A Grammar Based Ant Programming Algorithm for Mining Classification Rules

Juan Luis Olmo, José Raúl Romero and Sebastián Ventura

Abstract— **This paper focuses on the application of a new ACO-based automatic programming algorithm to the classification task of data mining. This new model, called GBAP algorithm, is based on a context-free grammar that properly guides the creation of new valid individuals. Moreover, its most differentiating factors, such as the use of two complementary heuristic measures for every transition rule, as well as the way it assigns a consequent and evaluates the extracted rules, are also discussed. These features enhance the final rule compilation from the output classifier. The performance of the proposed algorithm is evaluated and compared against other top algorithms, and the results obtained over 17 diverse data sets show that our approach reaches pretty competitive and even better accuracy values than those resulting from the other algorithms considered in the experimentation.**

I. Introduction

Data Mining (DM) involves the process of applying specific algorithms for extracting comprehensible, non-trivial and useful knowledge from data. The aim of the DM classification task is, given a set of labeled examples (the training set), to generate a classifier that takes into account the hidden relations between the values of the attributes and the classes, in such a way that this model can be applied later to novel and uncategorized data to label each instance with one of the predefined targets.

Ant Colony Optimization (ACO) [1] is a nature-inspired optimization metaheuristic based on 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 et al. [2], was the first algorithm based on 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 build automatically a program that solves a given problem, 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) [3], [4] and Ant Programming (AP) [5], which uses ACO as its search technique. The former has demonstrated that it is capable to provide good performance for the design of classifiers, but the latter has never been explored to tackle classification problems.

J.L. Olmo (*Student Member, IEEE*), J.R. Romero (*Member, IEEE*) and S. Ventura (*Senior Member, IEEE*) are with the Department of Computer Science and Numerical Analysis, University of Cordoba, Rabanales Campus, Albert Einstein building, 14071 Cordoba, Spain. Email: {juanluisolmo,jrromero,sventura}@uco.es

We believe that the development of AP algorithms for data mining is still an unexplored and promising research area. Hence, in this work we explore the application of an AP algorithm for mining classification rules, which takes advantage of the inherent benefits of both ACO metaheuristic and automatic programming. In addition, the search process is guided by a context-free grammar, which also provides flexibility to apply the developed algorithm to a variety of problems with minor changes. Our proposal generates a rule-based classifier, that aims to construct not only accurate but also comprehensible classifiers. First results comparing with other well known algorithms are promising and show that our approach achieves good performance in terms of accuracy.

The remainder of the paper is organized as follows. In the next section we introduce ACO as the primary technique of our algorithm, as well as AP. In Section III we describe the proposed algorithm. Section IV explains the experiments carried out and the data sets used. In Section V we discuss the results obtained and, finally, some concluding remarks and ideas for future research works are outlined in Section VI.

II. Related work

This section focuses primarily on explaining the functioning of Ant-Miner, which is the most referenced ACO algorithm employed for classification in data mining, and which we also use to compare the results of our algorithm. Likewise this section presents PSO/ACO2, the other field algorithm employed in the experimentation. Secondly, the section provides a review of the various AP algorithms published in the literature so far.

A. Ant Colony Optimization

ACO is a metaheuristic, nature-inspired optimization placed into Swarm Intelligence [6], which studies the collective behavior of simple agents, e.g., flock of birds, fish schools, colonies of bacteria or amoeba, or groups of insects living in colonies, such as bees, wasps or ants. Specifically, ACO bases the design of intelligent multi-agent systems on the behavior and organization of ant colonies in their search for food, where ants communicate between themselves in an indirect way by means of a chemical substance (pheromone) that they spread throughout the environment. The higher the pheromone concentration is in a path, the higher is the probability that a given ant will follow this path.

