-Miner

There are numerous discretization methods for handling continuous attributes available in the literature [15], [16]. These methods can be grouped according to different discretization strategies. Methods that make use of the examples' class information are referred to as *supervised*, while *unsupervised* methods do not use the class information (*supervised* vs. *unsupervised*). *Global* methods use the entire example space to define discrete intervals while *local* methods use a subset of example space (*global* vs. *local*). One can also categorize discretization methods as *static*, if they are applied in a data preprocessing phase before the classification algorithm is run, or as *dynamic*, if they are applied while a classifier is being built (*static* vs. *dynamic*). For a more detailed overview of different kinds of discretization methods, see [15], [16].

As mentioned in the previous section, the current version of Ant-Miner does not cope with continuous attributes directly. It requires continuous attributes to be discretized in a preprocessing step. In the experiments reported in [5], [6], the discretization method C4.5-Disc [17] was applied prior to Ant-Miner in a data preprocessing phase. In essence, the C4.5-Disc discretization method consists in using the well-known C4.5 [7] decision tree induction algorithm to create discrete intervals for each continuous attribute separately. For each continuous attribute, C4.5 is applied to a reduced dataset which only contains the attribute to be discretized and the class attribute. After the decision tree which contains binary splits referring only to the single attribute being discretized is built, each path of the tree from a leaf node to the root node corresponds to a discrete interval. For further details, refer to [17]. The C4.5-Disc discretization method would be categorized as *supervised*, *global* and *static* based on the criteria described above.

In this paper, we propose a *dynamic* discretization method which is incorporated into Ant-Miner's rule construction process and consequently avoids the

need for running a discretization method in a preprocessing step. First of all, we have extended the original Ant-Miner to support continuous attributes in the rule antecedent taking the form of ($attribute_c < value$) or ($attribute_c \geq value$), where $value$ is a value belonging to the domain of the continuous attribute $attribute_c$. Furthermore, we incorporated an entropy-based discretization method into Ant-Miner's rule construction process to dynamically create thresholds on continuous attributes domain values. The entropy measure, which is derived from information theory and often used in data mining, quantifies the impurity of a collection of examples and it is the same measure used as heuristic function in Ant-Miner. Details of the proposed Ant-Miner extension, named cAnt-Miner, are provided in the next sub-sections.

4.1 Construction Graph

The original Ant-Miner's construction graph consists of a fully connected graph in which for each nominal attribute a_i and value v_{ij} (where a_i is the i-th attribute and v_{ij} is the j-th value belonging to the domain of a_i), a node $(a_i = v_{ij})$ is added to the graph representing the $term_{ij}$ used to create a classification rule.

We have extended the construction graph to cope with continuous attributes as follows. For each continuous attribute a_i, we add a node (a_i) to the graph representing the $term_i$. Then, the node (a_i) is connected to all previous nodes of the construction graph. It should be noted that at this point the continuous values have not been discretized. The discretization occurs when an ant selects a node that represents a continuous attribute to be added to its current partial rule, as described in sub-section 4.3.

4.2 Heuristic Problem-Dependent Information

The heuristic value associated with each $term_{ij}$ in Ant-Miner involves a measure of entropy. In the case of nominal attributes, where every $term_{ij}$ has the form $(a_i = v_{ij})$, the entropy for the attribute-value pair is computed as in equation (1) – used in the original Ant-Miner:

$$entropy(a_i = v_{ij}) \equiv \sum_{c=1}^{k} -p(c \,|\, a_i = v_{ij}) \cdot \log_2 p(c \,|\, a_i = v_{ij}) \qquad (1)$$

where $p(c \,|\, a_i = v_{ij})$ is the empirical probability of observing class c conditional on having observed $a_i = v_{ij}$ and k is the number of classes. Note that the entropy is a measure of the impurity in a collection of examples, hence higher entropy values correspond to more uniformly distributed classes and smaller predictive power for the term in question.

