Using Ant Programming Guided by Grammar for Building Rule-Based Classifiers

Juan Luis Olmo, *Member, IEEE*, José Raúl Romero, *Member, IEEE*, and Sebastián Ventura, *Senior Member, IEEE*

***Abstract*—The extraction of comprehensible knowledge is one of the major challenges in many domains. In this paper an ant programming (AP) framework, capable of mining classification rules easily comprehensible by humans and, therefore, capable of supporting expert-domain decisions, is presented. The algorithm proposed, called GBAP (Grammar Based Ant Programming), is the first AP algorithm developed for the extraction of classification rules, and it is guided by a context-free grammar that ensures the creation of new valid individuals. To compute the transition probability of each available movement, this new model introduces the use of two complementary heuristic functions, instead of just one, as typical ant-based algorithms do. The selection of a consequent for each rule mined and the selection of the rules that make up the classifier is based on the use of a niching approach. The performance of GBAP is compared against other classification techniques on 18 varied data sets. Experimental results show that our approach produces comprehensible rules and competitive or better accuracy values than those achieved by the other classification algorithms compared with it.**

***Index Terms*—ant programming (AP), grammar-based automatic programming, ant colony optimization (ACO), classification, data mining (DM)**

I. Introduction

DATA MINING (DM) involves the process of applying specific algorithms for extracting comprehensible, non-trivial and useful knowledge from data. The discovered knowledge should have good generalization performance, i.e., it should accurately predict the values of some attributes or features of data that were not used during the run of the DM algorithm. This paper focuses on the classification task of DM, whose goal is to predict the value of the class given the values of certain other attributes (referred to as the predicting attributes). A model or classifier is inferred in a training stage by analyzing the values of the predicting attributes that describe each instance, as well as the class to which each instance belongs to. Thus, classification is considered to be supervised learning, in contrast to unsupervised learning, where instances are unlabelled. Once the classifier is built, it can be used later to classify other new and uncategorized instances into one of the existing classes.

A great variety of algorithms and techniques have been used to accomplish this task, including decision trees [1], decision rules [2], naive Bayes [3], support vector machines [4], neural networks [5], genetic algorithms [6], etc. In domains such as medical diagnosis, financial engineering, marketing, etc., where domain experts can use the model inferred as a decision-support system, decision trees and decision rules are especially interesting. These techniques have a high-level representation and, therefore, they allow the user to interpret and understand the knowledge extracted. For example, in medical problems, classification rules can be verified by medical experts, thus providing better understanding of the problem in-hand [7].

More recently, ant colony optimization (ACO) [8] has successfully carried out the extraction of rule-based classifiers. ACO is a nature-inspired optimization metaheuristic based on the behavior and self-organizing capabilities of ant colonies in their search for food. The first application of ACO to the classification task was the widely spread Ant-Miner algorithm, proposed by Parpinelly et al. [9], and it has become a benchmark algorithm in this field. Since then, several extensions and modifications of this sequential covering algorithm have been presented.

Another technique that has reported good results in classification is genetic programming (GP) [10]. GP is a particular type of automatic programming, a method that uses search techniques to find computer programs for solving a given problem, without requiring that the user knows the structure of the solution in advance. The user just has to specify the basic blocks that make up any program or individual, and how individuals are evaluated. Concretely, GP uses genetic algorithms as the search technique. Although automatic programming seems to fit well to classification problems, to the best of our knowledge, Ant Programming (AP)—a.k.a. ACO-based automatic programming [11]—, which is another kind of automatic programming method that instead uses ACO as the search technique, has never been explored to tackle classification problems. In this paper we first look at the AP works published in the literature, to prove that the development of AP algorithms and their application to DM is still an unexplored and promising research area. Then, 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, a context-free grammar (CFG) is used during the learning process. All generated individuals must adhere to this grammar, which also provides flexibility to apply the developed algorithm to a variety of problems with minor changes. Our proposal can support any number of classes, so that it can be easily applied to a large variety of data sets, generating a rule-based classifier. It aims to construct not only accurate but also comprehensible classifiers. In contrast to other ACO classification algorithms, our proposal provides more expressive power, because the grammar allows to control several aspects related to comprehensibility, such as the definition

Manuscript received August 7, 2010;

The authors are with the Department of Computer Science and Numerical Analysis, University of Cordoba, 14071 Cordoba, Spain (e-mail: {juanluisolmo,jrromero,sventura}@uco.es.

Digital Object Identifier ?

of specific operators, the specification of the conditions that can appear in rule antecedents or how these conditions are connected [12]. Moreover, our algorithm lacks the drawbacks of rule induction using sequential covering algorithms [13], as Ant-Miner, because it does not rule out examples when building the classifier.

It is a challenging task to compare the results obtained by this new approach with those obtained by other rule-based classification techniques. Thus, we perform a benchmarking experimental study where several classification techniques are considered, using a wide variety of data sets. The results obtained are promising and they show that our approach performs accurately, building understandable classifiers.

The remainder of this paper is organized as follows. In the next section we present some related work on ACO and a brief review of AP. In Section III, we describe the proposed algorithm. Section IV explains the experiments carried out, the data sets used and the algorithm set up. The results obtained are discussed in Section V. Finally, Section VI presents some concluding remarks.

II. Related Work

In this section, we first present some related work on the application of ACO to classification. We then provide a review of the various AP algorithms published in the literature so far.

A. Ant Colony Optimization

ACO is an agent-based nature-inspired optimization metaheuristic placed into swarm intelligence (SI) [14]. SI is concerned with the development of multi-agent systems inspired by the collective behavior of simple agents, e.g., flocks of birds, schools of fish, 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 foraging behavior and organization of ant colonies in their search for food, where ants communicate between themselves through the environment, in an indirect way, by means of a chemical substance—pheromone—that they spray over the path they follow—phenomenon known as *stigmergy*. The pheromone concentration in a given path increases as more ants follow this path, and it decreases more quickly as ants fail to travel it, since the evaporation in this path becomes greater than the reinforcement. The higher is the pheromone level in a path, the higher is the probability that a given ant will follow this path.

ACO algorithms were initially applied to combinatorial optimization problems [15], finding optimal or near optimal solutions. Since then, ACO algorithms have been engaged in an increasing range of problem domains, and they have also been shown to be effective when tackling the classification task of DM [16]. The first algorithm that applied ACO to rule induction was Ant-Miner [9], and it has become the most referred-to ACO algorithm in this field. It follows a separate-and-conquer approach where, starting from a training set and an empty set of rules, it finds new rules to be added to the set of discovered rules. As it discovers new rules, it removes those instances of the training set that are covered by each new rule, reducing the size of the training set. Ant-Miner chooses a new term for the current partial rule by applying the transition rule, and it only considers including terms that have not been previously chosen. It keeps on adding new terms to build this rule antecedent until one term from each available attribute has been selected, or until when selecting any term that is still available, the number of training instances covered by the rule is reduced below the value specified by the minimum cases per rule parameter. An information theoretic measure in terms of entropy is used as the heuristic function. The probability with which a given ant will select a node $term_{ij}$—a rule condition of the form $A_i = V_{ij}$, where A_i is the i-th attribute and V_{ij} is the j-th value of the domain of A_i—to be added to the current partial rule is assessed using the following formula:

$$P_{ij} = \frac{\eta_{ij} \cdot \tau_{ij}(t)}{\Sigma_{i=1}^{a} x_i \cdot \Sigma_{j=1}^{b_i} (\eta_{ij} \cdot \tau_{ij}(t))} \tag{1}$$

where η_{ij} is a problem-dependent heuristic function for $term_{ij}$; τ_{ij} is the amount of pheromone associated to the transition between attribute A_i and attribute A_j at time t; a is the total number of attributes; b_i is the number of values in the domain of attribute A_i; and x_i is set to 1 if the attribute A_i has not been selected yet for the construction of the current partial rule by the current ant, or otherwise, 0.

Ant-Miner continues discovering new rules until either the training set is empty or the number of training instances not covered by any rule is below a user-defined threshold. Finally, the majority class among the instances covered by the rule is assigned as the consequent. The quality of the rules is gauged by the following expression:

$$\begin{aligned} fitness =& sensitivity \cdot specificity \\ =& \frac{T_P}{T_P + F_N} \cdot \frac{T_N}{T_N + F_P} \end{aligned} \tag{2}$$

where T_P stands for true positives, which are positive instances correctly classified as positives; F_P stands for false positives, i.e., negative examples erroneously labeled as positives; T_N are true negatives, i.e., negative instances correctly identified as negatives; and F_N are false negatives, which are positive examples incorrectly classified as negatives [1].

Since the publication of Ant-Miner, other research works have followed the research lines suggested in [9], exploring the effects of several modifications. Table I lists chronologically the modifications of Ant-Miner, summing up the main differences of these algorithms with respect to the reference one. Some variations 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 or hierarchical classification.

