An Automatic Programming ACO-Based Algorithm for Classification Rule Mining

J.L. Olmo, J.M. Luna, J.R. Romero, S. Ventura

Abstract In this paper we present a novel algorithm, named GBAP, that jointly uses automatic programming with ant colony optimization for mining classification rules. GBAP is based on a context-free grammar that properly guides the search process of valid rules. Furthermore, its most important characteristics are also discussed, such as the use of two different heuristic measures for every transition rule, as well as the way it evaluates the mined rules. These features enhance the final rule compilation from the output classifier. Finally, the experiments over 17 diverse data sets prove that the accuracy values obtained by GBAP are pretty competitive and even better than those resulting from the top Ant-Miner algorithm.

1 Introduction

Data Mining (DM) entails the process of applying specific algorithms for extracting comprehensible, non-trivial and useful knowledge from data. The DM classification task aims to obtain a set of classification rules (a classifier) from a training data set. Once the classifier is built, one can apply these rules to other uncategorized data in order to label each instance with one of the predefined classes. The performance of the classifier is typically measured with the accuracy obtained when applying the classifier to a separate test set.

Support vector machines and neural networks have demonstrated to be accurate solutions to build classifiers. However, they have the disadvantage of generating non-linear classifiers. In constrast, logic based algorithms (i.e., decision trees and rule-based classifiers) provide more interpretability, but they are not as accurate as the previous methods [7].

Dept. of Computer Science and Numerical Analysis, Rabanales Campus, Albert Einstein building, 14071 Cordoba, Spain e-mail: {juanluisolmo, i32luarj, jrromero, sventura}@uco.es

Ant Colony Optimization (ACO) [4] is a nature-inspired optimization metaheuristic based in the behavior and organization of ant colonies in their search for food. Ant algorithms have been successfully applied to a broad range of domains, including the extraction of classification rules in DM. For example, Ant-Miner, originally proposed by Parpinelli and colleagues [10], was the first algorithm based in ACO applied to the classification task. Ant-Miner follows a sequential-covering approach and has become a top algorithm in this field.

Furthermore, automatic programming is a method that uses search techniques to construct automatically a program that solves a given problem automatically, without requiring the user to know the structure of the solution. In fact, the problem is solved by simply specifying the goals to be reached. Typical examples of this method are Genetic Programming (GP) [2, 8] and Ant Programming (AP) [11], which uses ACO as search technique. The former has demonstrated that is capable to provide good performance for the design of classifiers, but the latter has never been used to tackle classification problems.

In this work we explore the application of an AP algorithm for classification rule mining. Thus, our proposal generates a rule-based classifier, which is composed of a set of classification rules that take the form *IF* <antecedent> *THEN* <consequent>. Our algorithm aims to construct accurate but also comprehensible classifiers, and first results show that it achieves good performance in terms of accuracy.

The remainder of the paper is organized as follows. In the next section we describe the proposed algorithm. Section 3 explains the experiments carried out and the data sets employed. In Section 4 we discuss the results obtained. Finally, some concluding remarks and ideas for future work are provided in Section 5.

2 The GBAP algorithm

In this section we introduce Grammar Based Ant Programming (GBAP) algorithm.

Roughly speaking, GBAP follows a grammar guided automatic programming approach. This kind of systems are restricted by a defined grammar to ensure that any solution found is syntactically valid. The goal of GBAP is to obtain a classifier for a given data set, instead of a generic solution that could be applied to other data sets. This classifier takes the form of a decision list where discovered rules are sorted in descending order by fitness, and the bottom rule added to the classifier, which corresponds to the majority class in the data set, acts as default rule.

In ACO-based algorithms there must be an environment where ants cooperate each other. In GBAP, this environment is defined by the search space comprising all the possible expressions or programs that can be derived from the grammar. The space of states adopts the form of a derivation tree, and the path followed by the artificial ant could be seen as the sequence of derivation steps that leads the ant to a final state or solution.

In next sections we explain how classification rules are represented, presenting also a detailed pseudocode and description of the main characteristics of GBAP.

2.1 Rules encoding

GBAP follows the *ant=rule* (i.e., *individual=rule*) approach [5]. Notice that once the ant is created, it just represents the antecedent of the new rule. In Section 2.2 we will analyze how the consequent is properly assigned to the rule.