ACO algorithms have been successfully employed to obtain approximate solutions to optimization problems. They also have been carried out on a broad range of applications, including the classification task. Ant-Miner [2] has become

the most referenced ant-based algorithm in this field. It follows a sequential-covering approach in which, starting from a training set and an empty set of rules, it finds new rules to be added to the set of discovered rules. Instances of the training set covered by each new rule are removed, reducing the size of the training set. Ant-Miner continues then discovering new rules either until the training set is empty or until the number of discovered rules exceeds the maximum number of rules allowed. The quality of the rules is computed by the following formula:

$$\begin{aligned} fitness =& sensitivity \cdot specificity \\ =& \frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP} \end{aligned} \quad (1)$$

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

Some modifications of Ant-Miner entail the use of different mechanisms for pruning, pheromone updating, heuristic function, or they are designed for including interval rules, dealing with continuous attributes, extracting fuzzy classification rules or being applied to multi-label classification. Furthermore, some extensions propose the hybridization of ACO with other metaheuristics. Among them, we appreciate the PSO/ACO2 algorithm developed by Holden et al. [7], which is a hybrid Particle Swarm Optimization (PSO) - ACO algorithm for the discovery of classification rules. PSO is another branch of Swarm Intelligence, that consists in an optimization technique inspired from the motion of birds in flocks. PSO/ACO2 algorithm can deal both with numerical and nominal attributes, and it follows a sequential-covering approach, as Ant-Miner does. Another difference lies in the fitness function, which in this case is given by the Laplace-corrected precision:

$$fitness = \frac{1+TP}{1+TP+FP} \quad (2)$$

B. Ant Programming

AP is an automatic programming method that uses ACO as the search technique for constructing computer programs. Hereafter we review the different AP proposals investigated so far that have been applied to approximation or symbolic regression problems.

The first work that applied the ants paradigm to the automatic generation of programs was presented by Roux and Fonlupt [5]. In fact, their algorithm starts creating a random population of programs (trees) and storing a table of pheromones for each node of the tree. Each pheromone table holds the amount of pheromone associated with all possible elements (named terminals and functions). Then, each program is evaluated and the pheromone table is updated by evaporation and reinforcement based on the quality of solutions. These steps are repeated until some criteria is satisfied, but notice that new populations of programs are generated according to the pheromone tables (the higher the rate is, the higher is the probability to be chosen). This approach was used to solve symbolic regression problems and a multiplexer problem with relative success.

Boryczka and Czech [8] also applied AP for solving symbolic regression problems, calling their method Ant Colony Programming (ACP). They proposed two different versions of ACP, known as expression approach and program approach. In the expression approach the system generates arithmetic expressions in prefix notation from the path followed by the ant in a graph. This graph is defined as $G = (N, E)$ where N is the set of nodes, which can represent either a variable or an operator, and E is the set of edges, each one with a pheromone value associated. Green et al. [9] also presented an AP technique similar to the ACP expression approach. In turn, in the program or instruction approach the nodes in the graph represent assignment instructions, and the solution consists of a sequence of assignments which evaluates the function.

In [10], Boryczka extended the previous works in ACP with the aim of reducing the computational time neccessary for the evaluation of transition rules.

Another attempt to evolve programs using the ACO algorithm was AntTAG [11]. It was proposed by Abbass et al. as a method of automatic programming employing ACO as its search strategy and a Tree-Adjunct Grammar (TAG) to build programs. The authors tested its performance on symbolic regression problems and achieved better performance than Grammar Guided Genetic Programming and Tree Adjunct Grammar Guided Genetic Programming.

Keber and Schuster published another grammar-based work called Generalized Ant Programming (GAP) [12], which uses a context-free grammar instead of TAG. Salehi-Abari and White [13] worked on GAP, proposing a variation of the algorithm called Enhanced Generalized Ant Programming (EGAP). More specifically, it introduces a new pheromone placement method that tends to put in a derivation step an amount of pheromone proportional to the depth of the path; and it also employs a specific heuristic function to control the path termination.

More recently, Shirakawa et al. [14] proposed Dynamic Ant Programming (DAP). Its main difference with respect to ACP lies in the use of a dynamically changing pheromone table and a variable number of nodes, which lead to a more compact space of states. The authors only compared the performance of DAP against GP using symbolic regression problems.

III. The GBAP algorithm

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

Roughly speaking, GBAP follows a grammar guided automatic programming perspective. In this kind of systems there is a grammar that restricts the space of states and ensures that any solution found is syntactically valid. In fact, any state and each feasible solution –which corresponds to leaf states– in the environment can be reached from the initial state of the grammar in a sequence of steps by applying the production rules available.