Many of these extensions imply minor changes, and the results obtained are slightly different from the ones obtained by the original Ant-Miner. For example, Liu et al. [17] presented Ant-Miner2, where they applied a much simpler heuristic function, acting on the assumption that pheromone reinforcement has enough power to compensate possible errors induced by the use of this less effective heuristic measure. In addition to this change in the heuristic function, the same

TABLE I
MODIFICATIONS AND EXTENSIONS OF ANT-MINER

Reference	Pruning	Pheromone	Heuristic	Transition rule	Fitness measure	Interval rules	Continuous attributes	Fuzzy	Multi-Label	Hierarchical
Liu et al. [17]			√							
Liu et al. [18]		√	√	√						
Wang, Feng [19]		√	√	√						
Chen et al. [20]	√	√	√							
Chan, Freitas [21]	√									
Chan, Freitas [22]					√				√	
Smaldon, Freitas [23]	√	√	√		√					
Jin et al. [24]	√	√								
Galea, Shen [25]		√	√					√		
Swaminathan [26]		√				√				
Martens et al. [27]	√	√	√		√	√				
Otero et al. [28]		√	√				√			
Nalini, Balasubramanie [29]	√	√	√		√		√			
Otero et al. [30]	√	√	√		√		√			√
Salama, Abdelbar [31]		√	√	√	√					

authors proposed a different transition rule and pheromone reinforcement in Ant-Miner3 [18].

In contrast, Ant-Miner+, proposed by Martens et al. [27], demonstrated superior accuracy results than the previous Ant-Miner versions. This algorithm defines the environment as a directed acyclic graph, which allows the selection of better transitions and the inclusion of interval rules. It also implements the better performing max-min ant system (MMAS) [32] and uses a more accurate class-specific heuristic function. The fitness measure is defined as the sum of the confidence and the coverage of the individual (rule), as shown in the following equation:

$$\begin{aligned} fitness =& confidence + coverage \\ =& \frac{\mid A \cup C \subseteq I, I \in D \mid}{\mid A \subseteq I, I \in D \mid} + \frac{\mid A \cup C \subseteq I, I \in D \mid}{\mid D \mid} \end{aligned} \tag{3}$$

where the confidence refers to the quotient of the number of instances I belonging to the data set D that includes both the antecedent A and the consequent C, and the number of instances that include A. The coverage is gauged as the proportion between the number of instances that include both A and C and the number of instances that are not covered by any of the extracted rules, referred as $|D|$.

Another key difference of Ant-Miner+ lies in the value selected for the heuristic and the pheromone exponent parameters—α and β. In fact, it introduces a range for each parameter and lets the ants choose suitable values in an autonomous way.

In addition to these modifications, there are other extensions related to the hybridization of ACO with other metaheuristics. Among them, we appreciate the hybrid particle swarm optimization (PSO) - ACO algorithm, PSO/ACO2, developed by Holden et al. [33], for the discovery of classification rules. PSO is another optimization technique positioned among SI, inspired by the social behavior of birds in flocks or fish in schools. PSO/ACO2 is also a sequential-covering algorithm, and it can cope with both numerical and nominal attributes. It is a hybrid algorithm because it uses ACO to deal with nominal attributes and PSO to deal with numerical ones. The fitness function is given by the following expression:

$$fitness = \frac{1 + T_P}{1 + T_P + F_P} \tag{4}$$

B. Ant Programming

AP is an automatic programming technique that has certain similarities with GP, but rather than using genetic algorithms as search technique, it employs ACO to search for programs. There are different proposals using AP in the literature, which we now review, although their application is limited to problems such as symbolic regression, and no applications of AP to classification have been published so far.

The first work that combined the ants paradigm with the automatic generation of programs was presented by Roux and Fonlupt [11], and it was closely related to GP. In fact, their algorithm starts by creating a random population of programs (trees) using the ramped half-and-half initialization method 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 (also named terminals and functions, as GP does). Then, each program is evaluated and the pheromone table is updated by evaporation and reinforcement based on the quality of the solutions. These steps are repeated until some criteria are satisfied, but notice that new populations of programs are generated according to the pheromone tables. This approach was used to solve symbolic regression problems and a multiplexor problem with relative success.

A similar idea is that presented in [34], where each ant builds and modifies trees taking into account the quantity of pheromone at each node, which holds a pheromone table. The authors combined this approach with particle swarm optimization, with AP responsible for evolving the architecture of flexible neural networks, and PSO responsible for optimizing the parameters encoded in the neural tree. They applied developed flexible neural networks to a time-series prediction problem, showing the effectiveness of their algorithm.

Boryczka et al. [35], [36] also applied AP to solve symbolic regression problems, calling their method ant colony programming (ACP). They proposed two different versions of ACP, known as the expression approach and the 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. Green et al. [37] also presented an AP technique similar to the ACP expression approach. In turn, in the program approach the nodes in the graph represent assignment instructions, and the solution consists of a sequence of assignments that evaluate the function.

Boryczka also presented extensions to these works with the aim of improving the evaluation performance. In [38], the author improved the effectiveness and achieved more simplified solutions by eliminating introns, while in [39] the computational time necessary for the evaluation of transition rules was reduced.

Another attempt to evolve programs using the ACO algorithm was AntTAG [40]. It was proposed by Abbass et al. as a method of automatic programming employing ACO as its search strategy and a tree adjoining grammar (TAG) to build programs. TAGs are compact contex-sensitive grammars that use tree manipulation operations for syntactic analysis, and they can distinguish between derivation and derivation trees as well. The authors tested its performance on symbolic regression problems and achieved better performance than grammar guided GP [41] or TAG guided GP [42].

Keber and Schuster published another grammar-based work called generalized ant programming (GAP) [43], which uses a CFG instead of TAG, and where ants generate a program by following a path in the space of states. Salehi-Abari and White [44] 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. Then, Salehi-Abari and White published another work [45] comparing the performance of GP against their EGAP algorithm in three different problems: quarctic symbolic regression, multiplexer and Santa Fe ant trail.

More recently, Shirakawa et al. [46] proposed dynamic ant programming (DAP). Its main difference with regards to ACP lies in the use of a dynamically changing pheromone table and a variable number of nodes, which leads to a more compact space of states. The authors only compared the performance of DAP against GP using symbolic regression problems.

III. GBAP: Grammar Based Ant Programming Algorithm

In this section we describe the main features of Grammar Based Ant Programming (GBAP) algorithm.

In short, GBAP is an automatic programming algorithm that uses ACO as its search technique and which is also guided by a context-free grammar. The use of a grammar in this kind of systems establishes a formal definition of the syntactical restrictions, defines the space of states, and ensures that any solution found is syntactically valid. In fact, it guarantees the closure property that must be fullfilled in any automatic programming system [10], as any state and any feasible solution can only be reached from the initial state in a certain sequence of steps by applying the production rules available.

The GBAP algorithm has been conceived for obtaining a specific classifier arising from a learning process over a given training set. The output classifier is an ordered rule list in which discovered rules are sorted in descending order by their fitness. A default rule predicting the majority class of the training set is added at the bottom of the classifier. Once the model has been learned from the training set, to classify a new instance, the label assigned corresponds to the consequent of the first rule in the classifier whose antecedent matches the instance. In case it gets to the end of the classifier without any rule antecedent covering this new instance, it would be classified by the default rule.

As outlined in the following sections, the GBAP algorithm can not be fitted into a typical ACO system. Due to the bounding of the pheromone levels to within the interval $[\tau_{min}, \tau_{max}]$, and to the initialization of all edges to the maximum pheromone amount allowed, the algorithm with which GBAP shares more characteristics may be the MMAS [32]. However, unlike how the reinforcement is carried out in GBAP, in MMAS only the best ant is responsible for updating pheromone trails. The complexity of MMAS-based algorithms is a complex research area, which has been widely studied and analyzed by Neumann et al. [47].

On the other hand, regarding the AP algorithms reviewed in Section II-B, GBAP presents certain similarities with GAP [43] and EGAP [44], specially in the use of a context-free grammar and in the shape adopted by the space of states.

The main characteristics of GBAP are depicted in the following subsections, presenting also a detailed pseudocode of the algorithm.

G = (Σ_N, Σ_T, P, EXP)
Σ_N = {EXP, COND}
Σ_T = {'AND', '=', '!=',
'attr$_1$', 'attr$_2$', ..., 'attr$_n$',
'value$_{11}$', 'value$_{12}$', ..., 'value$_{1m}$',
'value$_{21}$', 'value$_{22}$', ..., 'value$_{2m}$',
..., 'value$_{n1}$', 'value$_{n2}$', ..., 'value$_{nm}$'}
P = {EXP = COND | 'AND' , EXP , COND ;
COND = all possible valid
combinations of the ternary
operator-attribute-value ; }

Fig. 1. Context-free grammar used in GBAP

A. Environment and Rule Encoding

GBAP prescribes a CFG for representing the antecedent of the rule encoded by each individual. It is expressed in extended

