Soft Comput (2012) 16:2143–2163
DOI 10.1007/s00500-012-0883-8

ORIGINAL PAPER

Classification rule mining using ant programming guided by grammar with multiple Pareto fronts

J. L. Olmo · J. R. Romero · S. Ventura

Published online: 10 July 2012

Abstract This paper proposes a multi-objective ant programming algorithm for mining classification rules, MOGBAP, which focuses on optimizing sensitivity, specificity, and comprehensibility. It defines a context-free grammar that restricts the search space and ensures the creation of valid individuals, and its heuristic function presents two complementary components. Moreover, the algorithm addresses the classification problem from a new multi-objective perspective specifically suited for this task, which finds an independent Pareto front of individuals per class, so that it avoids the overlapping problem that appears when measuring the fitness of individuals from different classes. A comparative analysis of MOGBAP using two and three objectives is performed, and then its performance is experimentally evaluated throughout 15 varied benchmark data sets and compared to those obtained using another eight relevant rule extraction algorithms. The results prove that MOGBAP outperforms the other algorithms in predictive accuracy, also achieving a good trade-off between accuracy and comprehensibility.

J. L. Olmo · J. R. Romero · S. Ventura (✉)
Department of Computer Science and Numerical Analysis, University of Cordoba, 14071 Cordoba, Spain
e-mail: sventura@uco.es

J. L. Olmo
e-mail: jlolmo@uco.es

J. R. Romero
e-mail: jrromero@uco.es

Keywords Ant programming (AP) · Grammar-based automatic programming · Multi-objective ant colony optimization (MOACO) · Classification · Data mining (DM)

1 Introduction

Mining comprehensible knowledge from data has become one of the major challenges in many domains, since the quantity of data available is huge and still continues to increase rapidly. Thus, it becomes quite complicated to manage and analyze these data in order to help domain experts adopt decisions. Data mining (DM) algorithms allow the automatic extraction of comprehensible, non-trivial, and useful information from these data. One of the most studied DM tasks is the classification task (Rudokaite-Margelevičiene et al. 2006), which aims to induce a set of classification rules (a classifier) from a training set of labeled examples. Once the classifier is built, one can apply these rules to other uncategorized data to label each instance with one of the predefined classes. The performance of the classifier is typically measured with the accuracy obtained when applying the classifier to a separate test set.

Many diverse classification algorithms have been proposed. Among them, non-linear models such as support vector machines (Huang et al. 2006) or neural networks (Floreano et al. 2008), despite having demonstrated their capacity to obtain accurate classifiers, have the disadvantage of behaving like black boxes, and are lacking in terms of comprehensibility. On the contrary, logic-based algorithms, i.e., decision trees (Kapočite-Dzikiene et al. 2008) and rule-based methods (Lanzi 2008), have the advantage of being interpretable, which is a key issue since this can

help experts during the decision-making process, especially to support borderline decisions.

A metaheuristic (Blum et al. 2011; Blum and Roli 2003) that has been successfully applied to the extraction of rule-based classifiers is ant colony optimization (ACO) (Dorigo et al. 2002), which is a nature-inspired optimization metaheuristic based on the behavior and capabilities of self-organization that ant colonies exhibit in their search for food. The first application of ACO to classification was the Ant-Miner algorithm (Parpinelli et al. 2002), which has become a top algorithm in this field. Since its publication, many classification algorithms based on ACO have been proposed. Another related metaheuristic that has recently been proved to extract accurate rule-based classifiers is ant programming (AP) (Olmo et al. 2010), which is a kind of automatic programming (Koza et al. 1992) that uses ACO as its search technique.

The design of a rule-based classifier can also be tackled from the point of view of multi-objective optimization (MOO): unlike single-objective optimization problems, it aims to discover classification rules optimizing simultaneously several metrics, and where the improvement in the values of one metric often may be detrimental to the others. Usually, multi-objective classification algorithms seek to obtain classifiers with a good trade-off between comprehensibility and accuracy. It should be noted at this point that a classifier can be considered to be more comprehensible than another in terms of its size (if it presents less rules than the other) or in terms of the length of its rules (if it has a lower average number of conditions per rule). Several proposals using MOO for classification rule mining have been presented, for example using multi-objective evolutionary algorithms (Dehuri et al. 2008; Ishibuchi et al. 2007) or multi-objective particle swarm optimization (PSO) (Alatas et al. 2009; Torácio et al. 2009). Nevertheless, to the best of our knowledge, ant-based algorithms with the multi-objective property have never been employed for this purpose.

A core contribution of this paper is investigating the application of AP to classification from a multi-objective point of view, for the sake of obtaining accurate but also comprehensible classifiers, since MOO allows to look for a trade-off between several metrics. Hence, a new multi-objective AP algorithm for classification rule mining is presented. The algorithm is called Multi-Objective Grammar-based Ant Programming for classification (MOG-BAP), and it generates a classifier composed of IF–THEN rules. Our proposal combines the advantages of grammar-based AP with those inherent to MOO, focusing on achieving a good balance between accuracy and comprehensibility. Another important contribution lies in the fact that the multi-objective algorithm developed addresses the Pareto front discovery in a novel way, appropriate for the classification task, finding a separate Pareto front for each class available in the data set and then combining the solutions of each Pareto front by using a niching approach. A study of our proposal optimizing two and three objectives is carried out, the former focusing on maximizing sensitivity and specificity, and the latter also considering the optimization of a comprehensibility metric. The best one is compared to eight other rule induction algorithms from several paradigms. Then, the results obtained are analyzed statistically and show that our algorithm is significantly more accurate than most of the algorithms considered, also obtaining a good performance with respect to comprehensibility metrics and, therefore, reaching a good compromise between predictive accuracy and comprehensibility.

The remainder of this paper is organized as follows. In the next section, we introduce ACO as the primary technique of our algorithm, as well as AP. In Sect. 3, we describe the proposed algorithm. Section 4 explains the experiments carried out and the data sets used. Section 5 discusses the results obtained and, finally, some concluding remarks are outlined in Sect. 6.

2 Related work

First, this section focuses on introducing the ACO metaheuristic, explaining briefly the ACO algorithms devoted to classification that intervene in the experimental study. Then, it also introduces the AP metaheuristic, mentioning the algorithms published in the literature to date and their applications.

2.1 Ant colony optimization

ACO is an agent-based nature-inspired optimization metaheuristic belonging to the field of swarm intelligence (SI) (Bonabeu et al. 1999). 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. 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 with each other through the environment, in an indirect way, by means of a chemical substance—a pheromone—that they spray over the path they follow—a phenomenon known as *stigmergy*. The pheromone concentration in a given path

increases as more ants follow this path and decreases more quickly as ants fail to travel it, since the evaporation in this path becomes greater than the reinforcement. The higher 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 (Dorigo et al. 1996), finding optimal or near optimal solutions. Since then, ACO algorithms have been used in an increasing range of problem domains and have also been shown to be effective when tackling the classification task of DM. The first algorithm that applied ACO to rule induction was Ant-Miner (Parpinelli et al. 2002), 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 the algorithm 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 a transition rule and it only considers including terms that have not been previously chosen. It keeps adding new terms to build the rule's 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. A theoretic information 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 ith attribute and V_{ij} is the jth 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 heuristic function for term_{ij} which depends on the problem; τ_{ij} is the amount of pheromone associated with 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 A_i has not yet been selected 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:

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

where T_P stands for the true positives, which are positive instances correctly classified as positives; F_P stands for the false positives, i.e., negative examples erroneously labeled as positives; T_N are the true negatives, i.e., negative instances correctly identified as negatives; and F_N are the false negatives, i.e., positive examples incorrectly classified as negatives (Cios et al. 2010).

From the publication of Ant-Miner onward, other research works have followed the research lines suggested in Parpinelli et al. (2002). They focus on using different mechanisms for pruning, pheromone updating, or heuristic functions, or they are designed to include interval rules, dealing with continuous attributes, extracting fuzzy classification rules, or being applied to multi-label or hierarchical classification. However, most of these extensions only implement minor changes, and the results obtained are only slightly different from the ones obtained by the original Ant-Miner. Later, two modifications of Ant-Miner that have reported superior accuracy results will be discussed and, therefore, those approaches are also included in the experimental study presented in this paper.

The first extension considered here is Ant-Miner+, proposed by Martens et al. (2007), which follows the max–min ant system (Stützle et al. 2000) philosophy. The authors devised the environment as a directed acyclic graph, stating that it allows of selecting transitions more adequate than those selected by the original Ant-Miner algorithm. In this version, the inclusion of interval rules is permitted and the original heuristic function is replaced by a more accurate class-specific heuristic. Ant-Miner+ measures the quality of individuals by summing their confidence and their coverage, as shown in Equation 3:

$$\begin{aligned} \text{fitness} &= \text{confidence} + \text{coverage} \\ &= \frac{|\, A \cup C \subseteq I, I \in D \,|}{|\, A \subseteq I, I \in D \,|} + \frac{|\, A \cup C \subseteq I, I \in D \,|}{|\, D_{\text{cov}=0} \,|} \end{aligned} \tag{3}$$

where confidence refers to the quotient of the number of instances I belonging to the data set D that include both the antecedent A and the consequent C by 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 to as $|D_{\text{cov}=0}|$.

Otero et al. (2008) presented another ACO algorithm for classification, called cAnt-Miner, capable of handling numerical attributes directly by creating discrete intervals

for continuous attributes internally, during the execution of the algorithm. Later, they also introduced a modification of this algorithm in (Otero et al. 2009), where they incorporated the minimum description length (MDL) (Barron et al. 1998) principle in the rule construction process, and the pheromone values in the construction graph were associated with edges instead of the vertices, used by both the previous version and Ant-Miner itself. cAnt-Miner2-MDL significantly outperformed Ant-Miner in most of the data sets tested, and this version is used for the experiments carried out in this paper.

There are also other extensions related to the hybridization of ACO with other metaheuristics. Among them, we appreciate the PSO/ACO2 algorithm, a hybrid algorithm developed by Holden et al. (2008) 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:

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

Since the focus of this section has been on the ACO-based algorithms used in the experimental study, we refer the reader to the recent survey of SI approaches to DM, written by Martens et al. (2011), where a deeper description of the ACO algorithms proposed to date can be found. It is also relevant to mention at this point the work published by Salama et al. (2011), where five extensions to the Ant-Miner classification rule discovery algorithm were presented, and an experimental study was performed to analyze their performance and compare them to three other traditional rule induction algorithms.

