Discovering Regression Rules with Ant Colony Optimization

James Brookhouse
School of Computing
University of Kent
Chatham Maritime, UK
jb765@kent.ac.uk

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

ABSTRACT

The majority of Ant Colony Optimization (ACO) algorithms for data mining have dealt with classification or clustering problems. Regression remains an unexplored research area to the best of our knowledge. This paper proposes a new ACO algorithm that generates regression rules for data mining applications. The new algorithm combines components from an existing deterministic (greedy) separate and conquer algorithm—employing the same quality metrics and continuous attribute processing techniques—allowing a comparison of the two. The new algorithm has been shown to decrease the relative root mean square error when compared to the greedy algorithm. Additionally a different approach to handling continuous attributes was investigated showing further improvements were possible.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search - Heuristic methods

Keywords

ant colony optimization; data mining; regression rules; sequential covering

1. INTRODUCTION

Data mining is a research area concerned with the the automated search for useful and usable patterns in data [3]. There are many data mining tasks, which are broadly divided into descriptive (e.g., association rule mining, clustering) and predictive (e.g., regression, classification). The task investigated in this paper is the discovering of regression rules. Regression aims to create a model to predict a continuous dependent variable (attribute) based on a number of independent variables (regressor or predictor attributes). The continuous nature of the dependent variable sets regression apart from the classification task, which aims to predict the value of a nominal dependent variable—i.e., a variable that has a number of predefined categories (values). A classical real-world regression problem is the prediction of property rental prices (the continuous dependent variable) based upon independent variables including the floor area, location and year of construction. Conventional regression models range from linear equations to non-parametric equations [2]. Regression rules contain a list of logical conditions that, if satisfied, predict a dependent value—predictions can take the form of a numeric value or, in the more complex case, a linear model [4]. Similarly to classification rules, regression rules can be represented in a *IF-THEN* form, where the *IF* is regarded as the antecedent of the rule and contains logical conditions involving the predictor attributes, while the *THEN* is regarded as the consequent of the rule and contains the prediction. When combined as a list, regression rules provide a comprehensible prediction model.

Ant Colony Optimization (ACO) algorithms [1] have been successful at discovering classification rules, most notably Ant-Miner and its derivatives [13, 6, 12]. ACO algorithms use a colony of artificial ants, where each ant creates a candidate solution by selecting individual components based on pheromone and heuristic information derived from the problem domain. Components with a greater pheromone level are more likely to be selected by an ant. After creation, each ant solution quality is evaluated and components that are used in good quality solutions have their pheromone level increased while those that do not will have their pheromone reduced. Every iteration will produce a more refined solution until the colony converges on a set of near optimal solutions. Ant-Miner and its derivatives employ an ACO procedure to create classification rules, while the application of ACO algorithms to discover regression rules remains an unexplored area to the best of our knowledge.

In this paper we propose a sequential covering ACO algorithm to create an ordered list of regression rules. The proposed algorithm follows Ant-Miner's rule creation principle and regression-specific metrics of SeCoReg [5], a deterministic (greedy) sequential covering regression algorithm that creates a list of regression rules. This will allow us to evaluate the benefits of incorporating an ACO search in the creation of regression rules. A crucial aspect in the design of the proposed algorithm is the strategy to deal with continuous values in the antecedent of rules. We evaluated the same split point generation method as SeCoReg and also evaluated a second strategy, comparing its effects in both the proposed and SeCoReg algorithms.

ACO algorithms have been successfully applied to many combinatorial optimisation problems, classical examples are

GECCO'15 Companion, July 11–15, 2015, Madrid, Spain
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-3488-4/15/07...$15.00
DOI: http://dx.doi.org/10.1145/2739482.2768450

the travelling salesman problem and network routing [1]. The ordering of nodes in combinatorial problems is important in the final result, this feature is easily expressed in ACO algorithms as the ants walk between nodes producing a natural order. Similarly the antecedent of regression rules can be seen as an ordered set of terms that are used to make a prediction. The successful implementation of Ant-Miner for classification problems, which has been shown to outperform classical rule induction classification algorithms [12], provides evidence that ACO has the potential to perform well when applied to regression problems.

The rest of the paper is structured as follows. Firstly we present related work that has been conducted in related areas in Section 2, including sequential covering and ACO algorithms for classification. This will be followed by a description of the proposed algorithm in Section 3, discussing the rule quality measures and split point generation. Next, we present our results in Section 4 followed by a discussion in Section 5. Finally, in Section 6 we draw our conclusions and discuss possible areas for further work.