However, the equation (1) cannot be straightforwardly applied to compute the entropy of nodes representing continuous attributes ($term_i$) since these nodes do not represent an attribute-value pair. In order to compute the entropy of

$term_i$, we need to select a threshold value v to dynamically partition the continuous attribute a_i into two intervals: $a_i < v$ and $a_i \geq v$. The best threshold value is the value v that minimizes the entropy of the partition, given by:

$$ep_v(a_i) \equiv \frac{|S_{a_i<v}|}{|S|} \cdot entropy(a_i < v) + \frac{|S_{a_i \geq v}|}{|S|} \cdot entropy(a_i \geq v) \qquad (2)$$

where $|S_{a_i<v}|$ is the total number of examples in the partition $a_i < v$ (partition of training examples where the attribute a_i has a value less than v), $|S_{a_i \geq v}|$ is the total number of examples in the partition $a_i \geq v$ (partition of training examples where the attribute a_i has a value greater or equal to v) and $|S|$ is the total number of training examples. After the selection of the threshold v_{best}, the entropy of the $term_i$ corresponds to the minimum entropy value of the two partitions and it is defined as:

$$entropy(term_i) \equiv min\left(entropy(a_i < v_{best}), entropy(a_i \geq v_{best})\right) \qquad (3)$$

We select the lowest entropy value since it corresponds to the value associated with the "purest" partition (the partition with more examples belonging to the same class) and it represents the expected predictive power (quality) of the $term_i$ (when $term_i$ is added to the rule). It should be noted that the entropy of every $term_i$ – i.e. every term having a continuous attribute – is always the same as the entropy value of every $term_{ij}$ – every term representing an attribute-value pair of a nominal attribute. Therefore, the entropy of all $term_i$ and $term_{ij}$ are computed as a preprocessing step to save computational time.

Concerning the computational complexity, the process of finding a threshold value can be divided into two steps. First, the continuous attribute values have to be sorted in order to facilitate the computation of the number of examples belonging to each candidate interval. The time complexity of this step is $O(n \log n)$, where n is the number of examples under consideration. Second, the evaluation of candidate threshold values has the complexity of $O(n)$, where in this case n represents the number of candidate values to be evaluated. It should be noted that not all candidate threshold values are evaluated, only those that form boundaries between classes, as proposed by [18]. Therefore, the efficiency of the evaluation is increased since less candidate threshold values need to be checked.

4.3 Rule Construction

As every term in the antecedent of a rule must be a triple (*attribute*, *operator*, *value*), when an ant chooses a node that represents a continuous attribute a_i to add to its current partial rule, a relational operator and a threshold value are selected as follows. First, the best threshold value for attribute a_i is selected as described in sub-section 4.2, subject to one restriction: only the examples covered by the current partial rule are considered in the evaluation of threshold values. Therefore, the selection of a threshold value is influenced by the terms

occurring in the current partial rule. This is what makes the proposed discretization method a *dynamic* one, so that the choice of a threshold value is tailored to the current candidate rule. The only exception to this restriction is when the current partial rule is empty. In those cases, all training examples are used in the evaluation of threshold values, as given by equation (2).

Then, after selecting the threshold value v_{best}, a relational operator op is selected based on the entropy values of the two partitions generated. If the partition $a_i < v_{best}$ has a lower entropy, then the operator "$<$" (less-than operator) is selected. If the partition $a_i \geq v_{best}$ has a lower entropy, then the operator "$\geq$" (greater-equal operator) is selected. The operator selection has a bias of selecting the more "pure" partition, given that lower entropy values are favored over higher entropy values.

Once the threshold value v_{best} and the operator op are selected, a term in the form of a triple (a_i, op, v_{best}) is added to the ant's current partial rule (e.g $age \geq 25$) and the rule continues to undergo the Ant-Miner's rule construction process.

4.4 Pheromone Updating

In the original Ant-Miner, every $term_{ij}$ has an associated pheromone value which undergoes the pheromone updating (increasing and decreasing) process. In summary, the pheromone updating process works as follows. The pheromone associated with each $term_{ij}$ occurring in the rule created by an ant is increased in proportion to the quality of the rule in question. The pheromone associated with each $term_{ij}$ that does not occur in the rule is decreased, simulating the pheromone evaporation effect observed in real ant colonies.