GBAP prescribes a context-free grammar for the representation of the individuals, as shown in Figure 1. Notice that this grammar is expressed in prefix notation and should be always derived from the left. It implies that each transition from a state i to another state j is triggered after applying a production rule to the first non-terminal symbol of the state i. This design decision was taken because of performance reasons, in order to expedite the calculations neccessary to compute the rule fitness.

V = { <EXP>, <COND> }
Σ = { AND, =, !=, $attr_1$, $attr_2$, ..., $attr_n$,
$value_{11}$, $value_{12}$, ..., $value_{1m}$,
$value_{21}$, $value_{22}$, ..., $value_{2m}$,
..., $value_{n1}$, $value_{n2}$, ..., $value_{nm}$ }
R = { <S> := <EXP>,
<EXP> := AND <EXP> <COND> |
<COND>
<COND> := all possible valid combinations of the
ternary operator-attribute-value }
S = { <S> }

Fig. 1 Context-free grammar used in GBAP, defined by $G = (V, \Sigma, R, S)$. Notice that any production rule consists of a left hand side (LHS) and a right hand side (RHS). The LHS always refers to a non-terminal symbol that might be replaced by the RHS of the rule (composed of a combination of terminal and non-terminal symbols).

2.2 Pseudocode and main characteristics of GBAP

An important characteristic of GBAP is the incremental generation of the space of states. In fact, depending on both the problem addressed and the number of derivations from the grammar permitted, it may be unfeasible to keep in memory the whole space of states. We therefore follow an incremental build approach in which as the ants are created we store the states they visit. This requires that each ant stores first the followed path. For this reason, the initial space of states is empty and all the possible transitions have the same amount of pheromones.

Another important characteristic of the algorithm proposed is that it considers two complementary heuristic measures. A first one is the *cardinality of the production rules (P_{card})*, and it is used due to the shape adopted by the space of states. This measure increases the probability of choosing transitions that lead to a greater number of solutions, and it is based on the cardinality measure proposed in [6]. When initializing the grammar in the algorithm, a cardinality table for the maximum num-

Algorithm 1 High level pseudocode of GBAP

Require: $numberOfGenerations, numberOfAnts$
1: Initialize space of states, starting up the grammar
2: Create the classifier
3: **for** $i = 0$ **to** $i = numberOfGenerations$ **do**
4: Create list $ants \leftarrow \{\}$
5: **for** $j = 0$ **to** $j = numberOfAnts$ **do**
6: $ant \leftarrow$ Create new ant
7: Store *ant*'s path states in the space of states
8: Evaluate *ant*, computing its fitness for each available class in the data set
9: Add *ant* to the list *ants*
10: **end for**
11: Do niching algorithm, assigning the consequent to the ants and establishing the classifier rules
12: **for** each *ant* in *ants* **do**
13: **if** $fitness > threshold$ **then**
14: Update pheromone rate in the path followed by *ant* proportionally to its fitness
15: **end if**
16: **end for**
17: Evaporate the pheromone rate along the whole space of states
18: Normalize values of pheromones
19: **end for**
20: Establish the default rule in the classifier
21: $predictiveAccuracy \leftarrow$ Compute the predictive accuracy obtained by the classifier when running over the test set
22: **return** *predictiveAccuracy*

ber of derivations allowed is computed per each production rule. Given a state i and all its possible subsequent states, the value of this heuristic for each possible transition is defined as the ratio between the number of solutions that can be successfully reached if the ant goes to the destination state applying this transition, and the number of all possible solutions that can be reached from the source state. Notice that this heuristic measure is only taken into account for intermediate transitions.

A second one is the *information gain* ($G(A_i)$). It is only used in transitions involving the application of production rules that imply the selection of attributes of the problem domain (i.e., $< COND >:= operator - attribute - value$). This measure is similar to the one used by the Ant-Miner algorithm.

The use of both heuristic measures affects the creation process of new ants, when they move to the next state of their path. The transition rule will assign a probability to each available next state. In case of derivations that are not able to reach any final state in a number of steps less than or equal to the maximum number of derivations remaining, a probability equal to zero will be assigned and, in consequence, the ant will not select such a movement.

The probability that a given ant moves from a state i to another valid state j is defined by the following equation:

$$P_{ij} = \frac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\Sigma_{i=0}^{j} \eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}} \tag{1}$$

where α is the heuristic exponent, β is the pheromones exponent and η is computed as $G(A_i) + P_{card}$ (at least one of the two components must be equal to zero).