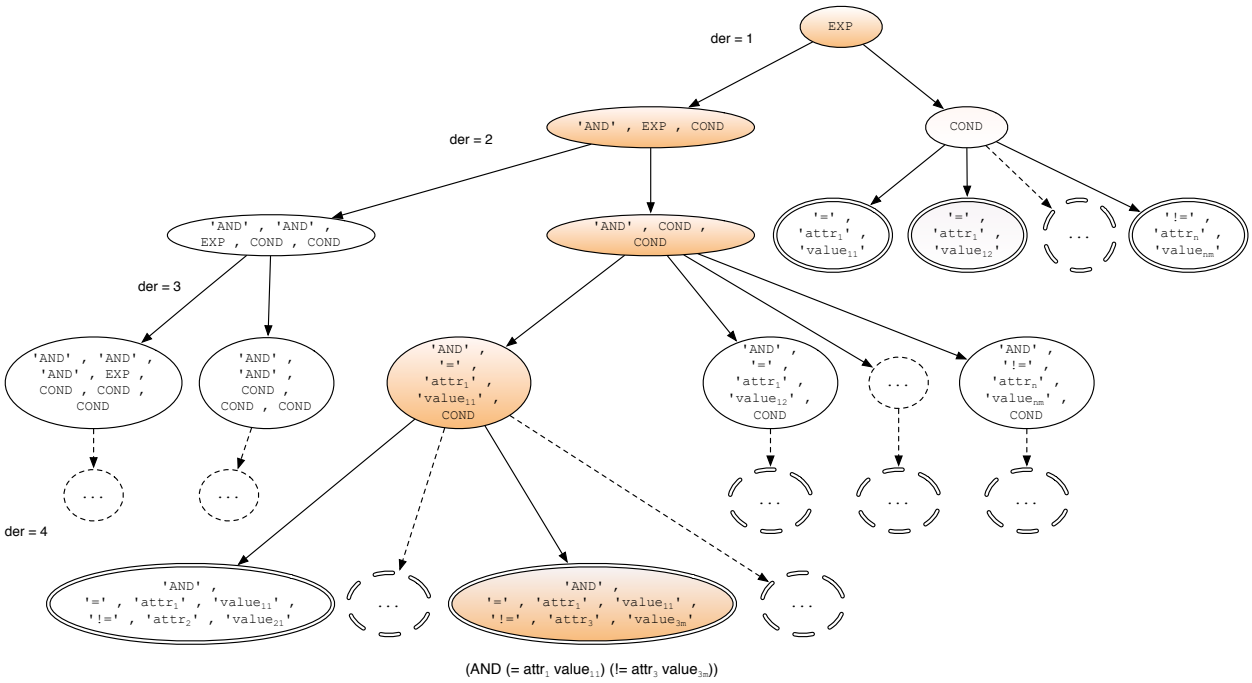

Fig. 2. Space of states at a depth of four derivations. The sample coloured path represents the antecedent found by a given ant.

Backus-Naur form (EBNF) notation, and its definition is given by $G = (\Sigma_N, \Sigma_T, P, EXP)$, where Σ_N is the set of non-terminals, Σ_T is the set of terminals, P is the set of production rules, and EXP is the start symbol of the grammar. The grammar is 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. Hence 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 for performance reasons, in order to save on computation costs when gauging rule fitness.

Observe that like GP, grammar guided systems also use the terminal and non-terminal nomenclature, but here it refers to the symbols of the grammar, rather than to the leaf nodes or function/internal nodes of an individual tree representation in GP. In grammar guided GP, the grammar controls the creation of the initial population of individuals, the crossover, mutation and reproduction processes; in contrast, in grammar guided AP, because there are no genetic operators involved, the grammar looks after each movement of each ant in such a way that each ant will follow a valid path and will find a feasible solution to the problem.

Concerning the design of any ant inspired algorithm, it is necessary to specify an environment where ants cooperate with each other. In GBAP, this environment is the search space comprising all possible expressions or programs that can be derived from the grammar in the number of derivations available. Thus, the environment adopts the shape of a derivation tree, as shown in Figure 2 at a depth of three derivations.

Starting with the initial state of the environment, which is associated with the start symbol defined by the grammar, each ant tries to build a feasible solution to the problem. Any solution found takes the form of a path from the root node to a final state 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. A final state—represented in the figure with a double-border oval—only contains terminal symbols and, therefore represents the evaluatable expression of the antecedent of the rule encoded. Although final states encode an evaluatable antecedent, fulfilling the properties of an artificial ant [48], ants have an internal memory to store the path in order to do an offline pheromone update.

The treatment of the environment is a key issue of the algorithm. It should be pointed out that, depending on both the dimensionality of the data set addressed and the number of derivations permitted from the grammar, it may suppose an excessive computational cost to keep in memory the whole space of states. Thus, with respect to the space of states generation, GBAP follows an incremental build approach. The data structure that represents the space of states is initialized just with the initial state and all possible transitions have the same quantity of pheromones. This data structure also contains attributes that take into account the effects of the evaporation and normalization processes over the environment. The data structure is filled as ants are created, storing there the states of each ant's path.

Regarding the individual encoding, GBAP follows the *ant=rule* (i.e., *individual=rule*) approach [12]. As aforementioned, when ants have been just created they only represent the antecedent of a new rule. The consequent will be

assigned by following the niching approach described later in Section III-F.

B. Algorithm

The main steps of GBAP are detailed in the pseudocode of Algorithm 1. It begins by starting up the grammar, creating a cardinality table for each production rule, and initializing the space of states with the initial state. It also creates an empty object that represents the classifier, which will contain the remaining—winner—ants of the competition that takes place in the niching algorithm in each generation. The algorithm starts with the minimum number of derivations that are necessary to find a solution in the space of states and computes the derivation step for each generation. Notice that in the case of the grammar defined, at least two derivations are needed to reach a solution from the initial state, as can be seen in Figure 2.

Algorithm 1 High Level Pseudocode of GBAP

Require: $numGenerations, numAnts, maxDerivations$
1: Initialize grammar and space of states
2: Create an empty classifier
3: $derivationStep \leftarrow \frac{maxDerivations-2}{numGenerations}$
4: $maxDerivations \leftarrow 2$
5: **for** $i = 0$ **to** $i = numGenerations$ **inc** 1 **do**
6: Create list *ants* $\leftarrow \{\}$
7: **for** $j = 0$ **to** $j = numAnts$ **inc** 1 **do**
8: $ant \leftarrow$ Create new ant (see Procedure 2)
9: Store *ant*'s path states in the space of states
10: Evaluate *ant*, computing its fitness for each available class in the data set
11: Add *ant* to the list *ants*
12: **end for**
13: Niching approach to assign the consequent to the ants and to establish the classifier rules (see Procedure 3)
14: **for** each *ant* in *ants* **do**
15: **if** $fitness > threshold$ **then**
16: Update pheromone rate in the path followed by *ant* proportionally to its fitness and inversely proportional to its path's length
17: **end if**
18: **end for**
19: Evaporate the pheromone along the whole space of states
20: Normalize values of pheromones
21: $maxDerivations \leftarrow maxDerivations + derivationStep$
22: **end for**
23: Establish the default rule in the classifier
24: $predictiveAccuracy \leftarrow$ Compute the predictive accuracy obtained when running the classifier built on the test set
25: **return** *predictiveAccuracy*

A new list of ants is initialized at the beginning of each generation, and the algorithm fills this list, creating the number of ants specified by a parameter. The states visited by each new ant are stored in the space of states. Then, the algorithm computes k fitness values per ant, k being the number of classes in the data set. Notice that at this point each ant encodes only the antecedent of a rule because the consequent has not been assigned yet.

Once all ants have been created, these ants along with the ants assigned to the classifier in the previous generation compete in the niching algorithm. They try to capture as many instances of the data set as they can, as explained in Section III-F. Then, a consequent is assigned to each ant. To conclude the niching algorithm, the winner ants are assigned to the classifier, replacing the previous rules.

Procedure 2 Ants Creation

Require: $maxDerivations$
1: Create list $path \leftarrow \{\}$
2: $n \leftarrow$ Initial state
3: Add n to the list $path$
4: **repeat**
5: $maxDerivations \leftarrow maxDerivations - 1$
6: $n \leftarrow$ Select next movement from space of states, n being the source node, and $maxDerivations$ the number of derivations available
7: Add n to the list $path$
8: **until** (n is a final node)
9: $ant \leftarrow$ New Ant with its path set to $path$
10: **return** ant

Afterwards, each ant created in this generation of the algorithm reinforces the amount of pheromones of the transitions followed only if it has a fitness greater than the threshold value. To complete the generation, an evaporation and a normalization process takes place. The maximum number of derivations is also incremented by the derivation step.

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

The creation process of a given ant is described in Procedure 2. First, the algorithm initializes a new empty list to store the nodes visited by the new ant. Then, it creates a new node n that corresponds to the initial state of the environment and adds this node to the path list. Following a stepwise approach, the main loop of the algorithm takes care of selecting the next movement of the ant from the current state, decreasing by one the number of derivations that remain available. It also adds the newly visited state to the list $path$. It finishes when a final state is reached and, therefore, the ant has found a solution. Finally, a new ant is created from the list of visited states $path$.

C. Heuristic Measures

Another differentiating factor of GBAP with respect to ACO algorithms lies in the use of two components in the heuristic function that can not be applied simultaneously. To distinguish which one apply, GBAP needs to find out which type of transition it is about, considering two different cases, which we refer as intermediate transitions (i.e., transitions not involving production rules that imply the selection of attributes of the problem domain) and final transitions (i.e., transitions that suppose the application of production rules of the type `COND = 'operator' , 'attribute' , 'value' ;`).