2. BACKGROUND

There are mainly two areas of related work, sequential covering and ACO-based classification algorithm. First, we will present the sequential covering algorithm followed by a description of relevant algorithms that use this strategy. Finally, a description of Ant-Miner—the first application of ACO for the classification task in data mining—is presented, as it is the base for the proposed algorithm presented in this paper.

2.1 Sequential Covering

A Sequential covering strategy, also known as separate and conquer, is commonly used in classification and regression to generate a list of rules. The pseudocode for sequential covering is shown in Algorithm 1.

Algorithm 1: SequentialCovering(Instances)

```
   Data: Instances
   Result: RuleList
1  RuleList ⟵ $\emptyset$
2  Rule ⟵ LearnOneRule(Instances)
3  while Performance(Rule, Instances) $>$ Threshold do
4      // Adds rule to list
5      RuleList ⟵ RuleList $\cup$ Rule
6      // Removes covered instances
7      Instances ⟵ Instances $-$ Covered(Rule, Instances)
8      // Creates the next rule
9      Rule ⟵ LearnOneRule(Instances)
10 end
11 // Adds the default rule
12 RuleList ⟵ RuleList $\cup$ DefaultRule
13 return RuleList
```

Sequential covering has a simple iterative structure, where each iteration generates a single rule using the procedure `LearnOneRule(Instances)`, which covers a number of previously uncovered instances. This new rule is then added to the ordered list of rules (line 5) that have been discovered so far. The newly covered instances are then removed from the training set (line 7) and the algorithm continues. The removal of the covered training instances guarantees that subsequent rules will cover a different subset of instances. Stopping criteria are normally defined so that the algorithm stops when an acceptable number of instances remain uncovered or the performance of the last rule drops below a threshold [8, 5].

A number of different algorithms exist that use the sequential covering strategy, where each algorithm provides a different implementation for the `LearnOneRule(Instances)` procedure. In this section we will be discussing three sequential covering algorithms that discover regression rules, namely M5'Rules [4], SeCoReg [5] and PSOminer [7].

M5'Rules is a wrapper for the M5 [14] algorithm. M5 generates regression trees, which contain linear models as leaf nodes instead of outputting a single value—the linear models enable M5 to improve the accuracy of the prediction. M5'Rules uses the sequential covering strategy to create a list of rules. In each iteration, an entire regression tree is generated. This tree is then flattened to produce a set of rules, where the best rule from the set is added to the list of rules. As M5, the prediction of the rules created by M5'Rules is made by a linear model, which compromise the comprehensibility of the rules that are produced. M5'Rules was tested using 30 continuous data sets with 10 fold cross validation. The results were compared to the original M5 implementation, where it was found that the M5'Rules rarely performed significantly worse but did significantly reduce the size of the rule list in 11 data sets, highlighting the advantage of the rule list representation over the regression tree. A particular interesting aspect of M5 for the design of regression algorithm is the strategy used to cope with continuous attributes: it chooses the split point that maximises the expected error reduction, where the error is measured as the standard deviation of the dependent variable in the generated subsets. We present more details of this procedure in Section 3, as it is one the strategies used in the proposed algorithm.

SeCoReg is a sequential covering algorithm that employs a top-down beam search strategy to create regression rules [5]. The strategy involves generating a list of possible modifications for the current rule; it then adds them one at a time searching for the modification that gives the best quality of all the modifications. If the new rule generated by the addition of the modification is better than the best-so-far it replaces the best rule. This process is repeated until there are no more modifications that can be added. The consequence of rules are obtained by calculating the mean value of all covered instances in the training data. The rule quality is defined as the product of two measures, the relative coverage and relative root mean square error—these measures are discussed in detail in Section 3.2.

The results presented for SeCoReg show that it does not perform significantly worse than a number of comparison algorithms including linear regression and SVMreg [15]. Finally the authors suggested a possible extension to the algorithm by replacing the prediction to a linear model in a similar fashion to M5'Rules, although they acknowledge the drawback of compromising comprehensibility.