We have extended the original Ant-Miner's pheromone updating process to cope with $term_i$ (a term that represents a continuous attribute a_i) as follows. Since the pheromone value is associated with a continuous attribute a_i, and not the triple (a_i, op, v_{best}) that is added to the current partial rule (see sub-section 4.3), the operator op and the threshold value v_{best} are discarded in the updating process. In other words, there is a single entry for each continuous attribute a_i in the pheromone matrix, in contrast to multiple entries for nominal attributes, which have an entry for every (a_i, v_{ij}) pair (where a_i is the i-th nominal attribute and v_{ij} is the j-th value belonging to the domain of a_i).

In the proposed *c*Ant-Miner, pheromone is still used to indicate the quality of continuous attributes, but the actual choice of the threshold for each continuous attribute is dynamically customized to each rule being constructed by the ants. This effectively incorporates task-dependent knowledge into the algorithm, which tends to increase its effectiveness.

5 Computational Results and Discussion

In order to evaluate the proposed *c*Ant-Miner algorithm, we have selected eight datasets from the UCI Irvine machine learning repository [19] which had at least

one continuous attribute. Table 1 shows a summary of the selected datasets. All experiments were conducted running a well-known 10-fold cross-validation procedure [2]. Since the original version of Ant-Miner requires the data to be discretized in a preprocessing step, for each cross-validation fold we separately discretized (using the C4.5-Disc discretization method [17]) the training set and the created discrete intervals were used to discretize the test set. This was necessary because, if we had discretized the entire dataset before creating the cross-validation folds, the discretization method would have access to the test data, which would have compromised the reliability of the experiments. Also, we removed the duplicated instances (instances with the same values for all attributes) from the resulting discrete dataset to avoid the possibility that a test set contains an example that is the same as a training example.

We have compared the performance of *c*Ant-Miner against Ant-Miner, with respect to predictive accuracy and simplicity of the discovered rule lists. In all experiments, the user-defined parameters of *c*Ant-Miner and Ant-Miner were set to: *No_of_ants* = 3000, *min_cases_per_rule* = 5, *max_uncovered_cases* = 10 and *No_rules_converg* = 10 (detailed explanation of these parameters can be found in [6]). We have made no attempt to optimize these parameters for the datasets used in the experiments. It should be noted that for *c*Ant-Miner, the original – with nominal (or discrete) and continuous attributes – datasets were used, and for Ant-Miner, the discrete – with only nominal (or discrete) attributes – datasets were used. To make the comparison as fair as possible, the same cross-validation folds were used in both datasets, with the exception that in the discrete datasets we removed the duplicated examples.

Table 2 summarizes the results comparing the predictive accuracy of Ant-Miner and *c*Ant-Miner. Each entry in the table shows the average value of the accuracy obtained via the cross-validation procedure followed by the standard deviation. An entry in the *c*Ant-Miner column is shown in bold if, for the corresponding dataset, the accuracy achieved with *c*Ant-Miner was significantly greater than the accuracy achieved with Ant-Miner for that dataset – according to a two-tailed Student's t-test with significance level $\alpha = 5\%$. In two datasets, namely hepatitis and glass, *c*Ant-Miner was significantly more accurate than Ant-Miner. Both Ant-Miner and *c*Ant-Miner achieved similar (with no significant difference) accuracies in the remaining six datasets.

Table 3 summarizes the results concerning the simplicity of the discovered rule lists, measured by the total number of terms (conditions) in all discovered rules. Each entry in the table shows the average rule list size obtained via cross-validation procedure followed by the standard deviation. An entry in the *c*Ant-Miner column is shown in bold if, for the corresponding dataset, the rule list discovered by *c*Ant-Miner was significantly simpler than the rule list discovered by Ant-Miner for that dataset – according to a two-tailed Student's t-test with significance level $\alpha = 5\%$. Concerning the simplicity of the discovered rule lists, *c*Ant-Miner discovered significantly simpler rule lists than Ant-Miner in six out of eight datasets. In two datasets, namely crx and australian, both *c*Ant-Miner and Ant-Miner discovered rule lists with similar simplicity.

