A Hybrid Ant Colony Optimization Algorithm for Software Project Scheduling

S.Jagadeesan, S.Gayathri
Assistant Professor, Final Year Student
Master of computer Application, Department of Computer Application,
Nandha Engineering College (Anna University), Erode-5,2Tamilnadu, India.

Abstract—*The extraction of comprehensible knowledge is one of the major challenges in many domains. In this concept, an ant programming (AP) framework, which is capable of mining classification rules easily comprehensible by humans, and, therefore, capable of supporting expert-domain decisions, is presented. The algorithm proposed, called grammar based ant programming (GBAP), 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, in-stead 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 are 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 Colony Optimization (ACO), Ant Programming (AP), classification, Data Mining (DM), Grammar-Based Automatic Programming (GBAP).

I. INTRODUCTION

DATA MINING (DM) involves the process of applying specific algorithms for extracting comprehensible, nontrivial and useful knowledge from data. The discovered knowledgeshould 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 for the values of certain other attributes (referred to as the predicting at-tributes). 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 unlabeled. 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 [9].

More recently, ant colony optimization (ACO) [7], [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.* [10], and it has become a bench-mark algorithm in this field. Since then, several extensions and modifications of this sequential covering algorithm have been presented.

ACO-based automatic programming [11]—which is another kind of automatic programming method that uses ACO as search technique, has never been explored to tackle classification problems.In this concept, 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 ben-efits of both ACO metaheuristic and automatic programming. 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 clas-sification algorithms, our proposal provides more expressive power, because the grammar allows to control several aspects related to comprehensibility, such as the definition of specific operators, the specification of the conditions that can appear in rule antecedents or how these conditions are connected. Moreover, our algorithm lacks the drawbacks of rule induction using sequential covering algorithms, as Ant-Miner, because it does not rule out examples when building the classifier. The remainder of this concept 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 meta-heuristic placed into swarm intelligence (SI) . SI is concerned with the development of multiagent 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 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 . The first algorithm that applied ACO to rule and func-tions, as GP does). Then, each program is evaluated and table is updated by evaporation and induction was Ant-Miner , and it has become the most referred 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 .

TABLE 1
MODIFICATIONS EXTENSIONS OF ANT MINER

Reference	Pruning	Pheromone	Heuristic	Transition rule	Fitness measure	Interval rules	Continuous attributes	Fuzzy	Multi-Label	Hierarchical
Liu et al. [18]			√							
Liu et al. [19]		√	√	√						
Wang, Feng [20]		√	√	√						
Chen et al. [21]	√	√	√							
Chan, Freitas [22]	√									
Chan, Freitas [23]					√				√	
Smaldon, Freitas [24]	√	√	√		√					
Jin et al. [25]	√	√								
Galea, Shen [26]		√	√					√		
Swaminathan [27]		√				√				
Martens et al. [28]	√	√	√		√	√				
Otero et al. [29]		√	√				√			
Nalini, Balasubramanie [30]	√	√	√		√		√			
Otero et al. [31]	√	√	√		√		√			√
Salama, Abdelbar [32]		√	√	√	√					

the original Ant-Miner. For example, Liu *et al.* [12] Many of these extensions imply minor changes, and the results obtained are slightly different from the ones obtained by 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 contrast, Ant-Miner+, proposed by Martens *et al.* [12], 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) and uses a more accurate class-specific heuristic function.

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 opti-mization (PSO)—ACO algorithm, PSO/ACO2, developed by Holden *et al.*, for the discovery of classification rules. PSO is another optimization technique positioned among SI, and 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.

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 prob-lems 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 [12], and it was closely related to GP. In fact, their algorithm started 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 func-tions, as GP does).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.

Boryczka*et al* 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 ex-pressions 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.* [38] 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.

More recently, Shirakawa*et al.* [13] proposed dynamic ant programming (DAP). Its main difference with regard 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 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. 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 cannot 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. 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 re-search area, which has been widely studied and analyzed by Neumann *et al.*

A. Environment and Rule Encoding

GBAP prescribes a CFG for representing the antecedent of the rule encoded by each individual.