For intermediate transitions, the measure considered is related to the cardinality of the production rules, and is referred to as P_{card}. This component increases the likelihood that a given ant chooses transitions that lead to a greater number of candidate solutions. It is based on the cardinality measure proposed in [49], and to use it, the algorithm works out a cardinality table for each production rule when it initializes

the grammar. This cardinality table will contain as many entries as the maximum number of derivations available, where each entry maps the number of derivations to the number of solutions that can be reached. Thus, given a state *i* having *k* subsequent states, be *j* a specific successor among those *k* states, and where *d* derivations remain available at that point, $P_{card_{ij}^k}$ is gauged as the ratio between the number of candidate solutions that can be successfully reached from the state *j* in *d-1* derivations, and the sum of all possible candidate solutions that can be reached from the source state *i* (see Eq. 5). Notice that P_{card} has no sense in final transitions, since each possible destination node embraces the same number of solutions.

$$P_{card_{ij}^k} = \frac{cardinality(state_j, d-1)}{\Sigma_{k \in allowed}(cardinality(state_k, d-1))} \quad (5)$$

In contrast, for final transitions, *information gain (*$G(A_i)$*)* [9] is the component considered. It measures the worth of each attribute for separating the training examples with respect to their target classification, and it is computed as:

$$G(A_i) = I - I(A_i) \quad (6)$$

where I represents the entropy of the classes in the training set, and $I(A_i)$ is the entropy of the classes given the values for attribute A_i. In turn, these are computed as follows:

$$I = -\sum_{c=1}^{\#classes} (\frac{n_c}{n} \cdot log_2 \frac{n_c}{n}) \quad (7)$$

where n_c is the number of instances of class c, and n stands for the number of instances in the training set.

$$I(A_i) = \sum_{j=1}^{\#valuesA_i} (\frac{n_{ij}}{n} \cdot I(A_{ij})) \quad (8)$$

where n_{ij} is the number of instances with value j in attribute A_i, and $I(A_{ij})$ is the entropy of the classes given the value j for attribute A_i, computed as:

$$I(A_{ij}) = -\sum_{c=1}^{\#classes} (\frac{n_{ijc}}{n_{ij}} \cdot log_2 \frac{n_{ijc}}{n_{ij}}) \quad (9)$$

where, finally, n_{ijc} stands for the number of instances of class c with the value j in attribute A_i.

D. Transition Rule

The 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 biases each step is considered in the transition rule, which defines the probability that a given ant moves from a state *i* to another state *j*:

$$P_{ij}^k = \frac{(\eta_{ij})^\alpha \cdot (\tau_{ij})^\beta}{\Sigma_{k \in allowed}(\eta_{ik})^\alpha \cdot (\tau_{ik})^\beta} \quad (10)$$

where k is the number of valid subsequent states, α is the heuristic exponent, β is the pheromone exponent, η is the value of the heuristic function, computed as $G(A_i)+P_{card}$ (having at least one of the two components equal to zero), and τ indicates the strength of the pheromone trail.

The transition rule will assign a probability to each available next state. The algorithm arranges that each possible transition can reach at least one final state or solution in the number of derivations that remain available at that point. If not, those transitions will be assigned a probability of zero and, therefore, the ant will never select such movements.

E. Pheromone Updating

With regards to the pheromone maintenance, two operations are considered: reinforcement and evaporation. Each ant of the current generation is able to reinforce the pheromone amount in its path's transitions only if the quality of the solution encoded by this 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 experimentally at 0.5 so that bad solutions will never influence the environment. All transitions in the path get an equal amount of pheromone, and this reinforcement is based both on the length and the quality of the solution encoded by the ant:

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot Q \cdot fitness \quad (11)$$

where τ_{ij} represents the amount of pheromone in the transition from the state *i* to the state *j*, and *Q* is a computed measure that favors comprehensible solutions. In fact, the value of this parameter depends on the length of the solution encoded by the ant, being calculated as the ratio between the maximum number of derivations in the current generation and the length of the path followed by the ant (shorter solutions will receive more pheromone thus).

Regardless of whether the ant's quality is greater than the threshold value, the evaporation process always takes place, involving all transitions in the space of states, as seen in the following expression:

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot (1-\rho) \quad (12)$$

where ρ represents the evaporation rate.

Once the pheromone trails in the environment have been reinforced and evaporated, a normalization process takes place, in order to bound the pheromone amount existing in each transition to the range $[\tau_{min}, \tau_{max}]$.

Notice that it is quite complicated to store the environment's pheromone values in a pheromone matrix. Instead, each value is kept in the data structure that corresponds to the destination state of a given transition.

F. Fitness Function and Consequent Assignment

The fitness function that GBAP uses in the training stage to conduct the search process is the Laplace accuracy [50]. This measure was selected because it suits well to multiclass classification since it takes into account the number of classes in the data set. The quality of the ants is then defined by:

$$fitness = \frac{1+T_P}{k+T_P+F_P} \quad (13)$$

Procedure 3 GBAP Niching Approach

Require: $ants, classifier, minCasesPerRule, trainingSet$
Create list *winnerAnts* $\leftarrow \{\}$
$numInstances \leftarrow$ number of instances in $trainingSet$
$numClasses \leftarrow$ number of classes in $trainingSet$
for $k \leftarrow 0$ **to** $k < numClasses$ **inc** 1 **do**
  Sort $ants$ by fitness corresponding to class k
  $flagsArray \leftarrow$ boolean array of $numInstances$ elements
  for $i \leftarrow 0$ **to** $i < |flagsArray|$ **inc** 1 **do**
    $flagsArray[i] \leftarrow$ **false**
  end for
  for each *ant* in *ants* **do**
    $idealTokens \leftarrow 0$
    $capturedTokens \leftarrow 0$
    for $j \leftarrow 0$ **to** $j < numInstances$ **inc** 1 **do**
      $instance \leftarrow$ Get instance j of $trainingSet$
      $cat \leftarrow$ Get category of $instance$
      if $cat = k$ **then**
        $idealTokens \leftarrow idealTokens + 1$
        if $flagsArray[j] =$ **false and** ant covers $instance$ **then**
          $flagsArray[j] \leftarrow$ **true**
          $capturedTokens \leftarrow capturedTokens + 1$
        end if
      end if
    end for
    if $capturedTokens >= minCasesPerRule$ **then**
      $fitness_{adj}[k] \leftarrow \frac{capturedTokens}{idealTokens}$
    else
      $fitness_{adj}[k] \leftarrow 0$
    end if
  end for
end for
for each *ant* in *ants* **do**
  $max \leftarrow 0$
  for $k \leftarrow 0$ **to** $k < numClasses$ **inc** 1 **do**
    if $fitness_{adj}[k] > fitness_{adj}[max]$ **then**
      $max \leftarrow k$
    end if
  end for
  Set $consequent$ of ant to class max
  if $fitness_{adj}[max] > 0$ **then**
    Add ant to $winnerAnts$
  end if
end for
return $winnerAnts$

where T_P and F_P 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 quite similar to that employed in [51], whose purpose is to evolve different multiple rules for predicting each class in the data set while preserving the diversity. Depending both on the data set and the distribution of instances per 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 that it does not overlap with instances of another class. In addition, it is appropriate when removing redundant rules. Moreover, it lacks the drawbacks that are present in covering algorithms related to the discarding of instances. Since these are covered by mined rules, they are removed from the training set and in consequence, the fitness of the subsequent rules discovered is computed by using a smaller number of instances. Thus, covering algorithms seek rules with good performance in the *sub*-data set.

The pseudocode of the niching algorithm developed is shown in Procedure 3. Notice that each existing instance in the training set is called a token: all ants will compete to capture the tokens, bearing in mind that a given token can be captured at most by one ant. Remember that GBAP has previously computed an array of *k* fitness values per individual, one for each class (assuming that the respective class is assigned as consequent to that individual). Then, the niching algorithm repeats the following process k times, considering each iteration a different class related to their order of appearance in the metadata section of the training set: (a) the ants are sorted out by their respective class fitness in descending order; and (b) each ant tries to capture as many tokens of the computing class as it covers, in case of tokens not seized before by any other ant. Finally, if the rule encoded by the ant has captured at least $minCasesPerRule$, the ant's adjusted fitness value for this class is computed as described in Eq. 14, where $idealTokens$ stands for the ideal number of tokens that the current ant could capture, i.e., the number of tokens belonging to the computing class (regardless of whether they have been previously captured by other ant or not). Otherwise, if the number of captured tokens is below the threshold established by $minCasesPerRule$, an adjusted fitness value of zero is assigned to the ant.

$$fitness_{adj} = fitness \cdot \frac{\#capturedTokens}{\#idealTokens} \tag{14}$$

Notice that the number of $idealTokens$ is always greater or equal than $capturedTokens$. Thus, the closer are their values, the less penalized is the ant (in fact, if $capturedTokens = idealTokens$, the ant is not penalized).

Once the *k* adjusted fitness values 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.

IV. Experimentation

In this section we will first present the data sets used in the experimental study, along with the preprocessing actions performed. Then, we explain the cross validation procedure employed. Finally, the parameter set-up for the different algorithms considered in the comparison is presented.