The fitness function that GBAP uses in the training stage for measuring the quality of the ants is the Laplace accuracy [3], which is defined as:

$$fitness = LaplaceAccuracy = \frac{1+TP}{k+TP+FP} \tag{2}$$

where TP and FP stands for true positives and false positives, respectively, and k refers to the number of classes in the data set.

Concerning the assignment of the consequent, GBAP follows a niching approach analogous to that employed in [1], whose purpose is to evolve different multiple rules for predicting each class in the data set while preserving the diversity. Depending on each dataset and in the distribution of the instances by class, it is often not possible for a rule to cover all instances of a class and therefore it is necessary to discover additional rules for predicting this class. The niching algorithm takes care of it but it does not overlap with instances of another class. In addition, it is appropriate when removing redundant rules.

In the niching algorithm developed every instance in a data set is called a token, for which all ants in the colony will compete to capture. First of all GBAP computes an array of k fitness values per individual, one for each class (assuming that the respective class is assigned as consequent to the individual). Then, the following steps are repeated for each class: first, the ants are sorted by their respectively class fitness in descending order. Second, each ant tries to take as many tokens as it covers in case of tokens that belong to the computing class and also if the token has not been seized by other ant previously. Finally, the ant's adjusted fitness for this class is computed as:

$$adjustedFitness = fitness \cdot \frac{numberOfCapturedTokens}{numberOfClassTokens} \tag{3}$$

Once the k adjusted fitnesses have been calculated, the consequent assigned to each ant corresponds to the one that reports the best adjusted fitness. To conclude, individuals that have an adjusted fitness greater than zero –and consequently cover at least one instance of the train set– are added to the classifier.

Finally, regarding the pheromone update, if the quality of an ant is greater than a threshold value, then a delayed pheromone update over the path of this ant takes place. The threshold value has been fixed to 0.5 with the aim that bad solutions will never incluence the environment. The reinforcement is based on the quality of the solution encoded by the ant:

$$\tau_{ij}(t+1) = \tau_{ij}(1-\rho) + \tau_{ij} \cdot Q \cdot fitness \tag{4}$$

where τ represents the amount of pheromones in the transition from the state i to the state j; ρ, the evaporation rate; and Q is the parameter that permits to vary the influence of the reinforcement.

3 Data sets and preprocessing

This section describes the data sets used for the experimentation, as well as the preprocessing steps performed.

GBAP algorithm has been tested with many diverse data sets from the well-known UCI[1] machine learning repository. In fact, the data sets selected for the experiments present varied dimensionality, and some of them include missing values, while other do not, as shown in Table 1.

During the preprocessing stage we performed the following two actions using the Weka Machine Learning library[2]. Firstly, data sets comprising missing values were preprocessed replacing these values with the mode (in case of nominal attributes) and the arithmetic mean (in case of numerical attributes) of the entire data set [9]. Secondly, data sets with numerical attributes were properly discretized in order to only deal with categorical attributes [12].

Regarding the algorithm evaluation, we applied a stratified 10-fold cross-validation, so that the prediction performance is considered as the average accuracy over

Table 1 Data sets description

DATASET	MISSING VALUES	INSTANCES	ATTRIBUTES			CLASSES
			Continuous	*Binary*	*Nominal*	
Hepatitis	yes	155	6	13	0	2
Sonar	no	208	60	0	0	2
Breast-c	yes	286	0	3	6	2
Heart-c	yes	303	6	3	4	2
Ionosphere	no	351	33	1	0	2
Horse-c	yes	368	7	2	13	2
Breast-w	yes	699	9	0	0	2
Diabetes	no	768	0	8	0	2
Credit-g	no	1000	6	3	11	2
Mushroom	yes	8124	0	0	22	2
Iris	no	150	4	0	0	3
Wine	no	178	13	0	0	3
Balance-scale	no	625	4	0	0	3
Lymphography	no	148	3	9	6	4
Glass	no	214	9	0	0	6
Zoo	no	101	1	15	0	7
Primary-tumor	yes	339	0	14	3	21

[1] All data sets can be reached from the UCI website at http://archive.ics.uci.edu/ml/datasets.html

[2] The Weka library is publicly available at http://www.cs.waikato.ac.nz/ml/index.html