PSOminer is a Particle Swarm Optimisation (PSO) based regression rule miner [7]. PSOminer uses a sequential covering strategy to build a list of rules that covers the training instances, using a PSO procedure to find high quality rules. PSOminer has the ability to parse both numeric and categorical attributes by encoding the attributes to each particle in the following manner. All attributes are encoded over a number of dimensions over the range of [0,1]. Numeric attributes do not require split points like M5 rules and SeCoReg, instead numeric attributes are mapped to two dimensions: the first dimension specifies the parameters lower bound while the second encodes the upper bound of each parameter. To allow unbounded attributes, for both lower and upper bounds independently, the domain high and low values are reserved to signify the parameter is unbound. Categorical attributes are encoded via dummy encoding, where a dimension is used for each allowed value and the dimension with the highest value is used to set the attributes value. Null is set through an additional dimension which enables an attribute to be unused. PSOminer is also limited to generating a list of rules which is constrained in size—in the experiments reported in [7], the rule list was limited to 5 or 10 rules. PSOminer showed promising results when compared against the reference SeCoReg implementation, outperforming with statistically significant differences when set to produce 10 rules (no differences were observed when set to produce only 5 rules).

2.2 Ant-Miner Overview

Ant Colony Optimization has been successfully used for generating classification rules for data mining applications, in particular Ant-Miner [13] and its derivatives. ACO algorithms are able to achieve this by creating rules based on the probabilistic decisions by artificial ants. The artificial ants are guided by a pheromone trail left behind by previous ants and problem-specific heuristic information. The pheromone trail gives the ants a feedback mechanism to promote individual terms (attribute-value conditions) used often in generating high quality rules. The pheromone also evaporates over time allowing the colony to forget poor decisions that they made in the past.

Ant-Miner requires a graph for the artificial ants to walk across while generating rules—Figure 1 shows a simplified graph of rule terms (nodes) that can be selected by an ant based on the probabilities derived from the pheromone and heuristic information. An ant would start at a random node and then move to another node by applying a stochastic selection based upon the pheromone and heuristic values of neighbouring nodes. Once a node is visited, it is added to the rule being created. The ant will continue to add new terms until either all attributes have been used or if the addition of term makes the rule to cover a number of examples below predefined threshold. Once all ants in the colony complete their rule, the best rule of the iteration is used to update the pheromone values—i.e., the pheromone associated with terms included in the best rule are increased, while the pheromone associated with unused terms are decreased. Following the sequential covering strategy, after a rule has been created, the covered instances are removed from the training set and the next rule is created. This process goes on until the training set has less than a predefined number of instances remaining.

*c*Ant-Miner [10] was adapted from Ant-Miner so that numeric attributes could be resolved at run time rather than a pre-processing step and thus require a small change to the graph the ant traverses. Figure 1 shows a graph of both categorical and numeric attributes, which allows an ant to walk between attribute-value pairs allowing the construction of a rule. Categorical attributes are added to the current rule

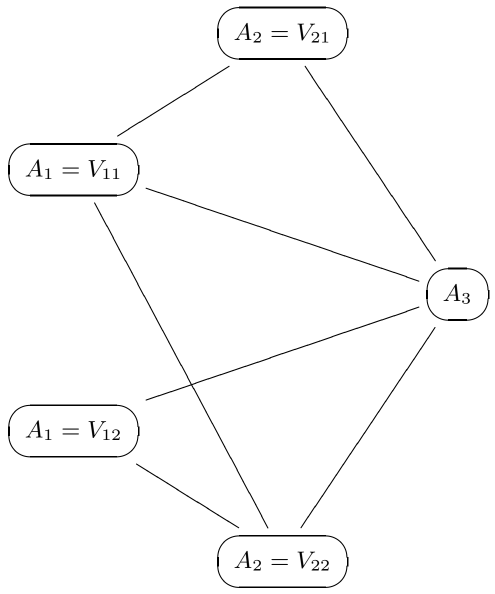

Figure 1: Simple construction graph that can be used to construct rules, including both categorical and numeric attributes.

if selected. Numeric attributes are handled differently: if a numeric attribute is chosen, a split point generation method is used to resolve a suitable partition in the data to create a complete term. The proposed algorithm follows a similar strategy, although it used measures to evaluate candidate split points suitable for regression problems.

3. DISCOVERING REGRESSION RULES

In this section we present the proposed Ant-Miner-Reg algorithm. The algorithm is based on both SeCoReg and Ant-Miner. The greedy rule construction procedure in SeCoReg has been replaced with an ACO implementation using the same rule quality measures and split point generation procedure. This will allow a comparison between the rule creation strategies. We also evaluated a second alternative split point generation method based on the M5 algorithm.