It is worth mentioning that there are no existent ACO proposals for classification that address this problem from a multi-objective point of view, since the current proposals usually employ, as their fitness function, simple measures or a scalar aggregation of them (e.g., the fitness measure used by Ant-Miner, as presented in Eq. 2). Nevertheless, multi-objective ACO (MOACO) algorithms have been used to address other kinds of problems rather than classification with great success. An excellent review and analysis paper on the subject can be found in Angus et al. (2009).

2.2 Ant programming

AP is an approach that follows the fundamentals of automatic programming for constructing computer programs by using ACO as the search technique, finding near-optimal solutions to optimization problems in a reasonable computational time. Similarly to other automatic programming techniques, such as genetic programming (GP), a program or individual in AP can be represented by using a tree structure, where the nodes represent functions, variables, and constants, and the closure and sufficiency properties should be satisfied (Koza et al. 1992). Several proposals use a graph structure instead of a tree structure, as described next, and others make use of a grammar that guides the search for solutions and the generation of the states. Hereafter, we review the most relevant proposals published to date using AP.

The work by Roux and Fonlupt (2000), where they combined the ACO paradigm with the automatic generation of programs, involved the first approach to AP. They implemented an AP algorithm for solving symbolic regression and multiplexor problems. Their algorithm first initializes a population of programs (trees) at random, by 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 (named terminals and functions). 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.

Chen et al. (2004) followed a similar approach, in which individuals build and modify trees taking into account the quantity of pheromone at each node, where a pheromone table is stored. The authors combined this approach with PSO, AP being responsible for evolving the architecture of flexible neural networks, and PSO being in charge of optimizing the parameters encoded in the neural tree. Then, the developed flexible neural networks were applied to a time-series prediction problem, showing the effectiveness of their algorithm.

Another application of AP to symbolic regression problems was presented by Boryczka and Czech (2002) and Boryczka et al. (2003), calling their method ant colony programming (ACP). Two different versions of ACP, known as the expression approach and the program approach, were presented. Green et al. (2004) simultaneously proposed an AP technique similar to the ACP expression approach, which basically consists in generating expressions in prefix notation from the path followed by a given ant in the graph that represents the environment. Here, nodes in the graph can refer to either a variable or an operator, and the pheromone values are assigned to the edges in the graph. In contrast, in the program approach, the nodes in the graph represent

assignment instructions, and the solution is made up from a set of nodes, therefore representing a sequence of assignments that evaluate the function. With the motivation of improving the evaluation performance, different extensions to this work were presented by Boryczka (2005, 2008).

Abbass et al. (2002) proposed AntTAG, an automatic programming method employing ACO as search strategy and a tree adjoining grammar (TAG) to build programs. TAGs are compact context-sensitive grammars that use tree manipulation operations for syntactic analysis and can distinguish between derivation and derivation trees as well. The authors tested their performance on symbolic regression problems.

Keber and Schuster published another algorithm guided by grammar called generalized ant programming (GAP) (Keber et al. 2002), which uses a context-free grammar (CFG) (Mernik et al. 2004) instead of a TAG, and where ants generate a program by following a path in the space of states. Salehi-Abari and White (2008) worked on GAP, proposing a variation of the algorithm called enhanced generalized ant programming (EGAP). More specifically, they introduced a new pheromone placement method that tends to put an amount of pheromone in a derivation step that is proportional to the depth of the path; it also employs a specific heuristic function to control the path termination. An extension to this work (Salehi-Abari and White 2009) was also published, comparing the performance of GP to that of their EGAP algorithm in three different problems: quarctic symbolic regression, multiplexer, and Santa Fe ant trail.

More recently, Shirakawa et al. (2008) 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 to that of GP using symbolic regression problems.

A very recent contribution to the AP field has been the grammar-based ant programming (GBAP) (Olmo et al. 2011) algorithm. It uses a CFG to generate the space of states and to create new valid individuals. This algorithm implemented, as far as we know, the first application of AP to the classification task of DM. The fitness function used by GBAP during the training stage to measure the quality of individuals is the Laplace accuracy, defined as:

$$\text{Laplace accuracy} = \frac{1 + T_P}{k + T_P + F_P} \quad (5)$$

where k represents the number of classes available in the data set.

An interesting point of the algorithm is how it assigns a consequent to the rules: for each individual created in the current generation of the algorithm, k fitness values are computed (one for each class in the data set), and then a parallel niching procedure is carried out where the k fitness values are degraded depending on the diversity of the respective solutions. Finally, the class assigned as consequent is the one that reports the best adjusted fitness value.

The algorithm proposed in the present paper takes GBAP as its starting point, aiming to improve the successful results obtained by GBAP by following a multi-objective approach to reach a better trade-off between the different objectives.

Finally, to complete chronologically the list of contributions to the AP field, it is worth mentioning Kumaresan's work (Kumaresan 2011), where AP is applied to control and modeling. In that proposal, the space of states is represented by means of a graph where the nodes represent functions and terminals and the edges are weighted by pheromone. To obtain optimal control, AP is used to solve differential algebraic equations to compute the solution of the matrix Ricatti differential equation, which is the central issue in optimal control theory. The author shows that the solution obtained with AP is very close to the exact solution of the problem.

3 Description of MOGBAP

This section presents the MOGBAP algorithm, which stands for Multi-Objective Grammar-Based Ant Programming algorithm for classification.

The particular characteristics of the algorithm are described in the following subsections, where it is explained how the environment is constructed, the generation and encoding of individuals, the heuristic measures, the pheromone reinforcement, the fitness objectives considered, and the multi-objective approach followed. Finally, the pseudocode of the algorithm is provided.

3.1 Environment, individual generation, and rule encoding

In MOGBAP, the environment that permits ants to communicate indirectly with each other is the derivation tree that can be generated from the grammar, as shown in Fig. 1. Therefore, ants can only visit valid states, enforcing any solution found to be valid too. This grammar is expressed in Backus–Naur form, and is defined in Fig. 2. Notice that it can be adapted to other specific problems, for instance by adding other logical operators such as not equal

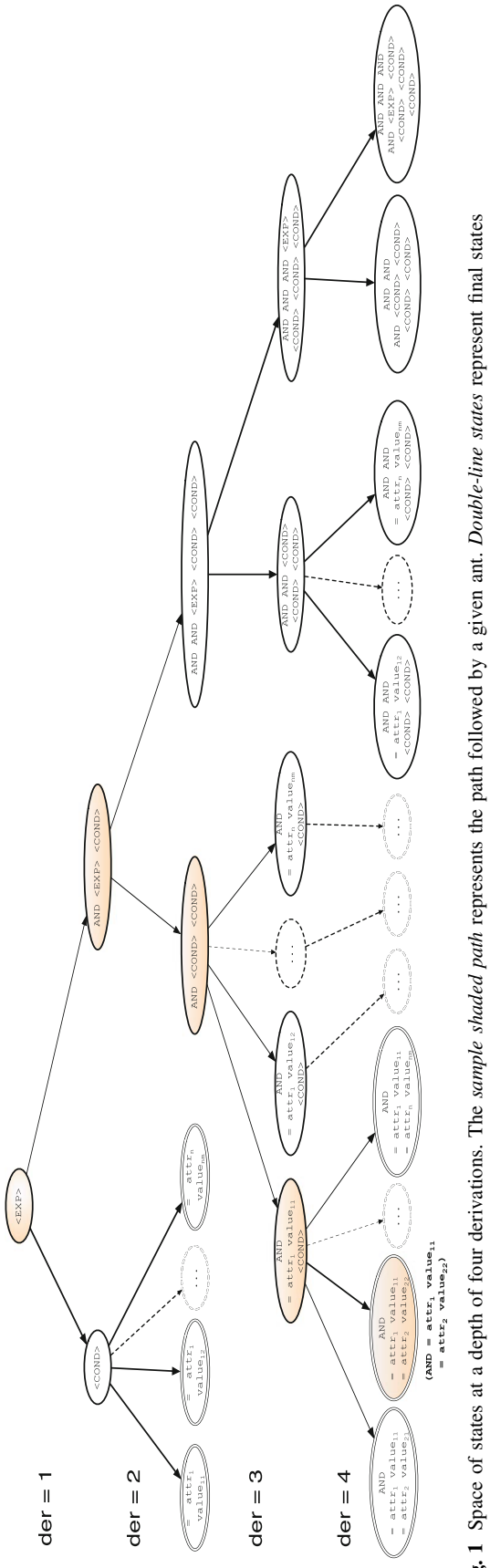


Fig. 1 Space of states at a depth of four derivations. The *sample shaded path* represents the path followed by a given ant. *Double-line states* represent final states

$$
\begin{aligned}
G &= (\Sigma_N,\ \Sigma_T,\ P,\ \texttt{<EXP>})\\
\Sigma_N &= \{\texttt{<EXP>},\ \texttt{<COND>}\}\\
\Sigma_T &= \{\texttt{AND},\ \texttt{=},\\
&\qquad \texttt{attr}_1,\ \texttt{attr}_2,\ \ldots,\ \texttt{attr}_n,\\
&\qquad \texttt{value}_{11},\ \texttt{value}_{12},\ \ldots,\ \texttt{value}_{1m},\\
&\qquad \texttt{value}_{21},\ \texttt{value}_{22},\ \ldots,\ \texttt{value}_{2m},\\
&\quad \ldots,\ \texttt{value}_{n1},\ \texttt{value}_{n2},\ \ldots,\ \texttt{value}_{nm}\}\\
P &= \{\texttt{<EXP> := <COND> | AND <EXP> <COND>},\\
&\qquad \texttt{<COND> := all possible valid}\\
&\qquad\qquad \texttt{combinations of the ternary}\\
&\qquad\qquad \texttt{= attr value}\}
\end{aligned}
$$

Fig. 2 Context-free grammar used in MOGBAP

or the disjunctive operator. It is a four-tuple, $G = (\Sigma_N, \Sigma_T, P, \langle \text{EXP} \rangle)$, where Σ_N is the set of non-terminal symbols, Σ_T is the set of terminal symbols, P is the set of production rules, and $\langle \text{EXP} \rangle$ stands for the start symbol. Note that any production rule is composed of two parts. The first one is the left-hand side, which always refers to a non-terminal symbol. This non-terminal symbol might be replaced by the second part, the right-hand side of the rule, which consists of a combination of terminal and non-terminal symbols. Production rules are internally implemented in prefix notation and should always be derived from the left. This implies that each transition from a state i to another state j is triggered after applying a production rule to the first non-terminal symbol of state i. This design decision was taken because of performance reasons, in order to save on computational costs when assessing rule quality.