A. Data Sets and Preprocessing

The performance of GBAP was tested on 18 publicly available data sets, both artificial and real-world, selected from the machine learning repository of the University of California at Irvine (UCI) [52] [1]. We have selected problems with a wide range of dimensionality with respect to the number of classes

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

TABLE II
DATA SETS DESCRIPTION

DATASET	MISSING VALUES	INSTANCES	ATTRIBUTES			CLASSES	DISTRIBUTION OF CLASSES
			Continuous	*Binary*	*Nominal*		
Hepatitis	yes	155	6	13	0	2	32 / 123
Sonar	no	208	60	0	0	2	97 / 111
Breast-c	yes	286	0	3	6	2	201 / 85
Heart-c	yes	303	6	3	4	2	165 / 138
Ionosphere	no	351	33	1	0	2	126 / 225
Horse-c	yes	368	7	2	13	2	232 / 136
Australian	yes	690	6	0	9	2	307 / 383
Breast-w	yes	699	9	0	0	2	458 / 241
Diabetes	no	768	0	8	0	2	500 / 268
Credit-g	no	1000	6	3	11	2	700 / 300
Mushroom	yes	8124	0	0	22	2	4208 / 3916
Iris	no	150	4	0	0	3	50 / 50 / 50
Wine	no	178	13	0	0	3	59 / 71 / 48
Balance-scale	no	625	4	0	0	3	288 / 49 / 288
Lymphography	no	148	3	9	6	4	2 / 81 / 61 / 4
Glass	no	214	9	0	0	6	70 / 76 / 17 / 13 / 9 / 29
Zoo	no	101	1	15	0	7	41 / 20 / 5 / 13 / 4 / 8 / 10
Primary-tumor	yes	339	0	14	3	21	84 / 20 / 9 / 14 / 39 / 1 / 14 / 6 / 2 / 28 16 / 7 / 24 / 2 / 1 / 10 / 29 / 6 / 2 / 1 / 24

and attributes. These data sets are listed in Table II, where their particular characteristics are also described.

Due to the fact that the data sets considered contained numerical attributes and missing values, two preprocessing actions were performed using Weka[2]. A first one entailed the replacement of missing values with the mode (for nominal attributes) or the arithmetic mean (for numerical attributes). And the other involved the discretization of such data sets containing numerical attributes, by applying Fayyad and Irani's discretization algorithm [53]. The replacement of missing values was done before partitioning the data set, and the discretization was applied for each specific training set, using the same intervals found to discretize the corresponding test set.

B. Cross Validation

For each data set and algorithm, we performed a stratified ten-fold cross-validation procedure, where we randomly split each data set into ten mutually exclusive partitions, P_1,...,P_{10}, containing approximately the same number of instances and the same proportion of classes present in the original data set. Then, ten 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 predictive accuracy obtained on a given data set is considered as the average accuracy over these ten folds, described as

$$predAcc = \frac{\Sigma_{i=1}^{10}(\#correctlyClassifiedP_i)}{\#instances} \cdot 100 \qquad (15)$$

where $\#correctlyClassifiedP_i$ is the number of correctly classified instances when using P_i as test set, and $\#instances$ is the number of instances in the original data set.

In addition, to avoid any chance of obtaining biased results when evaluating the performance of stochastic algorithms, ten executions per fold were performed, using ten different seeds.

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

C. Algorithms and Parameter Set-Up

For comparison purposes, six other rule induction algorithms were considered: three ant-based algorithms, Ant-Miner [3], Ant-Miner+ [4] and PSO/ACO2 [5], which were discussed in Section II-A; a GP algorithm, Bojarczuk-GP [54] [6], which will be explained briefly next; and two well-known classifiers, JRIP—the Weka's implementation of the popular sequential covering Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algorithm—and PART, which extract rules from the decision trees generated by the J48 Weka's algorithm. It is worth noting at this point that every algorithm used in the experimentation was run over the same discretized partitions of the data sets previously mentioned, even in the case of those capable of handling numerical values.

Bojarczuk-GP is a GP algorithm for classification rule mining that reports good accuracy and comprehensibility results when applied to medical data sets. It is a constrained syntax algorithm which represents the rules by defining a set of functions consisting both of logical operators (AND, OR) and relational operators ($=, \neq, \leq, >$). Bojarczuk-GP follows a mixed *individual=rule/rule set* approach, where each individual encodes a set of rules in disjunctive form that predict the same class, and the classifier generated for a given problem consists of k individuals, k being the number of classes in the data set. The genetic operators considered by this algorithm are crossover and reproduction, so that no mutation is performed during the evolution. For the sake of evolving comprehensible rules, the fitness function evolves three terms:

$$fitness = sensitivity \cdot specificity \cdot simplicity \qquad (16)$$

[3]Ant-Miner was run using framework Myra (version 2.1.2), which can be downloaded from http://myra.sourceforge.net/

[4]The source code of Ant-Miner+ was provided by the authors.

[5]The PSO/ACO2 algorithm was run using the implementation available at http://sourceforge.net/projects/psoaco2

[6]Bojarczuk-GP was run using the existing implementation in the evolutionary computation framework JCLEC [55], which is publicly available at http://jclec.sourceforge.net

TABLE III
USER-DEFINED PARAMETER CONFIGURATION

ALGORITHM	NAME	DESCRIPTION	VALUE
GBAP			
	numAnts	Number of ants	20
	numGenerations	Number of generations	100
	maxDerivations	Maximum number of derivations for the grammar	15
	minCasesPerRule	Minimum number of instances covered per rule	3
	$[\tau_0]$	*Initial pheromone amount for transitions*	*1.0*
	$[\tau_{min}]$	*Minimum pheromone amount allowed for transitions*	*0.1*
	$[\tau_{max}]$	*Maximum pheromone amount allowed for transitions*	*1.0*
	$[\rho]$	*Evaporation rate*	*0.05*
	$[\alpha]$	*Heuristic exponent value*	*0.4*
	$[\beta]$	*Pheromone exponent value*	*1.0*
ANT-MINER			
	number of ants	Number of ants	1
	min. cases per rule	Minimum number of instances covered per rule	10
	max. uncovered cases	Maximum number of uncovered cases	10
	rules for convergence	Convergence test size	10
	number of iterations	Maximum number of iterations	3000
ANT-MINER+			
	nAnts	Number of ants	1000
	rho	Evaporation rate	0.85
PSO/ACO2			
	numParticles	Number of particles	10
	numIterations	Number of iterations	200
	maxUncovExampPerClass	Maximum number of uncovered examples per class	2
GP			
	population-size	Number of individuals	200
	max-of-generations	Number of generations	100
	max-deriv-size	Maximum number of derivations for the grammar	20
	recombination-prob	Crossover probability	0.8
	reproduction-prob	Reproduction probability	0.05
	parents-selector	Selection method for both parents	Roulette
JRIP			
	checkErrorRate	Whether check for error rate $>= 1/2$ is included in stopping criterion	True
	folds	Determines the amount of data used for pruning. One fold is used for pruning, the rest for growing the rules	3
	minNo	The minimum total weight of the instances in a rule	2.0
	optimizations	The number of optimization runs	2
	pruning	Whether pruning is performed	True
PART			
	binarySplits	Whether to use binary splits on nominal attributes when building the partial trees	False
	confidenceFactor	The confidence factor used for pruning (smaller values incur more pruning).	0.25
	minNumObj	The minimum number of instances per rule	2
	numFolds	Determines the amount of data used for reduced-error pruning. One fold is used for pruning, the rest for growing the rules	3
	reducedErrorPruning	Whether reduced-error pruning is used instead of C4.5 pruning	False
	unpruned	Whether pruning is performed	False

where sensitivity and specificity are computed as indicated in Eq. 2 and simplicity is computed as follows:

$$simplicity = \frac{maxnodes - 0.5 \cdot numnodes - 0.5}{maxnodes - 1} \quad (17)$$

where $maxnodes$ is the maximum number of nodes allowed and $numnodes$ is the current number of nodes. Thus, the goal of the fitness function is to maximize both sensitivity and specificity, while minimizing the complexity of the rule set. When the evolutionary process terminates, the classifier is set-up with the best individual found for each class.

For each algorithm, excluding GBAP, its user-defined parameters were set to the values reported by the authors in the aforementioned references. The parameter configuration is summarized in Table III. As it can be observed, GBAP seems to have more parameters than the other ACO-based algorithms, and it may be a disadvantage for the final user. Nevertheless, the other ACO algorithms also have parameters that are hidden for the final user. For example, in the paper were Ant-Miner+ was proposed [27], the authors describe parameters such as α, β, *early stopping criterion*, or parameters that are implicit to the MMAS approach followed by this algorihtm—τ_0, τ_{min} and τ_{max}—, but the authors have preset their value in the code of the algorithm. We could have reduced the number of user-defined parameters just to four—$numAnts$, $numGenerations$, $maxDerivations$ and $minCasesPerRule$—, prefixing the value for the rest of parameters in the algorithm's code to the values reported in Table III, but this could be also a disadvantage for a given expert