Rules for both approaches take the form of a sequence of (attribute, operator, value) tuples connected by ANDs in the antecedent of the rule. The operator for categorical values must be =, while continuous attributes are allowed to use the operators $<$ or $\geq$. Finally the rule will predict the mean value of the covered instances in the training data. An example rule is shown below:

$$IF\ att_1 \geq value_1\ AND\ att_2 = value_2\ THEN\ 3.5$$

where att_1 is a continuous attribute and att_2 is a categorical attribute. The value after the $THEN$ corresponds to the rule prediction (consequent of the rule).

3.1 Ant-Miner-Reg

The proposed Ant-Miner-Reg algorithm follows the sequential covering strategy presented in Algorithm 1. The difference is that the `LearnOneRule(Instances)` procedure is replaced by an ACO procedure to create the best rule

Algorithm 2: ACOLearnOneRule(Instances)

```
Data: Instances
Result: BestRule
1  BestEval ⟵ ∞, BestRule ⟵ null
2  PheromoneInitialization()
3  for $i = 0$ to $ant\_iterations$ do
4      // Each ant creates a rule, remembering
5      // the best for later
6      MaxEval ⟵ $-\infty$, MaxRule ⟵ null
7      for $j = 0$ to $colony\_size$ do
8          Rule ⟵ CreateRule()
9          PruneRule(MaxRule)
10         Eval ⟵ EvaluateRule(Rule, Instances)
11         if $Eval > MaxEval$ then
12             MaxEval ⟵ Eval
13             MaxRule ⟵ Rule
14         end
15     end
16     // Adds the best rule and update the
17     // pheromone levels with the best rule
18     // produced by the colony
19     UpdatePheromone(MaxRule)
20     // If the max rule is better than the best
21     // rule found from all iterations update
22     // the best rule
23     if $MaxEval > BestEval$ then
24         BestEval ⟵ MaxEval
25         BestRule ⟵ MaxRule
26     end
27 end
28 return $BestRule$
```

given a set of training instance at each iteration. The rule construction procedure is based on the *c*Ant-Miner algorithm [10], which allows the algorithm to deal with continuous attributes directly.[1] The high-level pseudocode for the implementation of the `ACOLearnOneRule(Instances)` is shown in Algorithm 2, where *ant_iterations* is the number of iterations and *colony_size* is the number of ants that walk the construction graph in each iteration. The pheromone levels of each node define the probability of a node being chosen as the next term in the rule—conventionally the probability of being chosen also relies on a problem specific heuristic, however in this case no heuristic information has been used. Therefore, the selection of nodes relies only on pheromone values. The pheromone matrix initialisation is achieved by setting the value of each cell to $1/matrix\_size$—this is the same technique used by Ant-Miner—and values are updated using the best rule created in an iteration (*MaxRule*). The best-so-far rule (*BestRule*) created by the colony is stored, so that the best rule from all the iterations can be returned once the algorithm has finished.

The rule pruning (line 9) is a simple method that removes the last term and recomputes the quality of the rule—this is the same pruning procedure proposed in [11]. If the rule is as good or better than the original rule then the shorter, more general rule is preferred. This process is repeated until the quality of the rule does not improve.

Finally, the pheromone update mechanism (line 19) also uses the same method as employed by Ant-Miner. There are two stages governing the pheromone updates. Stage one increases the pheromone levels of terms that appear in the best rule of the iteration. The pheromone increase of $term_{ij}$ (denoted as τ_{ij}) is proportional to the quality Q of the entire rule—where Q is constrained between 0 and 1 as can be seen in Equation 6—and it is given by:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t) \cdot Q \tag{1}$$

Stage two simulates the evaporation of pheromone in the graph. This is achieved by re-normalising the pheromone matrix. Every element in the matrix is divided by the sum of all elements in the matrix. While this affects terms that have just been increased as well as unused terms, it still promotes the former. This is due to unused terms who have not had their pheromone increased will be lowered by the normalisation, since the total pheromone value has increased, while used terms will increase their normalised pheromone levels compared to previous iterations [13].