The environment comprises all possible expressions or programs that can be derived from the grammar in a given maximum number of derivations, and it adopts the shape of a derivation tree, as shown in Fig. 1. The initial state corresponds to the start symbol of the grammar. A path over the environment corresponds to the states visited by any ant until reaching a feasible solution. The last state of a path corresponds to a final state or solution, comprising only terminal symbols. Thus, it directly represents the expression of the encoded antecedent, which can be evaluated. Fulfilling the properties of an artificial ant (Mullen et al. 2009), each ant stores the path explored in order to do an offline pheromone update.

Concerning the individual encoding, MOGBAP follows the ant = rule (i.e., individual = rule) approach (Espejo et al. 2010). It is worth noting that when a given ant reaches a final state, the ant just represents the antecedent of the rule. MOGBAP assigns later a consequent to the ant, which corresponds to the most frequent class among the training instances covered by this antecedent.

3.2 Heuristic measure

A difference between MOGBAP and typical ACO algorithms is the employment of a heuristic function which considers two complementary components that cannot be applied at the same time. To distinguish which metric can be applied, first it is necessary to identify the kind of transition involved. Two cases are considered: final transitions, if the transition involves the application of a production rule that selects an attribute of the problem domain, and intermediate transitions, if otherwise.

In the case of intermediate transitions, a measure associated with the cardinality of the production rules is considered. It is referred to as P_{card}, and increases the likelihood of selecting transitions that may lead a given ant to a greater number of candidate solutions. It is based on the cardinality measure proposed by Geyer-Schulz et al. (1995). Thus, given a state i having k subsequent states, j being a specific successor among those k states, and where d derivations remain available, $P^k_{\text{card\_}ij}$ is the ratio between the number of candidate solutions that can be successfully reached from state j in $d-1$ derivations, and the sum of all possible candidate solutions that can be reached from the source state i in d derivations (see Eq. 6). If this measure is applied in final transitions, it would not be meaningful, since each possible destination node would embrace the same number of solutions.

$$P_{\text{card}^k_{ij}} = \frac{\text{cardinality}\,(\text{state}_j, d-1)}{\Sigma_{k\in\text{allowed}}(\text{cardinality}\,(\text{state}_k, d-1))} \tag{6}$$

In contrast, the component considered for final transitions is the information gain ($G(A_i)$) (Parpinelli et al. 2002). It measures the worth of each attribute in separating the training examples with respect to their target classification; it is computed as:

$$G(A_i) = I - I(A_i) \tag{7}$$

where I is the entropy of the classes in the training set (see Eq. 8), and $I(A_i)$ is the entropy of the classes given the values for attribute A_i (see Eq. 9). In turn, these are computed as follows:

$$I = -\sum_{c=1}^{\#\text{classes}} \left(\frac{n_c}{n} \cdot \log_2 \frac{n_c}{n}\right) \tag{8}$$

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

$$I(A_i) = \sum_{j=1}^{\#\text{values}A_i} \left(\frac{n_{ij}}{n} \cdot I(A_{ij})\right) \tag{9}$$

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 value j for attribute A_i, computed as:

$$I(A_{ij}) = -\sum_{c=1}^{\#\text{classes}} \left(\frac{n_{ijc}}{n_{ij}} \cdot \log_2 \frac{n_{ijc}}{n_{ij}}\right) \tag{10}$$

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

3.3 Solution construction

Any ACO-based algorithm follows a probabilistic stepwise solution construction method. Thus, in the MOGBAP algorithm, a given ant constructs a rule incrementally by following a sequence of steps or transitions. These transitions are biased by some information considered in the transition rule, which defines the probability with which a given ant may follow each transition. Therefore, for a given ant at the state i, the probability of taking the transition that leads to state j is defined as:

$$P^k_{ij} = \frac{(\eta_{ij})^{\alpha}(\tau_{ij})^{\beta}}{\Sigma_{k\in\text{allowed}}(\eta_{ik})^{\alpha}(\tau_{ik})^{\beta}} \tag{11}$$

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 $G(A_i) + P_{\text{card}}$, and τ indicates the strength of the pheromone trail.

Notice that a probability value is assigned to each available next state, but the number of transitions that can be followed is limited by the user-defined parameter *maxDerivations*. Thus, if moving to a given state does not allow any solution to the problem to be found in the number of derivations remaining available at that point, the algorithm will assign a probability of zero to this state and it will never be selected.

3.4 Pheromone initialization, evaporation, and reinforcement

In general, MOACO algorithms fall into two categories according to how the pheromone information is stored (Angus et al. 2009; García-Martínez et al. 2007). This way, MOACO algorithms can either use just one pheromone matrix, or a pheromone matrix for each objective considered. MOGBAP belongs to the first group, which entails a benefit regarding memory and computational time requirements (Angus et al. 2009).

Reinforcement and evaporation are the two operations considered with regard to pheromone maintenance. Evaporation takes place over the complete space of states:

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

where ρ represents the evaporation rate.

In turn, concerning pheromone reinforcement, only non-dominated ants are able to retrace their path to update the

amount of pheromone in the transitions followed. For a given individual, all transitions in its path are reinforced equally, and the value of this reinforcement is based upon the length and the quality of the solution encoded, represented by the Laplace accuracy:

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot Q \cdot \text{Laplace accuracy} \quad (13)$$

where τ_{ij} represents the amount of pheromone in the transition from state i to 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, and it is calculated as the ratio between the maximum number of derivations in the current generation and the length of the path followed by the ant (thus, shorter solutions will receive more pheromone).

When the pheromone updating operations have finished, a normalization process takes place, to bind the pheromone level in each transition to the range $[\tau_{\min}, \tau_{\max}]$. In addition, at the first generation of the algorithm, all transitions in the space of states are initialized with the maximum pheromone amount allowed.

3.5 Multi-objective fitness assessment

The quality of the individuals generated in MOGBAP is assessed on the basis of three conflicting objectives: sensitivity, specificity, and comprehensibility.

Sensitivity and specificity are two measures widely employed in classification problems, even as a scalar function of them, as seen in the fitness measure of Ant-Miner (see Eq. 2). Sensitivity indicates how well a rule identifies positive cases. On the contrary, specificity reports the effectiveness of a rule's identifying negative cases or those cases that do not belong to the class studied. If the sensitivity value of a rule is increased, it will predict a greater number of positive examples, but sometimes at the expense of classifying as positives some cases that actually belong to the negative class. Both objectives are to be maximized.

$$\text{Sensitivity} = \frac{T_P}{T_P + F_N} \quad (14)$$

$$\text{Specificity} = \frac{T_N}{T_N + F_P} \quad (15)$$

Since MOGBAP is a rule-based classification algorithm, it is intended to mine accurate but also comprehensible rules. So, somehow, it should also optimize the complexity of the rules mined. Although comprehensibility is a sort of subjective concept, there are several ways to measure the comprehensibility of the rules and the classifier, usually by counting the number of conditions per rule and the number of rules appearing in the final classifier. The latter can not be considered here as an objective, since MOGBAP follows the ant = rule approach, as mentioned in Sect. 3.1. On the other hand, if the number of conditions per rule is directly used as the comprehensibility metric, it should be minimized. Nevertheless, assuming that a rule can have up to a fixed number of conditions, comprehensibility can be measured as:

$$\text{Comprehensibility} = 1 - \frac{\text{numConditions}}{\text{maxConditions}} \quad (16)$$

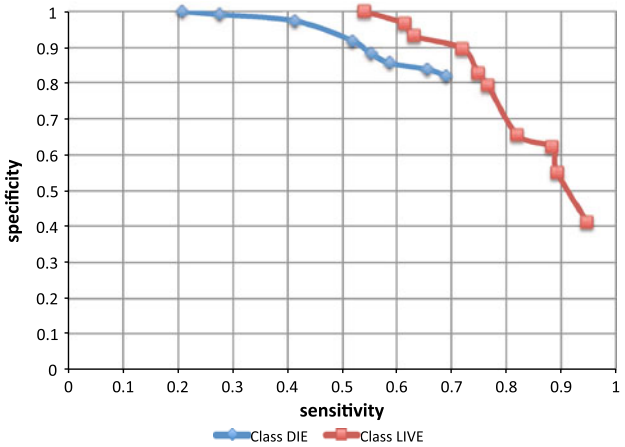
where numConditions refers to the number of conditions appearing in the rule encoded by the individual, whereas maxConditions is the maximum number of conditions that a rule can have (Dehuri et al. 2008).

In MOGBAP, it is easy to compute the maximum number of conditions that an individual can have, because the grammar is known beforehand and the maximum number of derivations allowed is also known. The advantage of using this comprehensibility metric lies in the fact that its values will be contained in the interval [0,1], and the closer its value to 1, the more comprehensible will be the rule. Hence, just as with the objectives of sensitivity and specificity, this objective, too, should be maximized. In Sect. 5.1, two versions of MOGBAP are analyzed, one optimizing both sensitivity and specificity, and the other considering simultaneously the three objectives of sensitivity, specificity, and comprehensibility. Finally, in the case of these three objectives, a given rule ant_i is said to dominate another rule ant_j, written $\text{ant}_i \succ \text{ant}_j$, if ant_i is not worse than ant_j in two objectives and is better in at least one objective.