TABLE IV
PREDICTIVE ACCURACY(%) COMPARATIVE RESULTS

	GBAP		ANT-MINER		ANT-MINER+		PSO/ACO2		GP		JRIP		PART	
Dataset	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}
Hepatitis	82.17	12.04	83.27	10.32	81.79	10.30	84.59	9.33	71.05	14.45	81.54	12.05	**84.64**	7.66
Sonar	**81.98**	7.44	76.95	6.89	76.05	7.22	78.49	8.05	79.82	9.24	80.33	6.61	77.84	8.10
Breast-c	71.40	7.86	**73.42**	7.29	73.05	6.86	68.63	6.87	68.63	10.94	72.00	6.41	68.48	7.90
Heart-c	**82.84**	5.24	78.01	6.69	82.41	5.10	82.25	5.36	70.02	7.08	82.20	5.12	80.13	6.39
Ionosphere	**93.02**	4.07	84.39	6.73	92.89	4.02	89.97	4.99	76.48	8.19	91.70	5.14	88.93	4.02
Horse-c	82.97	6.34	82.71	4.73	81.79	6.03	82.06	4.93	82.52	6.06	**83.72**	6.35	81.5	3.72
Australian	85.47	4.49	85.30	4.12	83.48	3.38	85.19	4.69	85.52	4.50	**86.70**	5.15	84.66	4.48
Breast-w	**96.50**	1.68	94.69	2.04	94.28	2.86	95.86	1.91	87.39	2.75	95.71	1.81	95.71	1.82
Diabetes	**75.80**	4.12	72.48	3.76	74.58	4.81	74.16	4.47	61.94	4.72	75.56	2.34	75.66	2.52
Credit-g	70.79	4.27	70.55	3.72	70.80	3.87	70.36	3.55	63.02	7.03	70.70	3.26	**72.70**	3.26
Mushroom	98.26	0.76	98.15	0.71	98.89	0.63	99.90	0.11	86.22	6.11	99.99	0.04	**100.00**	**0.00**
Iris	**96.00**	4.10	95.20	5.47	94.00	3.59	95.33	6.70	91.73	10.46	**96.00**	5.33	95.33	6.70
Wine	**97.01**	4.37	91.86	5.08	93.86	4.61	90.20	2.86	83.69	9.44	95.61	5.37	95.03	3.89
Balance-scale	75.49	4.97	68.36	5.30	**77.75**	6.31	77.14	4.93	58.38	7.76	73.42	5.66	76.50	3.51
Lymphography	**81.00**	10.35	75.51	9.59	77.23	10.91	76.59	12.20	77.78	12.77	78.84	11.49	78.43	14.30
Glass	69.13	8.66	65.52	9.26	62.03	9.80	71.16	10.54	39.23	11.34	69.00	8.70	**73.91**	8.43
Zoo	**95.60**	4.21	92.55	7.93	93.09	10.65	92.32	7.19	64.20	18.88	86.85	7.25	94.84	9.02
Primary	37.91	6.55	37.75	5.27	37.26	5.43	37.19	5.88	16.41	4.96	38.11	3.75	**38.36**	5.09
RANKING	**2.25**		4.78		4.39		4.11		6.08		3.06		3.33	

user, because it will probably be more difficult to harness the power of the algorithm. Thus, the first four parameters of GBAP are mandatory, and the other six parameters—enclosed into square brackets—are optional, having a default value.

For GBAP, the configuration considered in Table III was adopted after carrying out a cross-validation procedure over three data sets (primary-tumor, hepatitis and wine), using values from different ranks for each parameter, and then analyzing which specific set-up globally reported the best values. It is worth mentioning that no single combination of parameter values performed better for all data sets as expected. Nevertheless, notice that this adopted configuration should be tuned when classifying a particular data set.

V. RESULTS AND DISCUSSION

The performance and the understandability of the model proposed is compared to other classification algorithms. The aim of this section is to analyze statistically and interpret the experimental results obtained. Recall that in DM there is no classification algorithm that performs better than all others for every data set, as stated by the no free lunch theorem [56], [57].

A. Predictive accuracy analysis

A first evaluation criterion for the comparison is the predictive accuracy. Table IV shows average values for predictive accuracy with standard deviation. The best classification accuracies for each data set are highlighted in bold typeface. Analyzing the table, it is possible to realize that GBAP is competitive with respect to all the other algorithms considered, and also that it obtains the best results on 50% of the data sets used in the experimentation. In those data sets where GBAP does not reach the best results, its classification results are quite competitive. With regard to the standard deviation values, we can also observe that GBAP globally yields middling values in terms of stability.

Though GBAP obtains the best average accuracy values, we performed the Friedman test with the aim of comparing the results obtained and analyzing if there are significant differences between the classifiers. The Friedman test compares the average rankings of k algorithms over N datasets. Average rankings of all the algorithms considered are summarized at the bottom of Table IV. Looking at these ranking values, it can be noticed that the lowest ranking value, i.e., the best global position, is obtained by our proposal. The computed value for the Friedman statistic of average rankings distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom is 8.7404, which is greater than the tabled critical value at the $\alpha = 0.1$ significance level, $C_0 = [0, (F_F)_{0.1,6,102} = 1.8327]$. Thus, we reject the null-hypothesis that all algorithms perform equally well when $\alpha = 0.1$.

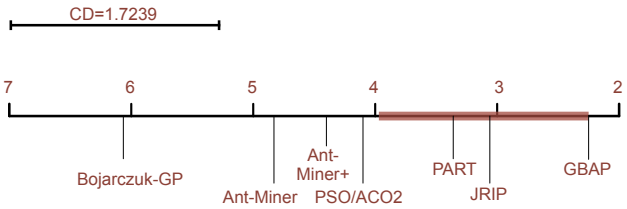

Fig. 3. Bonferroni–Dunn test. All classifiers whose ranks are outside the shaded interval have significant differences with respect to GBAP ($p < 0.1$)

Because of the rejection of the null-hypothesis by the Friedman test, we proceed with a post-hoc test to reveal the performance differences. Since all classifiers are compared with respect to a control classifier, we can apply the Bonferroni–Dunn test [58] focusing on all possible pairwise comparisons involving the GBAP algorithm. At the same significance level ($\alpha = 0.1$) the Bonferroni–Dunn critical value is 1.7239, which means that in order to be significant, the difference between any pair of means must be at least 1.7239 units [59]. Thus, the performance of GBAP is statistically better than those of the PSO/ACO2, Ant-Miner+, Ant-Miner

TABLE V
RULE SET LENGTH AND RULE COMPLEXITY COMPARATIVE RESULTS

	GBAP		ANT-MINER		ANT-MINER+		PSO/ACO2		GP		JRIP		PART	
Dataset	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R
Hepatitis	8.1	1.89	4.8	1.99	3.9	3.25	7.4	2.28	3.1	1.22	3.8	2.15	8.4	2.30
Sonar	12.3	1.81	5.2	2.07	4.0	3.48	6.1	2.92	3.0	1.00	4.6	2.21	13.9	2.98
Breast-c	13.2	1.91	6.0	1.28	5.4	2.82	11.8	1.75	3.5	1.01	3.3	1.70	17.1	2.12
Heart-c	14.5	1.67	5.9	1.20	4.4	2.82	11.9	3.81	3.0	3.02	5.3	2.32	17.3	2.35
Ionosphere	11.1	1.18	5.7	1.61	8.8	1.41	4.5	4.03	3.1	1.14	7.7	1.48	8.2	1.83
Horse-c	9.0	1.46	6.3	1.49	4.7	3.41	20.1	3.39	3.0	1.00	3.5	1.74	13.2	2.38
Australian	10.1	1.08	6.5	1.53	3.3	2.08	25.8	6.96	3.0	1.00	5.2	1.80	19.4	2.01
Breast-w	6.6	1.65	7.2	1.04	6.4	1.92	10.5	1.1	3.0	1.00	6.5	1.74	10.9	1.63
Diabetes	9.9	1.53	8.6	1.03	5.5	3.71	35.2	3.61	3.0	1.33	4.6	2.88	17.9	2.21
Credit-g	22.9	1.82	9.1	1.51	3.3	3.31	52.8	4.2	3.3	1.17	7.1	2.54	57.8	2.70
Mushroom	6.7	1.33	7.7	1.19	8.6	1.27	9.1	2.04	3.3	1.12	8.5	1.58	10.0	1.72
Iris	3.7	1.06	4.3	1.03	3.9	1.8	3.0	1.20	4.3	1.29	3.0	1.00	4.6	1.00
Wine	7.2	1.50	5.1	1.33	2.5	2.19	4.0	1.73	4.1	1.27	4.2	1.56	6.3	1.77
Balance-scale	16.7	1.92	12.4	1.01	9.1	3.35	27.0	2.69	5.2	1.56	12.4	1.84	28.6	1.55
Lymphography	10.2	1.60	4.7	1.69	4.6	2.83	15.6	2.11	5.1	1.02	6.9	1.53	10.2	2.30
Glass	21.6	1.79	8.4	1.76	12.4	4.10	24.5	3.13	8.2	1.48	8.0	2.03	13.7	2.32
Zoo	8.7	1.97	6.1	1.32	6.7	4.07	7.1	1.47	8.0	1.42	7.4	1.58	7.7	1.57
Primary	45.9	2.60	12.1	3.35	9.3	8.50	86.5	6.01	23.7	1.37	8.3	3.13	48.7	3.23
#R RANKING	5.30		3.67		2.69		5.25		**2.06**		2.72		6.31	
#C/R RANKING	3.22		2.5		6.33		5.55		**1.78**		3.86		4.75	

and Bojarczuk-GP algorithms, because the difference between their mean rank value and the mean rank of GBAP is greater than the mentioned critical value. These results are captured in Figure 3, where one can also see that GBAP achieves competitive or even better accuracy results than PART and JRIP.

Note that both at a significance level of $\alpha = 0.05$ and $\alpha = 0.01$, the Friedman test also rejects the null-hypothesis. In the first case, the Bonferroni–Dunn critical value is 1.8996, so that GBAP is significantly more accurate than Ant-Miner+, Ant-Miner and GP. At the $\alpha = 0.01$ significance level, the Bonferroni–Dunn critical value is equal to 2.2639 and, therefore, GBAP is significantly more accurate than Ant-Miner and GP. In both cases, GBAP is the control algorithm and its results are quite competitive or better than the results obtained by the other algorithms.