3.2 Rule Quality Measure

Janssen and Fürnkranz [5] suggested a number of ways to calculate the errors in regression problems. Their SeCoReg implementation uses two metrics in conjunction. The first one is the Relative Root Mean Square Error (RRMSE), given by:

$$L_{RRMSE} = \frac{L_{RMSE}}{\sqrt{\frac{1}{m}L_{default}}} \tag{2}$$

where L_{RMSE} is the root mean square error and $L_{Default}$ is a normalising factor that will approximately bound the Relative Root Mean Square Error between 0 and 1. Both are defined in as:

$$L_{RMSE} = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - \bar{y_i})^2} \tag{3}$$

$$L_{default} = \sum_{i=1}^{m} (y_i - y')^2 \tag{4}$$

where m is the total number of instances in the dataset, y is the value of the current instance $\bar{y}$ is the predicted value of the current instance and finally y' is the mean over all instances.

RRSME attempts to normalise the RMSE between 0 and 1, however it is still possible to achieve values above 1 when the predicted values are worse than predicting the mean.

The second error metric is the relative coverage of a rule, which normalises the absolute coverage of the rule between 0 and 1—a rule with a value of 1 covers all the cases in the current dataset. The relative coverage is given by:

$$relCov = \frac{1}{m} \cdot coverage(Rule) \tag{5}$$

These two metrics are combined to produce a single value Q for each rule, given by:

[1]*c*Ant-Miner's rule construction procedure is essentially the same as Ant-Miner, with the difference that it includes a dynamic entropy-based discretisation procedure.

$$Q = \alpha \cdot (1 - L_{RRMSE}) + (1 - \alpha) \cdot relCov \quad (6)$$

where α sets the weighting between RRSME and relative coverage. An α value of 1 will only take into account the RRSME and a value of 0 will consider just the coverage.

3.3 Continuous Attribute Processing

Two different methods have been used for generating the split points for continuous attributes. The first method is the same one found in the SeCoReg algorithm (dubbed SSP) and the second one is an adaptation of M5's standard deviation to the context of regression rules.

3.3.1 SeCoReg Split Point Generation

The SeCoReg method performs supervised clustering of (attribute, target) pairs and produces n split points, where n is determined by the user. It achieves this by creating clusters that minimise the mean absolute errors of the target values in the following manner. First, it creates clusters containing a single pair and the instances are then sorted by attribute value. Next, the method searches either side of each cluster looking for the merger that increases the error by the minimum amount. This process is repeated until the number of clusters is reduced to $n + 1$.

The generated clusters have an upper and lower bound for the attribute values contained within. For each cluster, the mid point between its upper bound and the next's lower bound is calculated and these values are then returned as the n split points for an attribute. The split points are then combined with the $<$ and $\geq$ operators, each term is temporarily added to the rule being constructed and the quality measured. The (operator, split point) tuple that yields the highest quality is chosen as the new term and added to the rule.

3.3.2 Standard Deviation Split Point Generation

The second split point generation method is based on M5 regression tree algorithm. This method attempts to maximise the expected reduction in the error of the target value in a subset of instances. The expected error reduction is given by:

$$\Delta error = \sigma(T) - \sum_{i} \frac{|T_i|}{|T|} \cdot \sigma(T_i) \quad (7)$$

where T is the entire set of instances while T_i is the i-th subset and σ is the standard deviation of a subset. To find the optimal split point p for a continuous attribute att_c, the current covered instances are scanned from beginning to end (the entire instance set is scanned in the case that a continuous attribute is the first attribute selected to be added to the antecedent of a rule). Each split point generated two candidate subsets T: one subset containing instances that satisfy the condition $att_c < p$ and another containing instance that satisfy the condition $att_c \geq p$. Once the optimal p has been identified, the operator that is associated with the subset with the lowest standard error is the one used to create the rule term.

Note that the split point p is not stored in the construction graph nor used to update pheromone values, since the split point generation is a deterministic procedure—the same slit point p will be generated if att_c is selected using the same set of instances [10].

Table 1: Number of instances and attribute makeup of the 15 data sets used in the experiments.