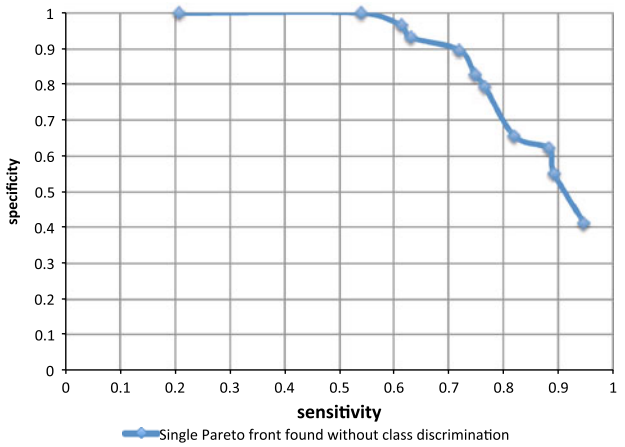
3.6 Multi-objective strategy

MOGBAP follows a multi-objective strategy that has been specially designed for the classification task. The idea behind this scheme is to distinguish solutions in terms of the class they predict, because certain classes are more difficult to predict than others. Actually, if individuals from different classes are ranked according to Pareto dominance, overlapping may occur, as illustrated in Figs. 3, 4 and 5, which show the Pareto fronts found after running MOGBAP for the hepatitis, breast-cancer and lymphography data sets, respectively, considering the optimization of sensitivity and specificity.

For instance, for the hepatitis problem, if a classic Pareto approach were employed, a single front of non-dominated solutions would be found, as shown in Fig. 3b. Hence, among the individuals represented in this figure, such a Pareto front would consist of all the individuals that predict the class 'LIVE' and just one individual of the class 'DIE' (the individual which has a specificity of 1.0). For the remaining individuals of the class 'DIE' to be considered, it would be necessary to find additional fronts, and they would have less likelihood of becoming part of the classifier decision list. On the other hand, the multi-

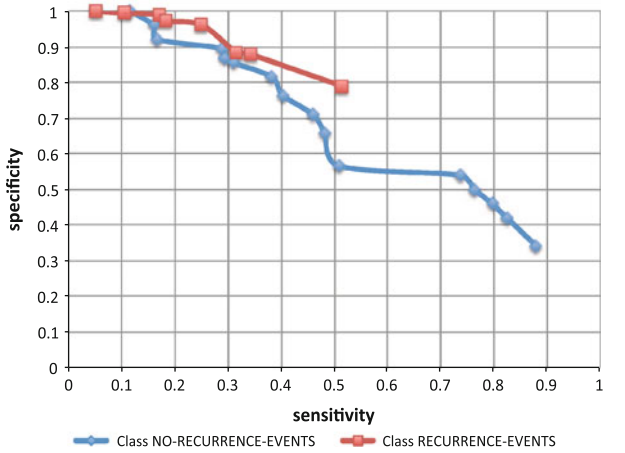


(a) Pareto fronts found for the two-class data set hepatitis following the proposed strategy

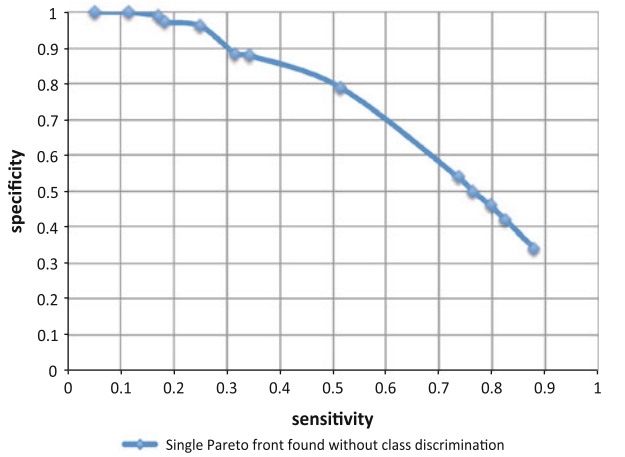


(b) Pareto front made up for the data set hepatitis with a classic approach

Fig. 3 Comparison between the *k*-Pareto fronts approach and a classic approach for the data set hepatitis



(a) Pareto fronts found for the two-class data set breast cancer following the proposed strategy



(b) Pareto front made up for the data set breast cancer with a classic approach

Fig. 4 Comparison between the *k*-Pareto fronts approach and a classic approach for the data set breast cancer

objective approach of MOGBAP (see Fig. 3a) guarantees that all non-dominated solutions for each available class will be found, so it ensures the inclusion of rules predicting each class in the final classifier.

Similar behavior could be argued when looking at Fig. 4, where many non-dominated solutions of the class 'NO-RECURRENCE-EVENTS' could be hidden if solutions for both classes are considered at the same time and, therefore, just a single Pareto front were to be found.

Moreover, considering now the lymphography data set as depicted in Fig. 5, in the case of a classic approach, the Pareto front would consist in just one point with a sensitivity and a specificity of 1.0. Moreover, to contemplate rules predicting the class 'MALIGN_LYMPH', it would be necessary to find at least three fronts—since the figure only shows the solutions belonging to the Pareto front for each class and, therefore, it hides solutions of other classes

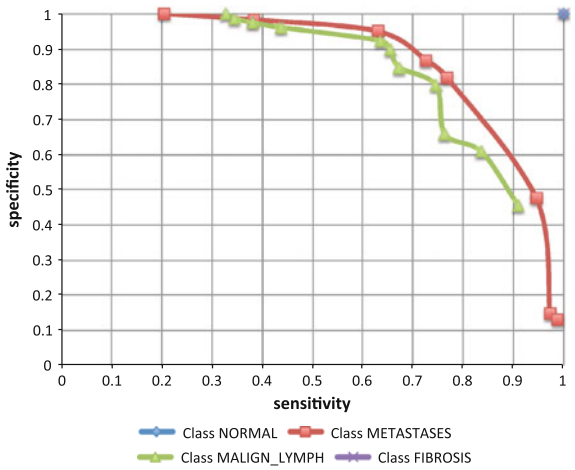


Fig. 5 Pareto fronts found for the four-class data set lymphography following the proposed strategy

rather than the target one that conforms to other intermediate fronts.

Roughly speaking, the multi-objective approach devised for MOGBAP consists in discovering a separate set of non-dominated solutions for each class in the data set. To this end, once individuals of the current generation have been created and evaluated for each objective considered, they are divided into k groups, k being the number of classes in the training set, according to their consequent class. Then, each group of individuals is combined with the solutions kept in the corresponding Pareto front found in the previous iteration of the algorithm, to rank all of them according to dominance, finding a new Pareto front for each class. Hence, there will be k-Pareto fronts, and only the non-dominated solutions contained will participate in the pheromone reinforcement.

The final classifier is made up of the non-dominated individuals that exist in each one of the k-Pareto fronts once the last generation of the algorithm finishes. To select the rules of the classifier appropriately, a niching procedure is carried out over each frontier, as described subsequently in Sect. 3.7.

The decision of addressing the multi-objective evaluation through considering a separate Pareto front for each class was adopted after comparing its performance with a dominance ranking procedure similar to that followed by the NSGA-II algorithm (Deb et al. 2000). The results achieved proved that the devised k-Pareto fronts approach outperformed the one used by NSGA-II, where a single Pareto front is found and, therefore, it can contain individuals predicting different classes. This multi-objective approach can also be considered as an online Pareto archival strategy for two reasons. Firstly, because non-dominated solutions are used once all the generations take place to select the individuals that make up the final classifier. Secondly, because Pareto optimal solutions are used during the algorithm execution to update pheromone rates in the environment.

3.7 Algorithm

The main steps of MOGBAP are given in the pseudocode of Algorithm 1. It begins by starting up the grammar and initializing the space of states with the initial state. It also creates an empty object that represents the classifier and also an empty hash table for storing the Pareto fronts found in each generation. The algorithm starts with the minimum number of derivations 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.

A new hash table ants is initialized when a new generation starts. This table has as many entries as there are classes in the data set, each one mapping to a new list of ants that will contain the ants predicting the corresponding class that will be created in this generation. Then, the algorithm creates numAnts individuals and assigns each of them a consequent. The creation process of a new individual is shown in Procedure 2: a new list path is created, inserting there the initial state. From this one, new states are added to path as the ant moves through the space of states following the transition rule described in Eq. 11. If the transition selected by the ant leads to a final state, the procedure finishes and the path followed by the ant is returned. Notice that Procedure 2 only returns the path followed by a given ant, i.e., the antecedent of the rule. The consequent is assigned by computing the percentage of instances of each available class covered by this antecedent, and assigning the class more covered in proportion. The algorithm also evaluates each new individual by the objective functions considered. Once the individual has been evaluated, it is added to the corresponding list of ants that predict the same consequent, which is stored in the hash table ants.

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 transition 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

Algorithm 1 Pseudocode of MOGBAP

Require: $numIterations, numAnts, maxDerivations, categories$

Initialize grammar and space of states
Create an empty classifier
Hash table $paretoFronts \leftarrow \{\}$
$derivationStep \leftarrow \frac{maxDerivations-2}{numIterations}$
$maxDerivations \leftarrow 2$
for $i \leftarrow 0$ **to** $i < numIterations$ **inc** 1 **do**
  Hash table $ants \leftarrow \{\}$
  for each *category* in *categories* **do**
    List $antList \leftarrow \{\}$
    Put entry $(category, antList)$ in hash table $ants$
  end for
  for $j \leftarrow 0$ **to** $j < numAnts$ **inc** 1 **do**
    $ant \leftarrow$ Create new ant (see Procedure 2)
    Store *ant*'s path states in the space of states
    $consequent \leftarrow$ Assign a consequent to the ant
    Evaluate *ant* by objective functions
    $antList \leftarrow$ Get value of entry with key $consequent$ from hash table $ants$
    Add *ant* to the list *antList*
    Put entry $(consequent, antList)$ in hash table $ants$
  end for
  for each *category* in *categories* **do**
    List *aux* $\leftarrow \{\}$
    Add to aux non-repeated individuals from lists retrieved from hash tables $ants$ and $paretoFronts$, given the key $category$
    List *newParetoFront* $\leftarrow$ Non-dominated solutions in aux(see Section 3.6)
    for each *ant* in *newParetoFront* **do**
      Update pheromone rate in the path followed by *ant* proportionally to Laplace accuracy and inversely proportional to the path's length
    end for
    Put entry $(category, newParetoFront)$ into hash table $paretoFronts$
  end for
  Evaporate pheromones along the whole space of states
  Normalize pheromone values
  maxDerivations $\leftarrow$ *maxDerivations* + *derivationStep*
end for
for each *category* in *categories* **do**
  List $paretoFront \leftarrow$ Get value of entry with key $category$ from hash table $paretoFronts$
  Add to the classifier the $winnerAnts$ ants returned after carrying out the niching approach over the $paretoFront$ ants (see Procedure 3)
end for
Establish the default rule in the classifier
$predictiveAccuracy \leftarrow$ Compute the predictive accuracy obtained when running the classifier built on the test set
return *predictiveAccuracy*

The next step performed by the algorithm is repeated for each available category in the data set: individuals created predicting the current category are merged with the non-dominated individuals of the previous generation that also predict this category. A new Pareto front is found among all these individuals. Individuals in this new Pareto front reinforce the pheromone concentration in the path they follow. Once the *k*-Pareto fronts have been found, evaporation and normalization processes take place, and the maximum number of derivations is incremented by the derivation step.