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. Notice that GBAP was originally designed for the DM classification task. However, it can be modified and adapted to any other problem that can be solved using automatic programming, by simply reconfiguring the way the algorithm evaluates the individuals and designing a suitable grammar for the problem to be solved.

As can be seen in the following sections, the GBAP algorithm can not be fitted into a typical ACO system. The closest algorithm may be the Max-Min Ant System (MMAS) [15], with which GBAP shares the idea of pheromone bounds and also initializes pheromone trails to the maximum value, as MMAS does; however, instead of updating the trails just with the best ant, in GBAP all the current generation ants whose fitness is over an established threshold reinforce the trails.

Next we present a description of the main characteristics of GBAP algorithm and a detailed pseudocode.

A. Environment and individual representation

GBAP prescribes a context-free grammar for representing individuals expressed in Backus-Naur Form (BNF) notation, and defined by $G = (V, \Sigma, R, S)$, as shown in Figure 1. 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). Production rules are 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, =, !=, attr1, attr2, ..., attrn,
           value11, value12, ..., value1m,
           value21, value22, ..., value2m,
       ..., valuen1, valuen2, ..., valuenm}
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

Notice that grammar guided systems use the terminal and non-terminal nomenclature, but it refers to the symbols of the grammar, rather than to the leaf nodes and function/internal nodes of an individual tree representation, respectively.

The first aspect in the design of an ACO-based algorithm is the specification of an environment where ants cooperate with each other. In GBAP the environment is defined as the search space comprising all the possible expressions or programs that can be derived from the grammar. Then, the environment adopts the form of a derivation tree, as shown in Figure 2 at a depth of 4 derivations.

The initial state of the environment simply corresponds to the starting symbol of the grammar. From this, each ant tries to build a feasible solution to the problem. Any solution found takes the form of a path over the derivation tree, as shown in the sample coloured path in Figure 2. This path consists of a sequence of states, where each derivation step is given by applying one of the available production rules at that point. The last state of the path is a final state or solution (a state that only contains terminal symbols, whereas the intermediate states of the path consist of a combination of terminal and non-terminal symbols, with at least one non-terminal symbol). Each ant stores the path explored, but observe that only the last state represents the evaluable expression of the solution encoded.

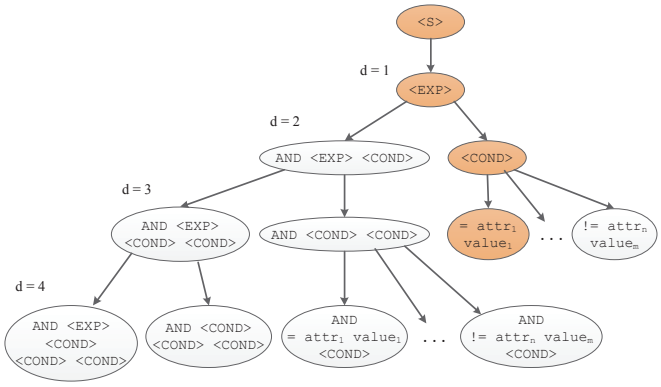

Fig. 2. Representation of the space of states with a sample coloured path

An important characteristic of GBAP is the incremental generation of the environment. In fact, depending on both the problem addressed and the number of derivations permitted from the grammar, it may be unfeasible to keep in memory the whole space of states. We therefore follow an incremental build approach in which once each ant is created and has found a solution, its visited states are stored in a data structure. This requires the ant to have an internal memory to store the path, which is one of the properties of an artificial ant [16]. For this reason, the initial space of states is empty and all the possible transitions have the same amount of pheromones.

GBAP follows the *ant=rule* (i.e., *individual=rule*) approach [17]. Once the ant is created, it just represents the antecedent of the new rule. In Section III-E we will analyze how the consequent is properly assigned to the rule.

B. Heuristic measures

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})*. Due to the shape of the space of states, it could lead to premature convergence in few generations, because of the reinforcement of the pheromone amount in the early transitions. Thus, we consider here a new heuristic measure that guides the ants to follow those transitions that lead to a greater number of solutions. This measure increases the probability of choosing this kind of transitions, and it is based on the cardinality measure proposed in [18]. When initializing the grammar in the algorithm, a cardinality table for the maximum number 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 measure is the *information gain ($G(A_i)$)*, similar to the one used by the Ant-Miner algorithm. It is only used in transitions involving the application of production rules that imply the selection of attributes of the problem domain (i.e., $< COND >:= operator - attribute - value$).