Table 2 Predictive accuracy(%) comparative results

Dataset	GBAP	Ant-Miner
Hepatitis	**82,17**	79,17
Sonar	**81,98**	74,70
Breast-c	71,40	**73,12**
Heart-c	**82,84**	76,62
Ionosphere	**93,02**	87,41
Horse-c	82,97	**84,38**
Breast-w	**96,50**	92,13
Diabetes	**75,80**	72,84
Credit-g	**70,79**	70,36
Mushroom	**98,26**	97,06
Iris	**96,67**	95,33
Wine	**97,01**	92,08
Balance-scale	**75,49**	68,09
Lymphography	**81,00**	77,69
Glass	**69,13**	64,43
Zoo	**95,60**	83,85
Primary	**37,91**	35,30

these 10 folds. In the stratified 10-fold cross-validation the entire data set is splitted into 10 mutually exclusive partitions, P_1,...,P_k, containing approximately the same number of patterns, and the same proportion of classes than the original data set. Then, 10 different experiments were executed using $\bigcup_{j \neq i} P_j$ as the training set at the ith-experiment and P_i as the test set.

4 Results

Experiments compare the performance of GBAP against Ant-Miner, in terms of predictive accuracy, over the data sets listed in Table 1. More specifically, Ant-Miner was used in all executions –10 per dataset– with its default parameters. For the GBAP algorithm its configuration parameters were set to: *number of ants* = 20, *number of generations* = 100, *max number of derivations* = 15, *initial pheromone amount* = 1.0, *evaporation rate* = 0.05, *min pheromone amount* = 0.1, Q = 1.0, *alpha* = 0.4, and *betha* = 1.0.

Table 2 summarizes the results obtained, where each row shows the data set tested and the resulting average accuracy (in %) for both algorithms. Best results per algorithm and dataset are highlighted in bold typeface.

As can be seen, GBAP obtains best results in approximately 88% of cases. We also analyzed the statistic significance of the obtained results applying the non-parametric Wilcoxon pair-test, and the results ($z = 3.195$, $p < 0.001$) proved that there are significant differences with a probability of 99%, where GBAP is significantly more accurate than Ant-Miner.

5 Conclusions and future work

In this paper we presented a novel automatic programming algorithm based in ACO restricted by the use of a context-free grammar for mining classification rules from diverse data sets. The proposal is supported by a two-sided heuristic function that guides the search process of the valid solutions, as well as the chance of modifying the complexity of rules mined by simply varying the number of derivations allowed for the grammar.

In this work we have compared the performance of GBAP against Ant-Miner in 17 different data sets publicly available. The obtained results prove that the former is significantly more accurate than the latter. As future work we plan to apply other problem-dependent heuristic measures. We also will explore the consideration of adding new functionality to deal with continuous attributes and adapting the current algorithm to the multi-objective approach.

Acknowledgments. This work has been supported by the Regional Government of Andalucia and Ministry of Science and Technology, projects TIC-3720 and TIN2008-06681-C06-03.

References

1. J. Ávila, E. Gibaja, A. Zafra, and S. Ventura. A niching algorithm to learn discriminant functions with multi-label patterns. In *Intelligent Data Engineering and Automated Learning - IDEAL 2009*, pages 570–577. 2009.
2. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming - An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, January 1998.
3. Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In *EWSL-91*, pages 151–163. Springer-Verlag, 1991.
4. M. Dorigo and C. Blum. Ant colony optimization theory: a survey. *Theoretical Computer Science*, 344:243–278, 2005.
5. Pedro G. Espejo, Sebastián Ventura, and Francisco Herrera. A survey on the application of genetic programming to classification. *IEEE Transactions on System, Man and Cybernetics Part C*, Article in press, 2008.
6. Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness*. Physica-Verlag, Heidelberg, 1995.
7. S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Reviews*, 26:159–190, 2006.
8. J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. The MIT Press, Cambridge, MA, 1992.
9. D. T. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. Wiley, 2005.
10. R. Parpinelli, A. A. Freitas, and H. S. Lopes. Data mining with an ant colony optimization algorithm. *IEEE Trans on Evolutionary Computation*, 6:321–332, 2002.
11. O. Roux and C. Fonlupt. Ant programming: or how to ants for automatic programming. In M. Dorigo and Et Al, editors, *ANTS'2000*, pages 121–129, 2000.
12. I. H. Witten and E. Frank. *Data Mining Practical Machine Learning Tools And Techniques*. Morgan Kauffman, 2005.