Procedure 3 Niching Approach

Require: $ants, cat, percentageOfCoverage, trainingSet$
List $winnerAnts \leftarrow \{\}$
$numInstances \leftarrow$ number of instances in $trainingSet$
$numInstancesClass \leftarrow$ number of instances of class cat in $trainingSet$
Sort $ants$ by $LaplaceAccuracy$
$flagsArray \leftarrow$ 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 $trainingSet$
    $catInstance \leftarrow$ Get category of $instance$
    if ant covers $instance$ **then**
      $idealTokens \leftarrow idealTokens + 1$
      if $flagsArray[j] \leftarrow$ **false and** $catInstance = cat$ **then**
        $flagsArray[j] \leftarrow$ **true**
        $capturedTokens \leftarrow capturedTokens + 1$
      end if
    end if
  end for
  if $capturedTokens >= (percentageOfCoverage \cdot numInstancesClass)$ **then**
    $LaplaceAccuracy_{adj} \leftarrow LaplaceAccuracy \cdot \frac{capturedTokens}{idealTokens}$
    Add ant to $winnerAnts$
  else
    $LaplaceAccuracy_{adj} \leftarrow 0$
  end if
end for
return $winnerAnts$

When the main loop of the algorithm ends, after performing numIterations iterations, a subset of ants predicting each class from the final Pareto fronts is selected by means of the niching approach described in Procedure 3. In this niching procedure, ants are sorted by their Laplace accuracy and then they compete to capture as many instances—here called tokens—as they can. Each ant can capture a token just in case it covers the token and it has not also been seized previously by another ant. Finally, only ants whose number of captured tokens exceeds the percentage of coverage established by the user are added to the *winnerAnts* list, having an adjusted Laplace accuracy computed as follows:

$$\text{Laplace accuracy}_{\text{adj}} = \text{Laplace accuracy} \cdot \frac{\text{capturedTokens}}{\text{idealTokens}} \tag{17}$$

where *idealTokens* is equal to the number of tokens covered by the ant.

Once the niching approach is carried out over the individuals of each Pareto front, the resulting ants are added to the classifier. The classifier's final decision list is made up of these ants, sorted by their adjusted Laplace accuracy in descending order. Finally, a default rule predicting the majority class in the training set is added at the bottom of the decision list and the classifier is run over the test set to compute its predictive accuracy.

4 Experimentation

4.1 Data sets and preprocessing

The simulation was performed using 15 public-domain UCI[1] (Frank et al. 2010) standard classification problems. This collection of data sets comprises problems with a broad range of dimensionality. The particular characteristics of these data sets can be seen in Table 1.

A tenfold cross-validation procedure is carried out, where each data set is randomly split into ten mutually exclusive folds, with each fold consisting of approximately the same proportion of classes as in the original data set. Each algorithm is executed ten times, with a different fold left out as the test set each time, the other nine being used for training. The predictive accuracy of the classifier is estimated by considering the average accuracy over the ten experiments:

$$\text{predAcc} = \sum_{i=1}^{10} \frac{\#\text{correctlyClassified}P_i}{\#\text{instances}P_i} \cdot 100 \tag{18}$$

where $\#\text{correctlyClassified}P_i$ is the number of test instances correctly classified when the partition P_i is left out as the test set, and #instances is the number of total instances in the original data set.

Notice that when evaluating the performance of non-deterministic algorithms, ten different seeds are used to carry out the stratified tenfold cross-validation ten times, in order to avoid any chance of obtaining biased results.

For each data set containing missing values, they were replaced with the mode (in the case of nominal attributes) or the arithmetic mean (in the case of numerical ones). In addition, since several algorithms are unable to process numerical attributes directly, by applying the Fayyad and Irani's discretization algorithm (Fayyad et al. 1993), each training set was discretized and then the discrete intervals found were used to discretize their corresponding test set. This separation is necessary to obtain reliable results, because when discretizing the entire data set before generating the cross-validation folds, the discretization algorithm would take into account the test data, thus invalidating the results obtained. Both preprocessing actions were perfomed by means of the tool Weka.[2]

[1] University of California at Irvine data set repository is available at http://www.ics.uci.edu/ml/datasets.html.

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

Table 1 Data sets used in computational experiments

Data set	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
Vote	No	435	0	0	17	2	267/168
Australian	Yes	690	6	0	9	2	307/383
Breast-w	Yes	699	9	0	0	2	458/241
Credit-g	No	1,000	6	3	11	2	700/300
Iris	No	150	4	0	0	3	50/50/50
Wine	No	178	13	0	0	3	59/71/48
Lymphography	No	148	3	9	6	4	2/ 81/61/4
Glass	No	214	9	0	0	6	70/76/17/13/9/29
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

Moreover, since the experimentation carried out includes five algorithms that are able to cope directly with numerical values (cAnt-Miner2-MDL, PSO/ACO2, Bojarczuk-GP, JRIP and PART) and, hence, do not require prior discretization of the cross-validation folds, we performed their respective experiments in duplicate, as if we were considering two different algorithms instead of just one: the first execution was performed over the original cross-validation folds containing continuous values, and the second over the discretized folds. To differentiate between them, the suffix 'D' has been appended to the name of each algorithm when run over the preprocessed data.

4.2 Comparison with other rule-based classification algorithms

This section presents the algorithms with which MOGBAP is compared. The comparison considers several representative rule-based classification algorithms belonging to different paradigms:

- Ant programming: GBAP, a grammar-based AP algorithm which was introduced in Sect. 2.2.
- Ant colony optimization: three ACO algorithms are considered, Ant-Miner, Ant-Miner+, and cAnt-Miner2-MDL. Another algorithm, PSO/ACO2, which follows an approach which is a mix of the ACO and PSO is used for the comparison. All the algorithms have been discussed in Sect. 2.1.
- Genetic programming: a constrained syntax algorithm, Bojarczuk-GP (Bojarczuk et al. 2004), is also included in the comparison. The set of functions that this algorithm considers consists of logical (AND, OR) and relational ($=$, $\neq$, $\leq$, $>$) operators. It follows a mixed individual = rule/rule set approach, where each individual encodes a set of rules in disjunctive form predicting the same class, and the classifier induced for a given problem has a number of individuals equal to the number of classes in the data set.
- Reduced error pruning: JRIP, the Weka implementation of the well-known sequential covering Repeated Incremental Pruning to Produce Error Reduction (Cohen 1995) algorithm.
- Decision trees: PART (Frank et al. 1998), which extracts rules from the decision tree generated by the J48 Weka's algorithm.

The algorithms were executed using the following implementations. For MOGBAP and GBAP, we used our own implementations. For Ant-Miner and cAnt-Miner2-MDL, the open source code provided in the framework Myra was employed.[3] In case of Ant-Miner+, the code given by the authors was used. To run PSO/ACO2, its open-source implementation was used.[4] The GP algorithm was executed using the implementation publicly available in the framework JCLEC[5] (Ventura et al. 2007). Finally,

[3] Myra framework is available at http://myra.sourceforge.net/.

[4] PSO/ACO2 is publicly available at http://sourceforge.net/projects/psoaco2.

[5] JCLEC framework is at http://jclec.sourceforge.net.

Table 2 User-defined parameter configuration

Algorithm	Name	Description	Value
MOGBAP	numAnts	Number of ants	20
	numIterations	Number of iterations	100
	maxDerivations	Maximum number of derivations for the grammar	15
	percentageOfCoverage	Minimum percentage of instances belonging to the class which the rule predicts and that it should cover (for MOGBAP)	5 %
	minCasesPerRule	Minimum number of instances covered per rule (for GBAP)	3
GBAP	[τ_0]	Initial pheromone amount for transitions	1.0
	[τ_{min}]	Minimum pheromone amount allowed for transitions	0.1
	[τ_{max}]	Maximum pheromone amount allowed for transitions	1.0
	[ρ]	Evaporation rate	0.05
	[α]	Heuristic exponent value	0.4
	[β]	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	3,000
cAnt-Miner2-MDL	Number of ants	Number of ants	60
	Min. cases per rule	Minimum number of instances covered per rule	5
	Max. uncovered cases	Maximum number of uncovered cases	10
	Rules for convergence	Convergence test size	10
	Number of iterations	Maximum number of iterations	1,500
Ant-Miner+	nAnts	Number of ants	1,000
	rho	Evaporation rate	0.85
PSO/ACO2	numParticles	Number of particles	10
	numIterations	Number of iterations	200
	maxUncovExampPerClass	Maximum number of uncovered cases 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 $\geq$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
	Pptimizations	The number of optimization runs	2
	Pruning	Whether pruning is performed	True
PARTS	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

PART and JRIP were run by using the implementations available in Weka.

It is also worth noting that, in order to make a fair comparison, the same training and test sets, which were obtained as explained in the previous section, were used to test these algorithms. Otherwise, it would be difficult to justify the equity of the experimental study.

Table 2 lists the parameter settings for each algorithm. MOGBAP and GBAP share the same parameters, with the exception of *percentageOfCoverage*, which replaces the *minCasesPerRule* parameter in the MOGBAP algorithm. Thus, for MOGBAP, the same parameter configuration used in GBAP was adopted. Notice that both AP algorithms have four mandatory parameters, and the other six parameters—enclosed in square brackets—are optional, having default values.