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.

C. Transition rule

ACO metaheuristic follows a constructive method, i.e., every solution is built according to a sequence of transitions guided by some information. The information that bias each step is considered in the transition rule, which defines the probability that a given ant moves from a state *i* to another valid state *j*:

$$P_{ij} = \frac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\Sigma_{i=0}^{j} \eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}} \tag{3}$$

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

D. Pheromone updating

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 influence the environment. All transitions in the path get an equal amount of pheromone, and this 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_{ij} 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.

Once the pheromone trails in the environment have been reinforced and evaporated, a normalization process takes place. This process limits the pheromone levels in each transition, requiring that the amount of pheromone is in the range [minimum pheromone rate, maximum pheromone rate].

E. Fitness function

The fitness function that GBAP uses in the training stage for measuring the quality of the ants is the Laplace accuracy [19], which is defined as:

$$fitness = \frac{1+TP}{k+TP+FP} \tag{5}$$

where TP and FP represent 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 [20], whose purpose is to evolve different multiple rules for predicting each class in the data set while preserving the diversity. Depending on each data set 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: (a) the ants are sorted by their respectively class fitness in descending order; and (b) 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:

$$adjustedFitness = fitness \cdot \frac{\#capturedTokens}{\#classTokens} \tag{6}$$

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 training set– are added to the classifier.

Algorithm 1 High level pseudocode of GBAP

Require: $numberOfGenerations, numberOfAnts$

```
Initialize grammar and space of states
Create the classifier
for i = 0 to i = numberOfGenerations do
    Create list ants ← {}
    for j = 0 to j = numberOfAnts do
        ant ← Create new ant
        Store ant's path states in the space of states
        Evaluate ant, computing its fitness for each available class in the data set
        Add ant to the list ants
    end for
    Do niching algorithm, assigning the consequent to the ants and establishing the classifier rules
    for each ant in ants do
        if fitness > threshold then
            Update pheromone rate in the path followed by ant proportionally to its fitness
        end if
    end for
    Evaporate the pheromone rate along the whole space of states
    Normalize values of pheromones
end for
Establish the default rule in the classifier
predictiveAccuracy ← Compute the predictive accuracy obtained by the classifier when running over the test set
return predictiveAccuracy
```

F. Algorithm

GBAP algorithm is detailed in Algorithm 1. First, it starts up the grammar, creating a cardinality table for each production rule, and then it initializes the space of states with the initial state –which corresponds to the starting symbol of the grammar–. It also creates an empty classifier that will contain the ants that will remain after the competition that takes place in the niching algorithm in each generation.

In each generation, the algorithm initializes a new empty list of ants. Then it fills this list creating the number of ants specified as parameter, guiding the grammar this creation process. After creating every ant, the space of states is updated storing the states contained in the ant's path. The algorithm also computes k fitness per ant, where k is the number of available classes in the data set. Notice that at this point the ants only represent antecedents of rules.

Once all the ants have been created, these ants along with the ants of the classifier compete in the niching algorithm for seize as many instances of the data set as they can, as explained in Section III-E. The winners are assigned as rules to the classifier, replacing the previous ones.

Then, each ant created in this generation of the algorithm reinforces the amount of pheromones of its path's transitions only if its fitness is greater than the threshold value. To complete the generation, an evaporation and a normalization of pheromone values takes place.

After finishing all the generations, the default rule is added to the classifier and it is applied to the test set, computing the predictive accuracy.

IV. Experimental section

This section firstly describes the data sets used for the experimentation, as well as the preprocessing steps performed. It also explains the cross validation procedure carried out, and it specifies the algorithms used in the comparison and the parameters set up.

A. Data sets and preprocessing

In order to evaluate the performance of the proposed algorithm, GBAP was applied to several data sets, both artificial and real-world, selected from the well-known University of California at Irvine (UCI[1]) machine learning repository. Notice that this selection consists of a wide range of dimensionality data sets (varied number of instances, attributes and classes), and also considers the presence or absence of missing values in them. All these particular characteristics of each data set can be seen in detail in Table I.