		Attributes	
Name	Instances	Nominal	Continuous
Airfoil	1503	0	6
CCPP	9568	0	5
Concrete	1030	0	7
CPU	209	1	8
Efficiency	768	0	9
Flare	1065	10	1
Forest Fire	516	2	11
Housing	452	1	13
Istanbul	535	0	8
MPG	392	3	5
Red Wine	1599	0	12
Skill Craft	3337	0	20
Stock	950	0	10
WPBC_r	194	0	33
Yacht	308	0	7

Table 2: Parameters used for SeCoReg and Ant-Miner-Reg algorithms

General Parameter	Value
Minimum Covered Rule	10
Minimum Uncovered Theory	0.1
Split Points	3
Error Weighting	0.59
Cross Validation Folds	10
ACO Parameter	**Value**
Iterations	500
Colony Size	10

4. RESULTS

The experiments were carried out on 15 data sets, shown in Table 1, using tenfold cross-validation. The two Ant-Miner-Reg variants (using each of the different split point generation strategies) were run 5 times[2] and the average of the 5 runs was reported, as the algorithm is stochastic in nature and therefore its performance may vary; since SeCoReg is a deterministic algorithm, only a single run is required (one execution of the cross-validation). A one-time pre-processing step was undertaken, where instances with missing values were removed from the data set.

There are a number of parameters that need specifying for each algorithm. Parameters which are transferred from SeCoReg use the same values specified by the original authors [5]: the weighting α between coverage and error was set to 0.59 and the separate and conquer stopping criteria was set at 0.1 (fraction of instances that can remain uncovered). The ACO specific criteria of maximum number of iterations and colony size were set to 500 and 10 respectively, no tuning or optimization has been performed on these parameters. The

[2]A run corresponds to one execution of the cross-validation, which consists of executing the algorithm 10 times. Therefore, the algorithm was executed 50 times (5 times 10 cross-validation folds).

Table 3: RRMSE of the rule list produced by each of the algorithms on each data set. The bold cell signifies the smallest error from all three algorithms and the standard deviation is shown in brackets

Data set	Ant-Miner-Reg+M5SP	Ant-Miner-Reg+SSP	SeCoReg+SSP	SeCoReg+M5SP
Airfoil	**0.5512 [0.0138]**	0.7869 [0.0109]	0.9713 [0.0254]	0.9678 [0.0149]
CCPP	**0.3484 [0.0007]**	0.3592 [0.0005]	0.3622 [0.0012]	0.4041 [0.0022]
Concrete	**0.4182 [0.0046]**	0.4987 [0.0106]	0.4788 [0.0101]	0.6242 [0.0130]
CPU	0.5624 [0.0450]	**0.4469 [0.0289]**	0.8218 [0.0157]	0.8634 [0.0312]
Efficiency	**0.2038 [0.0006]**	0.2044 [0.0028]	0.2224 [0.0057]	0.2232 [0.0102]
Flare	1.0035 [0.0012]	1.0027 [0.0007]	**1.0021 [0.0087]**	**1.0021 [0.0087]**
Forest Fire	1.0340 [0.0417]	**1.0317 [0.0092]**	1.0488 [0.0387]	1.0431 [0.0243]
Housing	0.5547 [0.0354]	**0.4873 [0.0090]**	0.6324 [0.0054]	0.6392 [0.0087]
Istanbul	**0.8563 [0.0246]**	0.9287 [0.0175]	0.9035 [0.0084]	0.9601 [0.0178]
MPG	0.5432 [0.0149]	**0.5322 [0.0083]**	0.5673 [0.0009]	0.5731 [0.0021]
Red wine	**0.9048 [0.0216]**	0.9161 [0.0114]	0.9800 [0.0186]	0.9851 [0.0203]
Skill Craft	**0.8219 [0.0209]**	0.8324 [0.0267]	0.8463 [0.0099]	0.8735 [0.0092]
Stock	**0.2457 [0.0106]**	0.2540 [0.0081]	0.2886 [0.0152]	0.3301 [0.0129]
WPBC_r	1.3549 [0.0494]	1.3265 [0.0375]	**1.0738 [0.0474]**	1.1035 [0.0381]
Yacht	0.5273 [0.0015]	0.3576 [0.0050]	**0.3448 [0.0067]**	0.5260 [0.0140]

Table 4: Wilcoxon signed-rank test results of the RRSME error when comparing Ant-Miner-Reg against SeCoReg with both the SeCoReg split point processing and M5 split points processing. Significant differences at the $\alpha = 0.10$ are shown in bold.

Algorithm Pairings		p
Ant-Miner-Reg+M5SP	SeCoReg+M5SP	**0.0103**
Ant-Miner-Reg+SSP	SeCoReg+M5SP	**0.0164**
Ant-Miner-Reg+M5SP	SeCoReg+SSP	**0.0637**
Ant-Miner-Reg+SSP	SeCoReg+SSP	0.1354

full list of parameters for both algorithms can be found in Table 2. It should be noted that the M5 split point generation method ignores the number of split points requested and always returns a single one.