Table 1. Summary of the datasets used in the experiments. The first column gives the dataset name, the second and third columns give the number of nominal and continuous attributes respectively, the forth column gives the number of classes, the fifth column gives the number of instances in the original dataset and the sixth column gives the number of instances in the discrete dataset (after the removal of duplicated examples).

Dataset	Attributes		Classes	Size	
	Nominal	Continuous		Original	Discrete
wdbc	0	30	2	569	366
crx	9	6	2	690	639
hepatitis	13	6	2	155	116
glass	0	9	7	213	119
ionosphere	0	34	2	350	292
wine	0	13	3	178	126
australian	8	6	2	690	637
heart	6	7	2	270	232

Table 2. Predictive accuracy (*mean* ± *standard deviation*) of Ant-Miner and *c*Ant-Miner after the 10-fold cross-validation procedure. An entry in the *c*Ant-Miner column is shown in bold if, for the corresponding dataset, the accuracy achieved with *c*Ant-Miner was significantly greater than the accuracy achieved with Ant-Miner for that dataset – according to a two-tailed Student's t-test with significance level $\alpha = 5\%$.

Dataset	Ant-Miner	*c*Ant-Miner
wdbc	93.27 ± 1.44	95.57 ± 0.55
crx	85.32 ± 1.26	85.56 ± 1.16
hepatitis	74.61 ± 2.80	**84.89 ± 2.57**
glass	51.48 ± 4.84	**65.69 ± 2.59**
ionosphere	90.68 ± 2.32	90.00 ± 1.49
wine	94.58 ± 2.51	95.14 ± 2.01
australian	85.52 ± 1.60	86.60 ± 1.46
heart	77.62 ± 3.27	79.27 ± 2.74

The results obtained in the experiments can be summarized as follows. With respect to predictive accuracy, *c*Ant-Miner was significantly more accurate than Ant-Miner in the hepatitis and glass dataset. In the remaining six datasets, both *c*Ant-Miner and Ant-Miner achieved similar accuracies. Hence, overall *c*Ant-Miner was the most accurate algorithm for this set of eight datasets. Regarding the simplicity of the discovered rule lists, in six out of eight datasets, *c*Ant-Miner discovered rules significantly simpler than Ant-Miner's rules. These results em-

Table 3. Simplicity of the discovered rule list (*mean* ± *standard deviation*), measured as total number of terms in all rules, of Ant-Miner and *c*Ant-Miner after the 10-fold cross-validation procedure. An entry in the *c*Ant-Miner column is shown in bold if, for the corresponding dataset, the rule list discovered by *c*Ant-Miner was significantly simpler than the rule list discovered by Ant-Miner for that dataset – according to a two-tailed Student's t-test with significance level $\alpha = 5\%$.

Dataset	Ant-Miner	*c*Ant-Miner
wdbc	54.60 ± 5.16	**9.40 ± 0.70**
crx	25.60 ± 1.86	22.80 ± 2.32
hepatitis	26.10 ± 2.51	**16.00 ± 0.95**
glass	38.70 ± 1.52	**33.30 ± 1.87**
ionosphere	21.90 ± 3.08	**10.20 ± 1.00**
wine	8.80 ± 0.79	**6.30 ± 0.45**
australian	18.30 ± 1.16	21.40 ± 1.45
heart	20.40 ± 1.53	**16.40 ± 0.76**

pirically show that the proposed *dynamic*-discretization *c*Ant-Miner facilitates the discovery of more accurate and significantly simpler classification rules when compared to the *static*-discretization Ant-Miner.

An important remark is that there was no significant increase in the computational time of *c*Ant-Miner by comparison with Ant-Miner's time, since the number of examples that need to be considered when a continuous attribute is added to the rule decreases proportionally to the number of terms present in the current partial rule (see sub-sections 4.2 and 4.3).

6 Conclusion and Future Work

This paper has presented an extension to Ant-Miner, named *c*Ant-Miner, which copes with continuous attributes during the rule construction process. By having the ability to create discrete intervals for continuous attributes "on-the-fly", *c*Ant-Miner does not require a discretization method in a preprocessing step.