Two preprocessing actions were performed using the Weka[2] Machine Learning library. One action involved the replacement of missing values (in such data sets comprising them) with the mode (for nominal attributes) or the arithmetic mean (for numerical attributes). On the other hand, to deal only with categorical attributes, the other action performed entailed discretizing data sets with numerical attributes [21], applying the Weka implementation of the Fayyad and Irani discretization algorithm [22].

[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 I
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

B. Cross validation

In order to evaluate GBAP algorithm and to compare it with others, we applied a stratified 10-fold cross-validation, i.e., we randomly splitted each data set into 10 mutually exclusive partitions, P_1,...,P_k, in such a way that each partition contained 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. Hence, the prediction performance of a given data set with N instances is considered as the average accuracy over these 10 folds, described as

$$predictiveAccuracy = \frac{\Sigma_{i=1}^{10}(C_i)}{N} \cdot 100 \qquad (7)$$

where C_i is the number of correctly classified instances when using de partition P_i as the test set.

In addition, in case of stochastic algorithms, we calculate the average accuracy and standard deviation of the 10 executions of each fold for the sake of reducing bias in the evaluation of the performance.

C. Algorithms and parameters

The rule induction algorithms considered for the experimentation are: Ant-Miner, PSO/ACO2, JRIP [21] and PART [21]. The first two algorithms were discussed in section II-A, since they are ACO based algorithms. JRIP is the Weka's implementation of Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algorithm, which is a rule induction algorithm. On the other hand, PART is based on the J48 Weka's algorithm –an improvement of C4.5– to generate decision trees from which rules are extracted.

These algorithms were executed with their default parameters, whereas GBAP's configuration parameters were set to: *number of ants* = 20, *number of generations* = 100, *max number of derivations* = 15, *initial pheromone rate* = 1.0, *minimum pheromone rate* = 0.1, *maximum pheromone rate* = 1.0, *evaporation rate* = 0.05, Q = 1.0, *alpha* = 0.4, and *betha* = 1.0.

V. RESULTS

Experiments compare the performance of our proposal with respect to other classification algorithms. This section interprets the results obtained.

The evaluation criteria for the comparison is the predictive accuracy. Table II shows average values for predictive accuracy with standard deviation. The results of the algorithm that yield the maximum average classification rate of each data set are highlighted in bold typeface. Analyzing the table, it is possible to realize that GBAP algorithm is competitive with respect to all the others considered, and also that it obtains the best results for the 52.9% of the data sets used in the experimentation. In the other data sets in which GBAP does not reach the best results, its classification results are competitive, being either the second or the third in rank, except for one data set in which it ranks fourth. Finally, regarding the standard deviation values of all algorithms per data set, we can also observe that GBAP globally yields middle values in terms of stability, and that in 6 of the data sets it has the stablest values.

To compare the results obtained and to analyze if there are significant differences between the classifiers, we perform the Friedman test. The Friedman test compares the average ranks of k algorithms over N datasets. Average rankings of all the algorithms considered are summarized in Table III,