In order to contrast the results obtained after the application of the Bonferroni–Dunn's procedure, we can use the Holm test, which is more powerful than the first one and makes no additional assumptions about the hypotheses tested [58]. The advantage of the Bonferroni–Dunn test lies in the fact that it is easier to describe and visualize because it uses the same critical difference for all comparisons. In turn, the Holm test is a step-down post-hoc procedure that tests the hypotheses ordered by significance, comparing each p_i with $\alpha/(k-i)$ from the most significant p value. Table VI shows all the possible hypotheses of comparison between the control algorithm and the others, ordered by their p value and associated with their level of significance α. To contrast the results obtained by the Bonferroni–Dunn method, we applied the Holm test, which rejects those hypotheses that have a p value less or equal to 0.025. Thus, at a significance level of $\alpha = 0.05$, according to the Holm test and regarding to the predictive accuracy results, GBAP is statistically better than PSO/ACO2, Ant-Miner+, Ant-Miner and Bojarczuk-GP algorithms.

TABLE VI
HOLM TABLE FOR $\alpha = 0.05$

i	Algorithm	z	p	α/i	Hypothesis
6	GP	5.323465	1.0180E-7	0.008333	Rejected
5	ANT-MINER	3.510401	4.4743E-4	0.01	Rejected
4	ANT-MINER+	2.970339	0.002974	0.0125	Rejected
3	PSO/ACO2	2.584581	0.009749	0.016666	Rejected
2	PART	1.504457	0.132463	0.025	Accepted
1	JRIP	1.118699	0.263268	0.05	Accepted

B. Comprehensibility analysis

A second evaluation criterion is the comprehensibility of the knowledge acquired. In contrast to predictive accuracy, comprehensibility is a subjective concept, and it is frequently associated to the syntactical simplicity of the classifier [54]. Thus, the smaller the number of rules and the number of conditions appearing in them, the smaller the complexity of the classifier.

Table V summarizes both the classifier's rule set complexity, by the average number of rules found per data set, and the complexity of the rules, by the average number of conditions per rule. The last but one row of the table shows the average ranking value of each algorithm using the Friedman test with respect to the number of rules in the classifier, and the last row does the same for the number of conditions per rule. In both cases the control algorithm found is GP, as it has the lowest ranking value.

Before analyzing the results obtained, it is important to mention that all algorithms except GP extract rules in the same form, as a conjunction of conditions. However, GP employs the OR operator, and due to the tree-based enconding of individuals in GP, to compute fairly the number of rules and the number of conditions per rule, for each OR operator it is necessary to split the rule into two separate rules, without considering OR nodes as conditions.

The first statistical analysis is carried out considering the average number of rules in the output classifier. At a

significance level of $\alpha = 0.05$ the application of the Friedman test rejects the null-hypothesis, because the value of the statistic, 23.4734, does not belong to the critical interval $C_0 = [0, (F_F)_{0.05,6,102} = 2.1888]$. To show the significant differences we applied the post-hoc Bonferroni–Dunn test. The Bonferroni–Dunn's critical value is 1.8995 when alpha=0.05, which means that GP, JRIP and Ant-Miner+ are statistically better than GBAP. In turn, GBAP does not perform significantly worse than Ant-Miner, PSO/ACO2 and PART.

Regarding the number of rules in the output classifier, the best possible result would be to mine one rule per class, but this may not lead to good results when the distribution of instances per class is not located in a definite space region. This can be observed in the behavior of the GP algorithm, because it nearly extracts one rule per class and, therefore, it obtains the best results in this respect. Notice that in this algorithm, although OR nodes are not considered to be conditions but a way of joining two different rules predicting the same class, the algorithm tends to minimize this kind of operator, as it decreases substantially the simplicity component of the fitness function and, therefore, decreases the quality of the rules mined. In addition, this number of rules may not be enough for obtaining accurate results in many data sets, as it can be deduced looking at the accuracy results obtained by GP algorithm in Section V-A. In contrast, by using the niching algorithm described in Section III-F, GBAP ensures the selection of the number of rules that are necessary to cover the examples of each class, also achieving very good classification results.

The second statistical analysis involved the average number of conditions per rule measured. To check whether the algorithms present differences, we applied the Friedman test at the same significance level considered in the previous study, $\alpha = 0.05$. The F-distribution's statistic value is 22.0539, which neither belongs to the critical interval $C_0 = [0, (F_F)_{0.05,6,102} = 2.1888]$. Therefore, there are significant differences between the algorithms. The subsequent application of the Bonferroni–Dunn test revealed that GBAP performs significantly better than Ant-Miner+ and PSO/ACO2 in this aspect. Another conclusion of this test is that GBAP is not significantly better than GP, Ant-Miner, JRIP and PART, neither significantly worse than these algorithms, which is more important.

Regarding this measure, it should be pointed out that the use of a grammar in GBAP has a benefit because we can restrict the complexity of each rule by the number of derivations allowed for such grammar. Thus, we can arrange a trade-off between rule complexity and performance, reaching a compromise (longer rules may report better rules as they can discover more complex relationships between attributes). As seen in Table VII, the GBAP algorithm is the third-best algorithm in obtaining a small number of conditions per rule, only beaten by GP and Ant-Miner. The reason why the GP algorithm obtains the lowest values of conditions per rule may lie in the fact that this algorithm considers a simplicity component in the fitness function, and so the algorithm tries to minimize this factor. GBAP also takes into account the complexity of the rules in the reinforcement, as seen in Section III-E.

TABLE VII
AVERAGE RESULTS OF THE ALGORITHMS

ALGORITHM	ACCURACY	#R	#C/R
GBAP	**81.85**	13.41	1.65
ANT-MINER	79.25	7.01	1.52
ANT-MINER+	80.29	5.93	3.13
PSO/ACO2	80.63	20.16	3.02
GP	70.22	**5.16**	**1.30**
JRIP	80.99	6.13	1.93
PART	81.26	17.44	2.11

The trade-off between comprehensibility and accuracy is perfectly illustrated in the results obtained by the GP algorithm, as it is the most comprehensible algorithm; however it obtains the poorest accurate results. Despite, we can conclude by saying that the GBAP algorithm presents a good comprehensibility-accuracy trade-off, since it is the algorithm that presents the best ranking in accuracy, though it does not give rise to bad comprehensibility results, reaching quite competitive results in this sense, as shown before.

Finally, an example of a classifier obtained by GBAP on a training fold of the hepatitis data set is shown in Table VIII.

TABLE VIII
SAMPLE CLASSIFIER ON HEPATITIS DATA SET

```
IF (!= ALBUMIN (-inf,2.65] ) THEN LIVE
ELSE IF (AND (!= PROTIME (44.5,inf) )
             (= ASCITES yes) ) THEN DIE
ELSE IF (= ALBUMIN (-inf,2.65] ) THEN DIE
ELSE IF (AND (!= AGE (-inf,29] )
             (= VARICES yes) ) THEN DIE
ELSE IF (= ANTIVIRALS no) THEN DIE
ELSE IF (AND (!= PROTIME (44.5,inf) )
             (= ASCITES no) ) THEN DIE
ELSE IF (AND (= PROTIME (-inf,44.5] )
             (= SPIDERS no) ) THEN DIE
ELSE LIVE
```

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a novel ACO-based automatic programming algorithm guided by a CFG for multi-class classification. This algorithm, called GBAP, uses two complementary heuristic measures that conduct the search process for valid solutions, and offers as well the opportunity to the user to modify the complexity of the rules mined by simply varying the number of derivations allowed for the grammar. In addition, the niching algorithm developed, which is responsible for assigning a consequent to the rules and selecting the rules that make up the final classifier, avoids the disadvantages of sequential covering algorithms, because it neither removes nor rules out examples from the training data set.

Though GBAP had been originally designed for the DM classification task, it can also be applied to other kinds of problems, setting up another way of evaluating individuals and designing a suitable grammar for the subject problem.

We have compared GBAP with other representative rule-induction algorithms: three state-of-the-art algorithms (Ant-Miner, Ant-Miner+ and PSO/ACO2), a GP algorithm, and two

other industry standard classifiers (JRIP and PART) over eighteen different data sets. Non-parametrical statistical methods have been used to analyze the accuracy and comprehensibility of the algorithms to conclude, on the one hand, that GBAP is statistically more accurate than PSO/ACO2, Ant-Miner+, Ant-Miner and the GP algorithm at a significance level of 95%, and that GBAP is also competitive with JRIP and PART in terms of accuracy. On the other hand, comprehensibility results prove that GBAP is a competitive classifier in this sense, too. We consider these results promising, as they demonstrate that AP can be successfully employed to tackle classification problems, just as GP has demonstrated in previous research.

ACKNOWLEDGMENTS

This work has been supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN2008-06681-C06-03, and FEDER funds.

We would also thank the authors of Ant-Miner+ for kindly providing the source code of their algorithm.

REFERENCES

[1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kauffman, 2006.

[2] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas, "Machine learning: a review of classification and combining techniques," *Artificial Intelligence Reviews*, vol. 26, pp. 159–190, 2006.