Table 3 shows the RRMSE of each algorithm, the first column contains the error produced by models generated by Ant-Miner-Reg with M5 Rules split point generation (Ant-Miner-Reg+M5SP); column 2 contains Ant-Miner-Reg with SeCoReg split point generation (Ant-Miner-Reg+SSP), finally the last two columns show the error produced by the models generated by both variants of the SeCoReg algorithm, the original SeCoReg (SeCoReg+SSP) and SeCoReg with M5 split point generation (SeCoReg+M5SP). The lowest value in each row (best result) is shown in bold.

As can be seen in Table 3, the Ant-Miner-Reg variants outperform SeCoReg variants in 12 of the 15 data sets. Also of note is the Ant-Miner-Reg+M5SP variant outperforms the Ant-Miner-Reg+SSP variant in 8 of the 12 data sets won by Ant-Miner-Reg. The results from all four algorithms have also been checked for statistical significance using the Wilcoxon signed-rank test, the pairings that compare similar algorithms have been ignored leaving four parings between Ant-Miner-Reg and SeCoReg, which can be found in Table 4. The decision to use the Wilcoxon signed-rank-test was made as we are interested in the comparison of the Ant-Miner-Reg and SeCoReg pairs and not the interaction between different variants of the same base algorithm.

Additionally, the average number of attributes-conditions (terms) that are evaluated in the model in order to classify instances in the test data was profiled, the results of this can be seen in Table 5. Rule lists that have very general rules will perform better as the majority of instances will require few rules. The data in Table 5 was generated by counting the number of terms of each rule used to classify an instance; this total was then divided by the number of instances classified to give the average number of terms required for classification—this measure is called *prediction-explanation size* [9].

Finally, Table 6 shows the significance of the average term usage when comparing the Ant-Miner-Reg to SeCoReg using the Wilcoxon signed-rank test.

5. DISCUSSION

The Ant-Miner-Reg+SSP variant showed an improvement in RRSME compared to the reference SeCoReg implementation also using the original split point processing method, beating it in 12 of the 15 data sets; however the results were not significantly significant according to the Wilcoxon signed-rank test. The results do show promise as the ACO implementation was not tuned before hand and instead used the parameters optimised by the original SeCoReg authors.

Table 5: Average number of attribute-conditions (terms) involved in the classification of an instance. The smallest number of terms is shown in bold while the standard deviation is shown in brackets.

Data set	Ant-Miner-Reg+M5SP	Ant-Miner-Reg+SSP	SeCoReg+SSP	SeCoReg+M5SP
Airfoil	157.14 [8.00]	55.83 [3.74]	**1.97 [0.01]**	2.11 [0.02]
CCPP	24.54 [0.53]	23.09 [1.56]	**13.13 [5.44]**	14.54 [4.23]
Concrete	**191.07 [1.83]**	196.69 [4.16]	212.23 [36.74]	202.30 [24.43]
CPU	**2.99 [0.13]**	5.89 [0.11]	3.90 [0.24]	4.21 [0.54]
Efficiency	7.34 [0.33]	**6.97 [0.07]**	7.08 [1.31	6.82 [0.89]
Flare	28.71 [0.77]	**28.49 [1.45]**	31.76 [5.88]	29.93 [2.78]
Forest Fire	**56.87 [2.13]**	84.64 [2.04]	63.50 [31.53]	61.46 [14.32]
Housing	**24.26 [7.89]**	104.70 [5.20]	45.24 [22.28]	48.39 [26.75]
Istanbul	101.31 [5.96]	29.98 [3.83]	**20.57 [11.69]**	30.62 [9.13]
MPG	15.42 [0.75]	18.09 [0.42]	**14.12 [3.41]**	22.45 [5.46]
Red Wine	341.66 [4.46]	311.26 [12.91]	**154.51 [23.57]**	176.87 [54.73]
Skill Craft	1317.85 [22.28]	1422.58 [29.54]	**1022.04 [140.92]**	1134.75 [121.34]
Stock	**21.07 [1.21]**	24.49 [3.48]	29.62 [11.28]	32.74 [8.35]
WPBC_r	**66.31 [3.71]**	103.70 [3.08]	87.36 [76.44]	84.21 [62.05]
Yacht	**1.22 [0.08]**	1.25 [0.01]	1.26 [0.12]	1.29 [0.17]

Table 6: Wilcoxon signed-rank test results for the average number of terms used to classify an instance for the models produced by each algorithm. Significant differences at the $\alpha = 0.10$ are shown in bold.