TABLE II

PREDICTIVE ACCURACY(%) COMPARATIVE RESULTS

DATASET	GBAP	ANT-MINER	PSO/ACO2	JRIP	PART
Hepatitis	82.17 ± 12.04	83.34 ± 10.19	84.59 ± 9.33	81.54 ± 12.05	**84.64 ± 7.66**
Sonar	**81.98 ± 7.44**	76.07 ± 5.91	78.49 ± 8.05	80.33 ± 6.61	77.84 ± 8.10
Breast-c	71.40 ± 7.86	**72.67 ± 7.75**	68.63 ± 6.87	72.00 ± 6.41	68.48 ± 7.90
Heart-c	**82.84 ± 5.24**	76.38 ± 5.97	82.25 ± 5.36	82.20 ± 5.36	80.13 ± 6.39
Ionosphere	**93.02 ± 4.07**	84.03 ± 6.32	89.97 ± 4.99	91.70 ± 5.14	88.93 ± 4.02
Horse-c	82.97 ± 6.34	82.51 ± 5.69	82.06 ± 4.93	**83.72 ± 6.35**	81.50 ± 3.72
Breast-w	**96.50 ± 1.68**	94.23 ± 2.07	95.86 ± 1.91	95.71 ± 1.81	95.71 ± 1.82
Diabetes	**75.80 ± 4.12**	73.00 ± 4.33	74.16 ± 4.47	75.56 ± 2.34	75.66 ± 2.52
Credit-g	70.79 ± 4.27	69.63 ± 4.21	70.36 ± 3.55	70.70 ± 3.26	**72.70 ± 3.26**
Mushroom	98.26 ± 0.76	98.03 ± 0.93	99.90 ± 0.11	99.99 ± 0.04	**100.00 ± 0.00**
Iris	**96.67 ± 3.72**	95.33 ± 5.34	95.33 ± 6.70	96.00 ± 5.33	95.33 ± 6.70
Wine	**97.01 ± 4.37**	92.27 ± 4.92	90.20 ± 2.86	95.61 ± 5.37	95.03 ± 3.89
Balance-scale	75.49 ± 4.97	68.05 ± 5.32	**77.14 ± 4.93**	73.42 ± 5.66	76.50 ± 3.51
Lymphography	**81.00 ± 10.35**	73.83 ± 11.62	76.59 ± 12.2	78.84 ± 11.49	78.43 ± 14.3
Glass	69.13 ± 8.66	66.12 ± 9.61	71.16 ± 10.54	69.00 ± 8.70	**73.91 ± 8.43**
Zoo	**95.60 ± 4.21**	94.54 ± 6.45	92.32 ± 7.19	86.85 ± 7.25	94.84 ± 9.02
Primary	37.91 ± 6.55	33.57 ± 5.30	37.19 ± 5.88	38.11 ± 3.75	**38.36 ± 5.09**

where one can observe that the computed control algorithm (the algorithm with the lowest ranking) is our proposal. The Friedman statistic of average rankings distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom is 6.7391, which does not belong to the critical interval $C_0 = [0, (F_F)_{0.1,4,64} = 2.0348]$. Thus, we reject the null-hypothesis that all algorithms perform equally well when $\alpha = 0.1$.

TABLE III

AVERAGE RANKINGS OF THE ALGORITHMS

ALGORITHM	RANKING
GBAP	1.9412
Ant-Miner	4.2941
PSO/ACO2	3.2353
JRIP	2.7941
PART	2.7353

Due to the rejection of the null-hypothesis by the Friedman test, we proceed with a post-hoc test to reveal the performance difference. Since all classifiers are compared with respect to a control classifier, i.e. GBAP, we apply the Bonferroni-Dunn test [23]. Its results indicate that, at a significance level of $\alpha = 0.1$ (i.e., with a probability of 90%) there are significant differences, being the performance of GBAP statistically better than those of Ant-Miner and ACO/PSO2 algorithms. These results are captured in Figure 3, where one can also realize that GBAP achieves competitive or even better accuracy results than PART and JRIP.

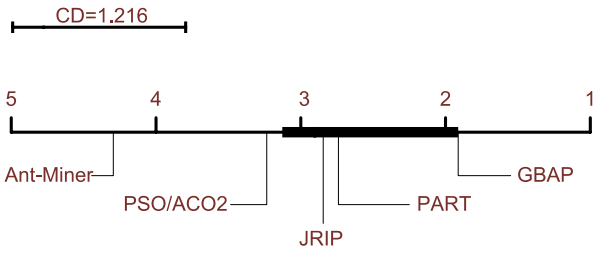

Fig. 3. Comparison of GBAP against the other classifiers with the Bonferroni-Dunn test. All classifiers with ranks outside the marked interval are significative different from the control ($p < 0.10$)

VI. CONCLUSIONS AND FUTURE WORK

In this paper we present a novel automatic programming algorithm based on ACO and restricted by a context-free grammar for mining classification rules from diverse data sets. This 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.

Moreover, in this work we have compared the performance of GBAP against two state-of-the-art algorithms (Ant-Miner and PSO/ACO2) and other two industry standard classifiers (JRIP and PART) over 17 different data sets. The results obtained demonstrate that GBAP is statistically more accurate than Ant-Miner and PSO/ACO2 with a significance level of 90%, and that GBAP is also competitive with JRIP and PART in terms of accuracy.

As future work we plan to apply other problem-dependent heuristic measures. We will also explore some new functionality to deal with continuous attributes and we will adapt the current version of GBAP to the multi-objective approach.