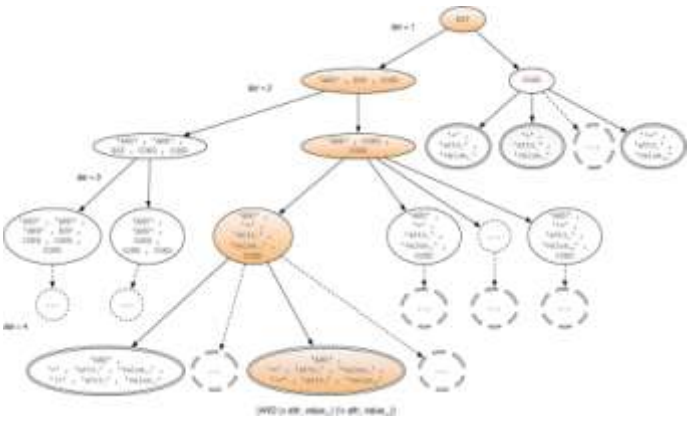

Fig. 1. Space of states at a depth of four derivations. The sample colored path represents the antecedent found by a given ant.

In an observation like in 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 represen-tation 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 with the 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, the environment is the search space com-prising 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 Fig. 1 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 colored path in Fig. 2. This path consists of 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 ants still have an internal memory to store the path to do an offline pheromone update.

Regarding the individual encoding, GBAP follows the *ant=rule* (i.e., *individual = rule*) approach [13]. As aforemen-tioned, when ants have been 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-D.

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 Fig. 1.

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* ← ((*maxDerivations* − 2)/ *numGenerations*)

4: *maxDerivations* ← 2
5: **for** *i* = 0 **to** *i* = *numGenerations* **inc** 1 **do**
6: Create list *ants* ← *{}*
7: **for** *j* = 0 **to** *j* = *numAnts* **inc** 1 **do**
8: *ant* ← Create new ant (see Procedure 2)
9: Store *ant*'s path states in the space of states
10: *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** *f itness* > *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* ← *maxDerivations*+ *derivationStep*
22: **end for**
23: Establish the default rule in the classifier
24: *predictiveAccuracy* ← Compute the predictive accu-racy 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.

Procedure 1 Ants Creation

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

Once all ants have been created, these ants along with the ants assigned to the classifier in the previous generation will compete in the niching algorithm. They will 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.

Afterwards, each ant created in this generation of the algo-rithm, 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 normaliza-tion process takes place. The maximum number of derivations is also incremented by the derivation step.

The creation process of a given ant is described in Procedure 1. 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 cannot be applied simultaneously. To distinguish which one applies GBAP, we need 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';).

D. Fitness Function and Consequent Assignment

The fitness function that GBAP uses in the training stage to conduct the search process is the Laplace accuracy. This measure was selected because it suits well to multiclass. 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.

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

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) . We have selected problems with a wide range of dimensionality with respect to the number of classes 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). Furthermore, the other involved the discretization of such data sets containing numerical attributes, by applying Fayyad and Irani's discretization algorithm [54]. The replacement of miss-ing 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 strati-fied tenfold cross-validation procedure, where we randomly split each data set into ten mutually exclusive partitions, P_1, . . . , P_{10}, containing approximately the same number ofinstances and the same proportion of classes present 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.

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 , which will be explained briefly next; and two well-known classi-fiers, JRIP—the Weka's implementation of the popular sequen-tial 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 min-ing 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 func-tions consisting both of logical operators (AND, OR) and re-lational operators (=,_=,≤, >). Bojarczuk-GP follows a mixed *individual = rule/ruleset*approach, where each individualencodes 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 each algorithm, excluding GBAP, its user-defined pa-rameters 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, the authors describe parameters such as α, β, *early stopping criterion*, or param-eters 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 pa-rameters in the algorithm's code to the values reported in Table III, but this could be also a disadvantage for a given expert 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 val-ues 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 .

A. Predictive Accuracy Analysis

A first evaluation criterion for the comparison is the pre-dictive accuracy. Table IV shows average values for predic-tive 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 it also 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* data sets. 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 free-dom is 8.7404, which is greater than the tabled critical value at the α= 0.1 significance level, C_0= $[0,(F_F)_{0.1,6,102}=1.8327]$. Thus, we reject the null-hypothesis that all algorithmsperform equally well when α= 0.1.

Because of the rejection of the null-hypothesis by the Friedman test, we proceed with a post-hoc test to reveal the per-formance differences. Thus, the performance of GBAP is statistically better than those

of the PSO/ACO2, Ant-Miner+, Ant-Miner 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 Fig. 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 α= 0.05 and α=0.01, the Friedman test also rejects the null-hypothesis. In thefirst 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 α= 0.01 significance level, the Bonferroni–Dunn critical value is equal to 2.2639 and, there-fore, 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.