Notice that no additional parameter optimization was done for MOGBAP. Thus, the configuration used for this algorithm should not be taken as the optimal set of parameters for each data set. Considering a particular data set, by fine tuning even better results can be achieved. Actually, the values employed correspond to the configuration selected in (Olmo et al. 2010) for GBAP, where the values for parameters were 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 setup globally reported the best values.

5 Results and discussion

This section presents and interprets the experimental study results. This section is divided into three different parts. The first one compares two versions of MOGBAP, one considering two objectives and the other one considering three. The second part of the study presents a comparison between MOGBAP and 8 other classifiers—13 if we take into account the separate experiments performed in the case of those classifiers capable of handling numeric attributes directly—regarding predictive accuracy, whereas the third part performs a similar analysis in terms of comprehensibility.

5.1 Comparing two-objective versus three-objective approaches

To study the performance of MOGBAP, its effectiveness was analyzed when considering two and three objectives. The two-objective MOGBAP version looks for the maximization of both sensitivity and specificity, whereas the three-objective version attempts an optimization of comprehensibility as well.

Table 3 presents several results obtained by each version of our algorithm. In both cases, the predictive accuracy results obtained for training are better than those obtained for the test. The three-objective MOGBAP obtains better average values in predictive accuracy (for test), average number of rules in the classifier and average

Table 3 Results obtained by two-objective versus three-objective MOGBAP versions

Data set	MOGBAP two-objectives						MOGBAP three-objectives					
	Acc_{tra}	σ_{tra}	Acc_{tst}	σ_{tst}	#R	#C/R	Acc_{tra}	σ_{tra}	Acc_{tst}	σ_{tst}	#R	#C/R
Hepatitis	89.00	1.40	84.59	10.93	8.7	2.24	89.33	1.52	85.15	11.51	9.1	1.99
Sonar	86.45	1.63	78.20	9.63	11.2	2.52	86.72	1.52	79.49	9.26	11.1	2.04
Breast-c	76.05	1.58	72.45	8.72	12.0	1.88	76.04	1.73	72.02	9.62	11.8	1.69
Heart-c	85.15	1.14	81.11	4.42	10.3	2.22	85.31	1.09	83.13	4.24	9.9	2.25
Ionos.	91.23	0.77	90.46	5.52	6.6	1.61	91.25	0.71	90.55	5.67	6.8	1.49
Horse-c	85.81	1.02	84.66	4.62	10.5	2.14	85.56	1.12	83.78	4.67	9.6	2.03
Vote	96.10	0.59	95.05	2.85	6.8	2.35	96.10	0.59	94.89	2.92	6.6	2.12
Australian	87.67	0.84	88.10	4.10	8.9	2.17	87.71	0.81	87.38	4.27	9.1	2.00
Breast-w	94.81	0.72	94.58	2.72	6.0	1.95	95.10	0.58	95.41	2.31	6.1	1.77
Credit-g	73.59	0.93	71.68	3.13	11.4	2.03	74.02	0.99	70.82	3.33	11.6	1.82
Iris	96.00	0.59	95.33	6.00	5.8	1.15	96.00	0.59	95.33	6.00	5.8	1.15
Wine	98.31	0.88	97.11	4.07	6.1	1.65	98.86	0.71	98.24	2.75	6.1	1.47
Lymph.	87.44	1.98	79.97	10.53	12.4	1.86	87.03	1.83	80.55	9.74	11.9	1.55
Glass	74.55	2.28	68.29	9.40	18.4	2.36	76.86	2.20	71.03	8.45	17.5	2.22
Primary tumor	50.52	1.39	41.83	6.79	35.6	2.89	50.16	1.40	42.18	7.19	34.9	2.53
Avg. results	84.85	1.18	81.56	6.23	11.38	2.07	84.67	1.16	82.00	6.13	11.19	1.87

number of conditions per rule. However, this analysis is not very meaningful, since it is based on the average of the absolute values of several measures across data sets that have different characteristics. Therefore, we performed the Wilcoxon signed rank test with the continuity correction over the test accuracy results of both MOGBAP versions, although the results obtained indicated that at a significance level of $\alpha = 0.05$, there were no significant differences between them.

We also analyzed the statistical significance of the number of conditions per rule applying the Wilcoxon pair test. The p value obtained was equal to 0.001363, which is lower than 0.01. This proves that the three-objective version, which also considers the comprehensibility metric, is significantly more comprehensible than the two-objective one with a 99 % probability.

Thus, the adopted version of MOGBAP is the one which aims to simultaneously optimize the three objectives of sensitivity, specificity, and comprehensibility.

5.2 Accuracy analysis

In this section, the performance of MOGBAP is compared to several rule-based algorithms regarding the predictive accuracy criterion. Table 4 reports on the average predictive accuracy values, with their standard deviations, obtained for each algorithm over each data set. The bold results indicate the algorithm that yields the maximum average classification rate in each data set. Notice that because of space restrictions, the results are shown in two separate tables: Table 4a presents the results obtained over the discretized data sets; and Table 4b presents them for the original data sets without discretization.

To statistically evaluate the differences between the algorithms, the Friedman test is performed. This is a non-parametric test that compares the mean ranks of k algorithms over N data sets. These ranks indicate which algorithm obtains the best results considering all data sets. To compute them, a rank of 1 is assigned to the algorithm with the highest accuracy in one data set, the algorithm with the next highest accuracy is given a rank of 2, and so on. Finally, the average ranks of each algorithm for all data sets are calculated: they are shown in the last row of Table 4.

According to the Iman&Davenport test, we state that all the algorithms are equivalent if the null hypothesis is accepted. In contrast, if the null hypothesis is rejected, we will state that there are differences between the algorithms. With a significance level of $\alpha = 0.05$, the Iman&Davenport statistic of average rankings distributed according to the F-distribution with $k - 1$ and $(k - 1) \cdot (N - 1)$ degrees of freedom is 7.1368, which does not belong to the critical interval equal to $C_0 = [0, (F_F)_{0.05,13,182} = 1.7743]$. Thus, the Iman&Davenport test rejects the null hypothesis that all algorithms perform equally well when $\alpha = 0.05$.

To reveal the classifiers from which MOGBAP is statistically significantly different, it is necessary to perform a post hoc test. The Bonferroni–Dunn test (Demšar 2006) can be applied since all algorithms are compared with respect to a control algorithm, MOGBAP, focusing on all possible pairwise comparisons that involve the MOGBAP algorithm. The critical value revealed by this test at the same significance level of $\alpha = 0.05$ is 4.4161 and, therefore, the performance of MOGBAP is statistically better regarding accuracy than those of PSO/ACO2$_D$, PSO/ACO, Ant-Miner+, Ant-Miner, cAnt-Miner2$_D$, cAnt-Miner2, GP, GP$_D$, and PART. These results are presented in Fig. 6, where it is also possible to see that MOGBAP achieves competitive or even better accuracy results than GBAP, PART$_D$, JRIP$_D$, and JRIP. These results show that MOGBAP is statistically more accurate than the other ant-based algorithms considered, except for GBAP, although it obtains a higher rank and superior average accuracy.

5.3 Comprehensibility analysis

As mentioned previously, all algorithms included in the comparison are rule-based algorithms. Hence, the complexity of the rule set obtained by each classifier and the complexity of the rules should be compared appropriately. The average number of rules and the average number of conditions per rule of all algorithms over the 15 data sets are presented in Table 5. Notice that there are specific metrics related to the comprehensibility of the CFGs (Crepinšek et al. 2010). However, they are not included in the study since only three algorithms are based on the use of a CFG. On the other hand, the complexity of the rule set might also be computed by existing approaches used in the programming language community (Kosar et al. 2010; Mernik et al. 2005).

It should be pointed out that apart from the GP algorithm, which also makes use of the OR operator, all algorithms mine rules as a conjunction of conditions and, therefore, the number of rules in the classifier can be computed directly by counting the number of rules. However, in the case of the GP algorithm, to compute correctly the number of rules in the classifier, it is necessary to consider each disjunction as the connection of two different rules, without considering OR nodes as conditions.

Table 6 summarizes the mean results of all algorithms over the 15 data sets with respect to predictive accuracy, the number of rules in the classifier, and the number of conditions per rule. However, these results are not significant, and the average rankings of the algorithms across

Table 4 Predictive accuracy (%) comparative results

	MOGBAP		GBAP		ANT-MINER		ANT-MINER+		CANT-MINER2$_D$		PSO/ ACO2$_D$		GP$_D$		JRIP$_D$		PART$_D$	
Data set	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}
Hepatitis	**85.15**	1.52	82.17	12.04	83.27	10.32	81.79	10.30	84.89	1.27	84.59	9.33	71.05	14.45	81.54	12.05	84.64	7.66
Sonar	79.49	9.26	**81.98**	7.44	76.95	6.89	76.05	7.22	76.75	10.76	78.49	8.05	79.82	9.24	80.33	6.61	77.84	8.10
Breast-c	72.02	9.62	71.40	7.86	73.42	7.29	73.05	6.86	**74.15**	1.22	68.63	6.87	68.63	10.94	72.00	6.41	68.48	7.90
Heart-c	**83.13**	4.24	82.84	5.24	78.01	6.69	82.41	5.10	75.54	9.77	82.25	5.36	70.02	7.08	82.20	5.12	80.13	6.39
Ionos.	90.55	5.67	**93.02**	4.07	84.39	6.73	92.89	4.02	86.17	10.67	89.97	4.99	76.48	8.19	91.70	5.14	88.93	4.02
Horse-c	**83.78**	4.67	82.97	6.34	82.71	4.73	81.79	6.03	81.98	7.12	82.06	4.93	82.52	6.06	83.72	6.35	81.5	3.72
Vote	94.89	2.92	94.37	3.57	94.29	3.27	94.66	3.72	94.33	4.86	94.30	3.81	**95.67**	2.78	95.44	3.52	94.51	3.08
Australian	**87.38**	4.27	85.47	4.49	85.30	4.12	83.48	3.38	85.38	4.5	85.19	4.69	85.52	4.50	86.70	5.15	84.66	4.48
Breast-w	95.41	2.31	**96.50**	1.68	94.69	2.04	94.28	2.86	94.44	1.79	95.86	1.91	87.39	2.75	95.71	1.81	95.71	1.82
Credit-g	70.82	3.33	70.79	4.27	70.55	3.72	70.80	3.87	70.65	5.67	70.36	3.55	63.02	7.03	70.70	3.26	72.70	3.26
Iris	95.33	6.00	**96.00**	4.10	95.20	5.47	94.00	3.59	95.33	7.83	95.33	6.70	91.73	10.46	**96.00**	5.33	95.33	6.70
Wine	**98.24**	2.75	97.01	4.37	91.86	5.08	93.86	4.61	92.48	7.76	90.20	2.86	83.69	9.44	95.61	5.37	95.03	3.89
Lymph.	80.55	9.74	**81.00**	10.35	75.51	9.59	77.23	10.91	74.15	18.34	76.59	12.20	77.78	12.77	78.84	11.49	78.43	14.30
Glass	71.03	8.45	69.13	8.66	65.52	9.26	62.03	9.80	64.51	14.51	71.16	10.54	39.23	11.34	69.00	8.70	**73.91**	8.43
Primary	**42.18**	7.19	37.91	6.55	37.75	5.27	37.26	5.43	36.21	9.23	37.19	5.88	16.41	4.96	38.11	3.75	38.36	5.09
RANKING	**3.0667**		4.0333		8.8		8.1		8.9		7.7333		10.3		4.1667		6.5333	