*c*Ant-Miner has been compared against Ant-Miner with respect to predictive accuracy and simplicity of the discovered rule lists in eight public domain datasets. Regarding predictive accuracy, *c*Ant-Miner significantly outperformed Ant-Miner in two datasets. Regarding simplicity of the discovered rule lists, *c*Ant-Miner found significantly simpler rule lists than Ant-Miner in six out of eight datasets. Therefore, the results obtained by *c*Ant-Miner are promising.

As future research direction, it would be interesting to extend the entropy-based discretization method used in the rule construction process to allow the creation of intervals with both lower and upper bound values in the form $v_{lower} \leq attribute \leq v_{upper}$.

References

1. Fayyad, U., Piatetsky-Shapiro, G., Smith, P.: From data mining to knowledge discovery: an overview. In Fayyad, U., Piatetsky-Shapiro, G., Smith, P., Uthurusamy, R., eds.: Advances in Knowledge Discovery & Data Mining, MIT Press (1996) 1–34
2. Witten, H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. 2nd edn. Morgan Kaufmann (2005)
3. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
4. Freitas, A., Parpinelli, R., Lopes, H.: Ant colony algorithms for data mining. To apper in *Encyclopedia of Info. Sci. & Tech.* 2nd Ed (2008)
5. Parpinelli, R., Lopes, H., Freitas, A.: An ant colony algorithm for classification rule discovery. In Abbass, H., Sarker, R., Newton, C., eds.: Data Mining: a Heuristic Approach, Idea Group Publishing (2002) 191–208
6. Parpinelli, R., Lopes, H., Freitas, A.: Data mining with an ant colony optimization algorithm. IEEE Transactions on Evolutionary Computation **6**(4) (2002) 321–332
7. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
8. Clark, P., Niblett, T.: The CN2 rule induction algorithm. Machine Learning **3**(4) (1989) 261–283
9. Chan, A., Freitas, A.: A new classification-rule pruning procedure for an ant colony algorithm. In: Artificial Evolution (Proceedings of the EA-2005). Lecture Notes in Artificial Intelligence 3871 (2005) 25–36
10. Martens, D., Backer, M.D., Haesen, R., Vanthienen, J., Snoeck, M., Baesens, B.: Classification with ant colony optimization. IEEE Transactions on Evolutionary Computation **11**(5) (2007) 651–665
11. Stützle, T., Hoos, H.: $\mathcal{MAX} - \mathcal{MIN}$ ant system. Future Generation Computer Systems **16**(8) (2000) 889–914
12. Galea, M., Shen, Q.: Simultaneous ant colony optimization algorithms for learning linguistic fuzzy rules. In Agraham, A., Grosan, C., Ramos, V., eds.: Swarm Intelligence in Data Mining, Springer-Verlag (2006) 75–99
13. Swaminathan, S.: Rule induction using ant colony optimization for mixed variable attributes. Master's thesis, Texas Tech University (2006)
14. Chan, A., Freitas, A.: A new ant colony algorithm for multi-label classification with applications in bioinformatics. In: Proc. Genetic and Evolutionary Computation Conference (GECCO-2006). (2006) 27–34
15. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and unsupervised discretization of continuous features. In: Machine Learning: Proceedings of the Twelfth Int. Conference on Artificial Intelligence, Morgan Kauffmann (1995) 194–202
16. Liu, H., Hussain, F., Tan, C., Dash, M.: Discretization: An enabling technique. Data Mining and Knowledge Discovery **6** (2002) 393–423
17. Kohavi, R., Sahami, M.: Error-based and entropy-based discretization of continuous features. In: Proceedings of the 2nd International Conference Knowledge Discovery and Data Mining, AAAI Press (1996) 114–119
18. Fayyad, U., Irani, K.: Multi-interval discretization of continuous-valued attributes for classification learning. In: Thirteenth International Joint Conference on Artifical Inteligence, Morgan Kaufmann (1993) 1022–1027
19. Asuncion, A., Newman, D.: UCI machine learning repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html (2007)