ACKNOWLEDGMENTS

This work has been supported by the Regional Government of Andalusia and Ministry of Science and Tech-

nology, projects P08-TIC-3720 and TIN2008-06681-C06-03, and FEDER funds.

REFERENCES

[1] M. Dorigo and T. Stützle, *The Ant Colony Optimization metaheuristic: Algorithms, Applications and Advances*, ser. International Series in Operations Research and Management Science, F. Glover and G. Kochenberger, Eds. Kluwer Academic Publishers, 2002, also available as technical report TR/IRIDIA/2000-32, IRIDIA, Université Libre de Bruxelles. [Online]. Available: ftp://iridia.ulb.ac.be/pub/mdorigo/tec.reps/TR.11-MetaHandBook.pdf

[2] R. Parpinelli, A. A. Freitas, and H. S. Lopes, "Data mining with an ant colony optimization algorithm," *IEEE Trans on Evolutionary Computation*, vol. 6, pp. 321–332, 2002.

[3] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming - An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. San Francisco, CA, USA: Morgan Kaufmann, January 1998. [Online]. Available: http://www.elsevier.com/wps/find/bookdescription.cws_home/677869/description\#description

[4] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: The MIT Press, 1992.

[5] O. Roux and C. Fonlupt, "Ant programming: or how to ants for automatic programming," in *ANTS'2000*, M. Dorigo and E. Al, Eds., 2000, pp. 121–129.

[6] E. Bonabeu, T. Eric, and M. Dorigo, *Swarm Intelligence: From Natural to Artificial Systems*. Nueva York, EUA : Oxford University, 1999.

[7] N. Holden and A. A. Freitas, "A hybrid PSO/ACO algorithm for discovering classification rules in data mining," *Journal of Artificial Evolution and Applications*, 2008.

[8] M. Boryczka and Z. J. Czech, "Solving approximation problems by ant colony programming," in *GECCO Late Breaking Papers*, 2002, pp. 39–46.

[9] J. Green, J. Whalley, and C. Johnson, "Automatic programming with ant colony optimization," pp. 70–77, 2004.

[10] M. Boryczka, "Ant colony programming with the candidate list," *LNAI*, vol. 4953, pp. 302–311, 2008.

[11] H. A. Abbass, X. Hoai, and R. I. Mckay, "AntTAG: A new method to compose computer programs using colonies of ants," in *In The IEEE Congress on Evolutionary Computation*, 2002, pp. 1654–1659.

[12] C. Keber and M. G. Schuster, "Option valuation with generalized ant programming," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds. New York: Morgan Kaufmann Publishers, 9-13 Jul. 2002, pp. 74–81. [Online]. Available: http://www.cs.bham.ac.uk/~wbl/biblio/gecco2002/aaaa075.ps

[13] A. Salehi-Abari and T. White, "Enhanced generalized ant programming (EGAP)," in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2008, pp. 111–118.

[14] S. Shirakawa, S. Ogino, and T. Nagao, "Dynamic ant programming for automatic construction of programs," *IEEJ Transactions on Electrical and Electronic Engineering (TEEE)*, vol. 3, no. 5, pp. 540–548, Aug 2008. [Online]. Available: http://dx.doi.org/doi:10.1002/tee.20311

[15] T. Stützle and H. H. Hoos, "Max-min ant system," *Future Generation Computer Systems*, vol. 16, pp. 889–914, 2000.

[16] R. J. Mullen, D. Monekosso, S. Barman, and P. Remagnino, "A review of ant algorithms," *Expert Systems with Applications*, vol. 36, pp. 9608–9617, 2009.

[17] P. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 40, no. 2, pp. 121–144, march 2010.

[18] A. Geyer-Schulz, *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, ser. Studies in Fuzziness. Heidelberg: Physica-Verlag, 1995, vol. 3.

[19] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *EWSL-91*. Springer-Verlag, 1991, pp. 151–163.

[20] 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*, 2009, pp. 570–577. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04394-9_69

[21] I. H. Witten and E. Frank, *Data Mining Practical Machine Learning Tools And Techniques*. Morgan Kauffman, 2005.

[22] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *13th International Joint Conference on Uncertainly in Artificial Intelligence(IJCAI93)*, 1993, pp. 1022–1029.

[23] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.