(a) Predictive accuracy results over discretized data sets

	CANT-MINER2		PSO/ ACO2		GP		JRIP		PART	
Data set	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}
Hepatitis	79.93	10.61	85.03	13.81	73.72	11.40	80.95	9.96	78.08	13.99
Sonar	73.86	8.83	74.56	9.75	79.11	8.49	78.80	13.99	75.49	6.89
Breast-c	**74.15**	1.22	68.63	6.87	68.63	10.94	72.00	6.41	68.48	7.90
Heart-c	77.33	6.77	80.68	7.00	73.57	8.24	76.59	5.26	80.93	7.38
Ionos.	87.59	5.49	84.82	4.88	77.98	8.57	90.03	4.57	89.17	4.68
Horse-c	82.45	8.37	80.82	6.79	76.53	8.11	83.65	7.75	80.62	7.93
Vote	94.33	4.86	94.80	3.81	**95.67**	2.78	95.44	3.52	94.51	3.08
Australian	85.42	3.81	84.88	3.27	85.51	2.04	86.03	2.87	84.49	3.85
Breast-w	95.03	2.30	96.02	1.38	91.61	4.08	95.56	2.60	94.70	3.00
Credit-g	71.20	4.53	70.04	3.65	65.76	7.66	**73.50**	4.57	70.50	4.50
Iris	93.93	5.53	95.80	6.07	89.59	11.36	95.33	4.27	94.00	6.29
Wine	89.23	8.04	88.87	7.15	80.46	11.97	93.22	6.11	90.96	4.53
Lymph.	75.23	11.02	79.28	8.03	77.66	19.04	77.76	9.26	78.15	10.48
Glass	66.24	10.86	68.91	10.34	38.78	11.67	65.70	8.37	68.50	10.65
Primary	36.21	9.23	37.19	5.88	16.41	4.96	38.11	3.75	38.36	5.09
RANKING	9.1667		8.0		11.0333		5.9		9.2666	

(b) Predictive accuracy results over data sets without discretization

data sets was computed again. The last but one row of Table 5 represents the average ranking obtained by each algorithm with regard to the rule set length, and the last row specifies the average rankings regarding the number of conditions per rule. In both cases, the algorithm with the lowest ranking value (in bold) corresponds to GP-Bojarczuk$_D$.

A first statistical analysis is performed considering the average number of rules in the classifier (the lower the number of rules, the more comprehensible the classifier). Assuming a significance level of $\alpha = 0.05$, the value of the Iman&Davenport statistic according to the F-distribution is equal to 18.1522, and the critical interval is $C_0 = [0, (F_F)_{0.05,13,182} = 1.7743]$. Hence, the Iman&Davenport test rejects the null hypothesis, which means that there are significance differences among the algorithms considering this measure. Further, according to the Bonferroni–Dunn test at the 5 % level, a difference in mean ranks greater than 4.4161 implies the existence of significant differences between the pair of algorithms considered. At the same significance level, the difference between the ranks of GP$_D$, GP, JRIP$_D$, JRIP and Ant-Miner+, and the rank of MOGBAP is greater than the Bonferroni–Dunn critical value, 4.4161. Therefore, these algorithms are significantly better than MOGBAP regarding rule set complexity. Nevertheless, there are no differences between MOGBAP and GBAP, Ant-Miner, PSO/ACO2$_D$, and PART$_D$.

Ideally, the best result regarding the classifier's rule set complexity would be obtained by mining a single rule for

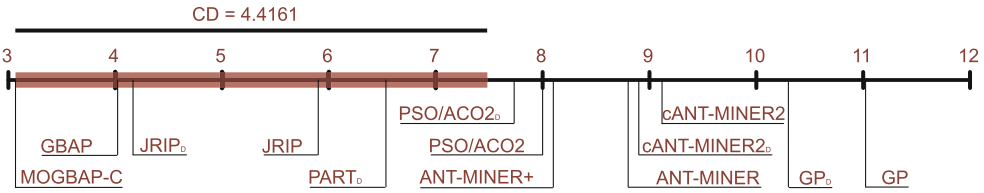


Fig. 6 Results of the Bonferroni–Dunn test. All classifiers with ranks outside the marked interval are significantly different from MOGBAP regarding predictive accuracy ($p < 0.05$)

Table 5 Rule set length and rule complexity comparative results

Data set	MOGBAP		GBAP		Ant-Miner		Ant-Miner+		cAnt-Miner2$_D$		PSO/ACO2$_D$		GP$_D$		JRIP$_D$		PART$_D$	
	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R
(a) Rule set length and rule complexity results over discretized data sets																		
Hepatitis	9.1	1.99	8.1	1.89	4.8	1.99	3.9	3.25	6.0	2.04	7.4	2.28	3.1	1.22	3.8	2.15	8.4	2.30
Sonar	11.1	2.04	12.3	1.81	5.2	2.07	4.0	3.48	6.6	1.99	6.1	2.92	3.0	1.00	4.6	2.21	13.9	2.98
Breast-c	11.8	1.69	13.2	1.91	6.0	1.28	5.4	2.82	6.9	1.25	11.8	1.75	3.5	1.01	3.3	1.70	17.1	2.12
Heart-c	9.9	2.25	14.5	1.67	5.9	1.20	4.4	2.82	7.2	1.41	11.9	3.81	3.0	3.02	5.3	2.32	17.3	2.35
Ionos.	6.8	1.49	11.1	1.18	5.7	1.61	8.8	1.41	7.7	1.59	4.5	4.03	3.1	1.14	7.7	1.48	8.2	1.83
Horse-c	9.6	2.03	9.0	1.46	6.3	1.49	4.7	3.41	7.5	1.57	20.1	3.39	3.0	1.00	3.5	1.74	13.2	2.38
Vote	6.6	2.12	17.2	2.19	5.6	1.36	5.2	2.34	6.8	1.59	6.1	1.33	3.0	1.00	3.1	1.38	7.7	1.84
Australian	9.1	2.00	10.1	1.08	6.5	1.53	3.3	2.08	7.6	1.52	25.8	6.96	3.0	1.00	5.2	1.80	19.4	2.01
Breast-w	6.1	1.77	6.6	1.65	7.2	1.04	6.4	1.92	8.1	1.12	10.5	1.10	3.0	1.00	6.5	1.74	10.9	1.63
Credit-g	11.6	1.82	22.9	1.82	9.1	1.51	3.3	3.31	9.7	1.37	52.8	4.20	3.3	1.17	7.1	2.54	57.8	2.70
Iris	5.8	1.15	3.7	1.06	4.3	1.03	3.9	1.80	5.2	1.02	3.0	1.20	4.3	1.29	3.0	1.00	4.6	1.00
Wine	6.1	1.47	7.2	1.50	5.1	1.33	2.5	2.19	6.5	1.41	4.0	1.73	4.1	1.27	4.2	1.56	6.3	1.77
Lymph.	11.9	1.55	10.2	1.60	4.7	1.69	4.6	2.83	6.9	1.74	15.6	2.11	5.1	1.02	6.9	1.53	10.2	2.30
Glass	17.5	2.22	21.6	1.79	8.4	1.76	12.4	4.10	10.4	2.25	24.5	3.13	8.2	1.48	8.0	2.03	13.7	2.32
Primary	34.9	2.53	45.9	2.60	12.1	3.35	9.3	8.50	21.8	4.43	86.5	6.01	23.7	1.37	8.3	3.13	48.7	3.23
#R ranking	9.9		10.9		6.5333		4.7667		8.6667		9.7667		**3.0333**		4.5667		12.1333	
#C/R ranking	7.2		5.5333		5.1		12.7333		5.8333		10.9333		**2.8**		6.7667		9.8333	

Data set	cAnt-Miner2		PSO/ACO2		GP		JRIP		PART	
	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R	#R	#C/R
(b) Rule set length and rule complexity results over data sets without discretization										
Hepatitis	6.5	1.71	3.6	1.79	3.0	1.02	4.3	2.23	10.7	2.18
Sonar	6.6	1.99	4.5	3.09	3.0	1.00	4.9	1.92	12.6	2.51
Breast-c	3.3	1.70	11.8	1.75	3.5	1.01	3.3	1.70	17.1	2.12
Heart-c	7.4	1.71	12.8	3.18	4.0	3.39	4.0	2.05	20.0	1.76
Ionos.	6.8	1.81	4.8	2.95	3.6	1.16	5.9	1.20	7.3	2.41
Horse-c	7.0	1.78	15.3	3.60	3.1	1.72	4.2	1.81	18.7	2.53
Vote	6.8	1.59	6.1	1.33	3.0	1.00	3.1	1.38	7.7	1.84
Australian	8.5	1.56	3.8	1.51	3.0	1.00	4.9	2.01	17.2	2.05
Breast-w	6.7	1.87	7.1	1.93	3.0	1.00	5.8	1.86	9.5	2.46
Credit-g	9.9	1.57	54.1	4.25	3.1	1.07	6.3	2.33	70.1	3.07
Iris	4.3	1.14	3.0	1.03	4.2	1.21	3.8	1.08	3.3	1.30
Wine	4.9	1.46	4.0	1.88	4.4	1.33	4.2	1.52	4.5	1.57
Lymph.	6.8	1.68	13.3	2.11	5.2	1.03	6.3	1.60	12.1	2.54
Glass	10.4	1.94	20.5	2.92	8.3	1.12	7.4	2.07	15.2	2.81
Primary	21.8	4.43	86.5	6.01	23.7	1.37	8.3	3.13	48.7	3.23
#R ranking	7.7		7.9667		3.3		4.0667		11.5	
#C/R ranking	6.9		10.1667		3.1667		7.2		10.8333	