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 [59]. The advantage of the Bonferroni–Dunn test lies in the fact that it is easier to de-scribe 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 V 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. Tocontrast 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

TABLE III

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	9.35	71.05	14.45	81.54	12.05	**84.64**	7.66
Sonar	**81.88**	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	**73.62**	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	6.89	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	88.83	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	4.48	85.30	4.12	85.48	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	2.84	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	98.36	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	**97.81**	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	**77.25**	6.31	77.14	4.93	58.38	7.36	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	7.83	93.09	10.65	92.32	7.19	64.20	18.88	86.85	7.25	94.84	9.02
Primary	37.81	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	

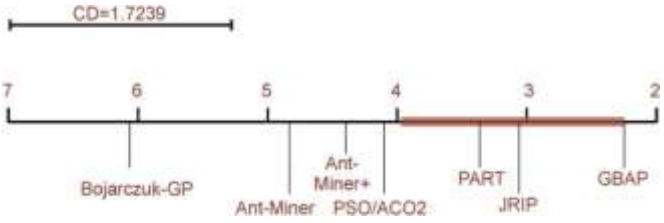

Fig. 2. Bonferroni–Dunn test. All classifiers whose ranks are outside the shaded interval have significant differences with respect to GBAP (p <0.1).

level of α= 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 IV

HOLM TABLE FORα= 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. Thus, the smaller the number of rules and the number of conditions appearing in them, the smaller the complexity of the classifier.

Table VI 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 encoding 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 av-erage number of rules in the output classifier. At a significance level of α= 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 thepost-hoc Bonferroni–Dunn test. The Bonferroni–Dunn's crit-ical 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 join-ing two different rules predicting the same class, the algorithm tends to minimize this kind of operator, as it decreases sub-stantially 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.

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	

TABLE VI

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

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 tradeoff, 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.

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 num-ber 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 previ-ous study, α= 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 significantdifferences 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, nei-ther 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

TABLE VII

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
```

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.

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

VI. CONCLUSIONS AND FUTURE WORK

In this concept, we have presented a novel ACO-based au-tomatic 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 has 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 com-prehensibility 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, compre-hensibility 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.

REFERENCES

[1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. San Mateo, CA: Morgan Kaufman, 2006.

[2] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas, "Machine learning: A review of classification and combining techniques," *Artif. Intell. Rev.*, vol. 26, no. 3, pp. 159–190, Nov. 2006.

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

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

[5] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ: Pearson, 2009.

[6] S. U. Guan and F. Zhu, "An incremental approach to genetic-algorithms-based classification," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 2, pp. 227–239, Apr. 2005.

[7] K. C. Tan, Q. Yu, C. M. Heng, and T. H. Lee. (2003, Feb.). Evolutionary computing for knowledge discovery in medical diagnosis. *Artif. Intell. Med.* [Online]. *27(2)*, pp. 129–154. Available: http://www.sciencedirect.com/science/article/B6T4K-47RRWS9-2/2/ 5c8dfaf6e49d194b0c8ed6e2fd1b5117

[8] M. Dorigo and T. Stützle, *The Ant Colony Optimization Meta-heuristic: Algorithms, Applications and Advances*, F. Glover and G. Kochenberger, Eds. Norwell, MA: Kluwer, 2002, ser. International Series in Operations Research and Management Science.

[9] M. Dorigo and T. Stützle, *The ant colony optimization metaheuristic: Algorithms, applications and advances*, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2000-32. [Online]. Available: ftp://iridia.ulb.ac.be/pub/mdorigo/tec.reps/TR.11-

MetaHandBook.pdf

[10] R. Parpinelli, A. A. Freitas, and H. S. Lopes, "Data mining with an ant colony optimization algorithm," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 321–332, Aug. 2002.

[11] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

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

[13] P. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 2, pp. 121–144, Mar. 2010.

[14] J. Fürnkranz. (1999, Jan.). Separate-and-conquer rule learning. *Artif. Intell. Rev.* [Online]. *13(1)*, pp. 3–54. Available: http://portal.acm.org/ citation.cfm?id=309283.309291

[15] E. Bonabeu, T. Eric, and M. Dorigo, *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford Univ. Press, 1999.