[3] H.-J. Huang and C.-N. Hsu, "Bayesian classification for data from the same unknown class," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 32, no. 2, pp. 137–145, 2002.

[4] T.-M. Huang, V. Kecman, and I. Kopriva, "Support vector machines in classification and regression - an introduction," in *Kernel Based Algorithms for Mining Huge Data Sets: Supervised, Semi-supervised, and Unsupervised Learning (Studies in Computational Intelligence)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[5] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Pearson, 2009.

[6] S. U. Guan and F. Zhu, "An incremental approach to genetic-algorithms-based classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 35, no. 2, pp. 227–239, 2005.

[7] K. C. Tan, Q. Yu, C. M. Heng, and T. H. Lee, "Evolutionary computing for knowledge discovery in medical diagnosis," *Artificial Intelligence in Medicine*, vol. 27, no. 2, pp. 129 – 154, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/B6T4K-47RRWS9-2/2/5c8dfaf6e49d194b0c8ed6e2fd1b5117

[8] 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

[9] 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.

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

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

[12] 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.

[13] J. Fürnkranz, "Separate-and-conquer rule learning," *Artif. Intell. Rev.*, vol. 13, pp. 3–54, February 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=309283.309291

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

[15] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, pp. 29–41, 1996.

[16] D. Martens, B. Baesens, and T. Fawcett, "Editorial survey: swarm intelligence for data mining," *Mach. Learn.*, vol. 82, pp. 1–42, January 2011. [Online]. Available: http://dx.doi.org/10.1007/s10994-010-5216-5

[17] B. Liu, H. A. Abbass, and B. McKay, "Density-based heuristic for rule discovery with ant-miner," in *Proceedings of the 6th Australasia-Japan Joint Workshop on Intell. Evol. Syst.*, 2002, pp. 180–184.

[18] ——, "Classification rule discovery with ant colony optimization," in *IAT '03: Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*. Washington, DC, USA: IEEE Computer Society, 2003, p. 83.

[19] Z. Wang and B. Feng, "Classification rule mining with an improved ant colony algorithm," *LNAI*, vol. 3339, pp. 357–367, 2004.

[20] C. Chen, Y. Chen, and J. He, "Neural network ensemble based ant colony classification rule mining," *Innovative Computing, Information and Control, International Conference on*, vol. 3, pp. 427–430, 2006. [Online]. Available: http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/ICICIC.2006.477

[21] A. Chan and A. Freitas, "A new classification-rule pruning procedure for an ant colony algorithm," 2006, pp. 25–36. [Online]. Available: http://dx.doi.org/10.1007/11740698_3

[22] A. Chan and A. A. Freitas, "A new ant colony algorithm for multi-label classification with applications in bioinformatics," pp. 27–34, July 2006.

[23] J. Smaldon and A. A. Freitas, "A new version of the ant-miner algorithm discovering unordered rule sets," in *GECCO*, 2006, pp. 43–50.

[24] P. Jin, Y. Zhu, K. Hu, and S. Li, *Classification Rule Mining Based on Ant Colony Optimization Algorithm*. Springer, 2006, vol. 344, pp. 654–663. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-37256-1_82

[25] M. Galea and Q. Shen, "Swarm intelligence in data mining," *Swarm Intelligence in Data Mining*, pp. 75–99, 2006. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-34956-3_4

[26] S. Swaminathan, "Rule induction using ant colony optimization for mixed variable attributes," Master's thesis, 2006.

[27] D. Martens, M. De Backer, J. Vanthienen, M. Snoeck, and B. Baesens, "Classification with ant colony optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 651–665, 2007.

[28] F. Otero, A. A. Freitas, and C. Johnson, "cant-miner: An ant colony classification algorithm to cope with continuous attributes," *LNCS*, vol. 5217, pp. 48–59, 2008.

[29] C. Nalini and P. Balasubramanie, "Discovering unordered rule sets for mixed variables using an ant-miner algorithm," *Data Science Journal*, vol. 7, pp. 76–87, May 2008.

[30] F. E. Otero, A. A. Freitas, and C. G. Johnson, "A hierarchical classification ant colony algorithm for predicting gene ontology terms," in *EvoBIO '09: Proceedings of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 68–79.

[31] K. M. Salama and A. M. Abdelbar, "Extensions to the ant-miner classification rule discovery algorithm," in *Swarm Intelligence - Proc. 7th International Conference, ANTS 2010, LNCS*, vol. 6234. Springer, 2010, pp. 167–178.

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

[33] N. Holden and A. A. Freitas, "A hybrid PSO/ACO algorithm for discovering classification rules in data mining," *J. Artif. Evol. App.*, vol. 2008, pp. 2:1–2:11, January 2008. [Online]. Available: http://dx.doi.org/10.1155/2008/316145

[34] Y. Chen, B. Yang, and J. Dong, "Evolving flexible neural networks using ant programming and pso algorithm," in *ISNN (1)*, 2004, pp. 211–216.

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

[36] M. Boryczka, Z. J. Czech, and W. Wieczorek, "Ant colony programming for approximation problems," in *GECCO*, 2003, pp. 142–143.

[37] J. Green, J. Whalley, and C. Johnson, "Automatic programming with ant colony optimization," in *Proceedings of the 2004 UK Workshop on Computational Intelligence*, 2004, pp. 70–77.

[38] M. Boryczka, "Eliminating introns in ant colony programming," *Fundam. Inf.*, vol. 68, no. 1-2, pp. 1–19, 2005.

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

[40] 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.

[41] P. Whigham, "Grammatically biased genetic programming," in *Proceedings of the Workshop on Genetic Programming: from Theory to Real-World Applications*, 1995, pp. 33–41.
[42] N. Hoai and R. McKay, "A framework for tree adjunct grammar guided genetic programming," in *Proceedings of the Post-Graduate ADFA Conference on Computer Science (PACCS'01)*, 2001, pp. 93–99.
[43] 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. L. et al., Ed. 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
[44] 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.
[45] ——, "The uphill battle of ant programming vs. genetic programming," in *IJCCI*, 2009, pp. 171–176.
[46] 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
[47] F. Neumann, D. Sudholt, and C. Witt, "Computational complexity of ant colony optimization and its hybridization with local search," in *Innovations in Swarm Intelligence*, ser. Studies in Computational Intelligence, C. Lim, L. Jain, and S. Dehuri, Eds. Springer Berlin / Heidelberg, 2009, vol. 248, pp. 91–120. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04225-6_6
[48] 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.
[49] A. Geyer-Schulz, *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, ser. Studies in Fuzziness. Heidelberg: Physica-Verlag, 1995, vol. 3.
[50] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *EWSL-91*. Springer-Verlag, 1991, pp. 151–163.
[51] 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
[52] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml
[53] 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.
[54] C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz, "A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets," *Artificial Intelligence in Medicine*, vol. 30, pp. 27–48, 2004.
[55] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, "JCLEC: a java framework for evolutionary computation," *Soft Comput.*, vol. 12, no. 4, pp. 381–392, 2007.
[56] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Comput.*, vol. 8, no. 7, pp. 1341–1390, 1996.
[57] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, April 1997. [Online]. Available: http://dx.doi.org/10.1109/4235.585893
[58] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.
[59] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 2007.

Juan Luis Olmo was born in Cordoba, Spain, in 1984. He received a B.Sc. degree from the University of Cordoba in 2005, and a M.Sc. degree from the University Oberta of Catalonia, Barcelona, in 2007, both in Computer Science.

Since 2009, he has been with the Department of Computer Science and Numerical Analysis, the University of Cordoba, Spain, where he is currently working towards obtaining the Ph.D., as well as developing teaching and research tasks, with a grant from the regional government of Andalusia. His research interests include the application of evolutionary computation and swarm intelligence to data mining.

Juan Luis Olmo is a Member of the IEEE Computer, Computational Intelligence and Systems, Man and Cybernetics societies and the Association of Computing Machinery Special Interest Group on Genetic and Evolutionary Computation.

José Raúl Romero is currently an Associate Professor at the Department of Computer Science of the University of Cordoba, Spain. He received his Ph.D. in Computer Science from the University of Malaga, Spain, in 2007. He has worked as an IT consultant for important business consulting and technology companies for several years. His current research interests include the use of bio-inspired algorithms for data mining, the industrial use of formal methods, open and distributed processing and model-driven software development and its applications.

Dr. Romero is a member of IEEE, the ACM, and the Spanish Technical Normalization Committee AEN/CTN 71/SC7 of AENOR. He can also be reached at http://www.jrromero.net.

Sebastián Ventura was born in Cordoba, Spain, in 1966. He received the B.Sc. and Ph.D. degrees from the University of Cordoba, in 1989 and 1996, respectively.

He is currently Associate Professor in the Department of Computer Science and Numerical Analysis, the University of Cordoba, where he heads the Knowledge Discovery and Intelligent Systems Research Laboratory. He is the author or coauthor of more than 90 international publications, 30 of which have been published in international journals. He has also been engaged in eleven research projects (being the coordinator of two of them) supported by the Spanish and Andalusian governments and the European Union, concerning several aspects of the area of evolutionary computation, machine learning, and data mining and its applications. His current main research interests are in the fields of soft-computing, machine learning, data mining and its applications.

Dr. Ventura is a Member of the IEEE Computer, Computational Intelligence and Systems, Man and Cybernetics societies and the Association of Computing Machinery.