Algorithm Pairings		p
Ant-Miner-Reg+M5SP	SeCoReg+SSP	0.6788
Ant-Miner-Reg+SSP	SeCoReg+SSP	**0.0302**
Ant-Miner-Reg+M5SP	SeCoReg+M5SP	0.8904
Ant-Miner-Reg+SSP	SeCoReg+M5SP	**0.0833**

The switch to a M5 split point generation method improved the performance of Ant-Miner-Reg, outperforming the SeCoReg reference implementation in the same 12 data sets; the improvement gained by ACO rule construction procedure and the M5 split point generation showed that these results are significant at the 10% level with a p value of 0.0637. These results show that the ACO made a measurable improvement to the performance of a sequential covering algorithm, as this is an initial implementation further work may lead to greater performance. The improvement shown by the ACO may be due to its ability to choose (initially) poor terms for rules that will become good choices when combined with later terms, while the greedy strategy will always choose the best term at each decision point. The M5 split point generation procedure was unable to improve the performance of SeCoReg's greedy search strategy, this may be due to the reduction in the available terms as the M5 split point generation procedure returns a single split point, while the original split point generation returns multiple candidate points.

Secondly the average number of terms to classify each instance was measured. It was found that the best Ant-Miner-Reg+M5SP variant did not increase the average number of terms required by a significant amount when compared with both SeCoReg at the 10% level. Ant-Miner-Reg+SSP was found to significantly increase the number of terms required to classify an instance when compared to both SeCoReg variants.

6. CONCLUSION

This paper presented a ACO-based regression algorithm, called Ant-Miner-Reg, which generates comprehensible regression rules. The proposed algorithm significantly outperformed the reference implementation SeCoReg without increasing the number of terms required to classify an instance. Overall, we regard these results as positive, as it shows the benefits of using a global search technique in the form of an ACO to search for the best rule at each iteration of the sequential covering. Optimisation of Ant-Miner-Reg's parameters is required, as currently it uses the same values as the reference implementation—these may not be optimal for the proposed Ant-Miner-Reg algorithm and could improve the results presented here further.

Further investigation is required to realise the full potential of the ACO search used to create regression rules. This could include the adoption of a better continuous attribute processing technique, which enables the optimisation of the numeric values chosen by fully integrating them inside the pheromone matrix. The adoption of linear models as leaf nodes in a similar fashion to M5'Rules might improve the error of the algorithm further, although this needs to be balanced by the loss of comprehensibility. Enabling ants to create entire rule lists instead of individual rules (as proposed in [12]), allowing rule interaction to influence the pheromone matrix, is another interesting direction. Additionally, Ant-Miner-Reg takes no advantage of available heuristic information, which can be used to increase rule quality.

7. ACKNOWLEDGEMENTS

The authors would like to thank Dr. Frederik Janssen for providing support with the SeCoReg algorithm.

8. REFERENCES

[1] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. A Bradford Book, 2004.

[2] L. Fahrmeir, T. Kneib, S. Lang, and B. Marx. *Regression: Models, Methods and Applications*. Springer, 2013.

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

[4] G. Holmes, M. Hall, and E. Frank. Generating rule sets from model trees. In *Proceedings 12th Australian Joint Conference on Artificial Intelligence*, pages 1–12. Springer, 1999.

[5] F. Janssen and J. Fürnkranz. Seperate-and-conquer regression. In *Proceedings of the German Workshop on Lernen*, pages 81–89, 2010.

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

[7] B. Minnaert and D. Martens. Towards a particle swarm optimization-based regression rule miner. In *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*, pages 961–963, 2012.

[8] T. Mitchell. *Machine Learning*. McGraw-Hill Science, 1997.

[9] F. Otero and A. Freitas. Improving the interpretability of classification rules discovered by an ant colony algorithm. In *2013 Genetic and Evolutionary Computation Conference (GECCO 13)*, 2013.

[10] 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 (Proc. ANTS 2008)*, 2008.

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

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

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

[14] J. Quinlan. Learning with continuous classes. In *Proceedings 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348. World Scientific, 1992.

[15] S. Shevade, S. Keerthi, C. Bhattacharyya, and K. Murthy. Improvements to the smo algorithm for svm regression. *IEEE Transactions on Neural Networks*, 11(5):1188–1193, 1999.