Table 6 Average results of the algorithms

Algorithm	Accuracy	#R	#C/R
MOGBAP	**82.00**	11.19	1.87
GBAP	81.50	14.24	1.68
Ant-Miner	79.29	6.46	1.62
Ant-Miner+	79.71	5.47	3.08
cAnt-Miner2-MDL$_D$	79.13	8.32	1.75
PSO/ACO2$_D$	80.14	19.37	3.06
GP$_D$	72.60	**5.09**	**1.27**
JRIP$_D$	81.17	5.37	1.89
PART$_D$	80.68	17.16	2.18
cAnt-Miner2-MDL	78.81	7.85	1.86
PSO/ACO2	79.36	16.75	2.62
GP	72.73	5.21	1.30
JRIP	80.18	5.11	1.86
PART	79.13	18.31	2.29

Bold types indicate the algorithm that achieves the best result for a given metric

Table 7 Sample classifier on iris data set

```
IF      (= petalwidth (-inf,0.8]) THEN setosa
ELSE IF (AND (= petalwidth (1.75,inf))
             (= sepallength (6.15,inf)))
             THEN virginica
ELSE IF (= petallength (2.45,4.75])
             THEN versicolor
ELSE IF (= petalwidth (1.75,inf))
             THEN virginica
ELSE IF (= petalwidth (0.8,1.75])
             THEN versicolor
ELSE    virginica
```

predicting each class in the data set. However, this could be detrimental to the accuracy obtained by the algorithm. This statement is captured both in the results obtained by the GP-Bojarczuk algorithm over the data sets discretized and without discretization. This algorithm extracts nearly one rule per class in the classifier, but conversely has the worse predictive accuracy results. In contrast, MOGBAP has a good trade-off: obtaining the best accuracy results and at the same time behaving competitively when the rule set length is considered.

A second statistical analysis is performed applying the Iman&Davenport test to the results obtained for the average number of conditions per rule. The *F*-distribution's statistical value is 14.8741, which does not belong to the critical interval. Therefore the null hyphotesis that the algorithms perform equally well is again rejected. The post hoc Bonferroni–Dunn test leads to two conclusions. On the one hand, MOGBAP is significantly better than Ant-Miner+ regarding this metric. On the other hand, MOGBAP is not significantly better than the other algorithms, nor significantly worse which is more important.

It should be mentioned that the length of the rules mined by MOGBAP directly depends on the *maxDerivations* parameter. While more derivations are allowed, the algorithm is capable of finding more complex relationships, normally entailing the extraction of rules which have more conditions.

After performing both tests, it is possible to conclude that MOGBAP obtains competitive results in terms of comprehensibility, regarding both the rule set length and the number of conditions per rule. Actually, it presents a good trade-off between accuracy and comprehensibility, since it is the algorithm that presents the best ranking values in predictive accuracy, also obtaining competitive comprehensibility results. Complexity of rules mined can be seen in the sample classifier of Table 7, obtained running the algorithm over a training set of the horse-c data set.

6 Concluding remarks

This paper proposed a multi-objective AP-based algorithm for classification rule mining. In short, this algorithm, called MOGBAP, is guided by the use of a CFG that restricts the search space and also uses a two-sided heuristic function. Another contribution of this work to the classification field is the novel multi-objective evaluation approach that avoids the overlapping problem when ranking individuals from different classes according to Pareto dominance. To do this, the algorithm finds a set of non-dominated individuals for each possible class in the training set. These solutions are kept in separate Pareto fronts until the last generation of the algorithm, when they are combined in the final classifier using a niching approach.

The behavior of two versions of MOGBAP on 15 data sets was studied. The first algorithm focuses on optimizing

sensitivity and specificity measures, whereas the second one also aims to optimize a comprehensibility metric. The latter algorithm statistically outperforms the former in terms of comprehensibility, even without being superseded by the two-objective version regarding accuracy. Thus, the three-objective version is later used in another experimental study which considers seven other rule-based algorithms from different paradigms. Non-parametric statistical tests are used to analyze both the accuracy and comprehensibility of the algorithms. The results indicate that, with a likelihood of 95 %, MOGBAP is statistically more accurate than the other ant-based algorithms considered—Ant-Miner, Ant-Miner+, cAnt-Miner2-MDL, and PSO/ACO2—and also a GP algorithm and the PART algorithm run over the data sets without discretization. In addition, our algorithm obtains competitive results regarding the complexity of the classifier rule set and the rules mined, reaching a good trade-off between accuracy and comprehensibility. These results confirm that multi-objective grammar-guided AP is an adequate technique for tackling classification problems.

Acknowledgments This work was supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN-2011-22408, and FEDER funds.

References

Abbass HA, Hoai X, Mckay RI (2002) AntTAG: a new method to compose computer programs using colonies of ants. In: IEEE Congress on evolutionary computation (IEEE CEC), pp 1654–1659

Alatas B, Akin E (2009) Multi-objective rule mining using a chaotic particle swarm optimization algorithm. Knowledge-Based Syst 22(6):455–460

Angus D, Woodward C (2009) Multiple objective ant colony optimisation. Swarm Intell 3(1):69–85

Barron A, Rissanen J, Yu B (1998) The minimum description length principle in coding and modeling. IEEE Trans Inf Theory 44(6):2743–2760

Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 11(6):4135–4151

Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 35(3):268–308

Bojarczuk CC, Lopes HS, Freitas AA, Michalkiewicz EL (2004) A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. Artif Intell Med 30:27–48

Bonabeu E, Eric T, Dorigo M (1999) Swarm intelligence: from natural to artificial systems. Oxford University, Oxford

Boryczka M (2005) Eliminating introns in ant colony programming. Fundamenta Informaticae 68(1–2):1–19

Boryczka M (2008) Ant colony programming with the candidate list. In: KES International conference on agent and multi-agent systems: technologies and applications (KES-AMSTA). LNAI, vol 4953, pp 302–311

Boryczka M, Czech ZJ (2002) Solving approximation problems by ant colony programming. In: Genetic and evolutionary computing late breaking papers, pp 39–46

Boryczka M, Czech ZJ, Wieczorek W (2003) Ant colony programming for approximation problems. In: Genetic and evolutionary computation conference (GECCO), pp 142–143

Chen Y, Yang B, Dong J (2004) Evolving flexible neural networks using ant programming and PSO algorithms. In: Advances in neural networks ISSN. LNCS, vol 3173

Cios K, Pedrycz W, Swiniarski R, Kurgan L (2010) Data mining: a knowledge discovery approach. Springer, Berlin

Cohen W (1995) Fast effective rule induction. In: International conference on machine learning (ICML), pp 115–123

Crepinšek M, Kosar T, Mernik M, Cervelle J, Forax R, Roussel G (2010) On automata and language based grammar metrics. Comput Sci Inf Syst 7(2):309–329

Deb K, Agrawal S, Pratap A, Meyarivan T (2000) A fast elitist nondominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In: International conference on parallel problem solving from nature (PPSN). Springer, Berlin, pp 849–858

Dehuri S, Patnaik S, Ghosh A, Mall R (2008) Application of elitist multi-objective genetic algorithm for classification rule generation. Appl Soft Comput 8:477–487

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Dorigo M, Maniezzo V, Colorni A (1996) The ant system: Optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B 26:29–41

Dorigo M, Stützle T (2002) The ant colony optimization metaheuristic: algorithms, applications and advances. International series in operations research and management science. Kluwer Academic Publishers, Dordrecht

Espejo P, Ventura S, Herrera F (2010) A survey on the application of genetic programming to classification. IEEE Trans Syst Man Cybern Part C Appl Rev 40(2):121–144

Fayyad UM, Irani KB (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In: International joint conference on uncertainly in artificial intelligence (IJCAI), pp 1022–1029

Floreano D, Drr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intell 1:47–62

Frank A, Asuncion A UCI machine learning repository (2010). URL http://archive.ics.uci.edu/ml

Frank E, Witten IH (1998) Generating accurate rule sets without global optimization. In: International conference on machine learning (ICML), pp 144–151

García-Martínez C, Cordón O, Herrera F (2007) A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. Eur J Oper Res 180(1): 116–148

Geyer-Schulz A (1995) Fuzzy rule-based expert systems and genetic machine learning, studies in fuzziness, vol 3. Physica, Heidelberg

Green J, Whalley J, Johnson C (2004) Automatic programming with ant colony optimization. In: UK workshop on computational intelligence, pp 70–77

Holden N, Freitas AA (2008) A hybrid PSO/ACO algorithm for discovering classification rules in data mining. J Artif Evol Appl 2008:2:1–2:11

Huang TM, Kecman V, Kopriva I (2006) 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. Springer, New York

Ishibuchi H, Nojima Y (2007) Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning. Int J Approx Reason 44(1):4–31

Kapočite-Dzikiene J, Raškinis A (2008) Hierarchical classificator: a cognitive approach to decision tree building. Inf Technol Control 37:43–51

Keber C, Schuster MG (2002) Option valuation with generalized ant programming. In: Genetic and evolutionary computation conference (GECCO), pp 74–81

Kosar T, Oliveira N, Mernik M, Pereira MJV, Črepinšek M, da Cruz DC, Henriques PR (2010) Comparing general-purpose and domain-specific languages: An empirical study. Comput Sci Inf Syst 7(2):247–264

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. The MIT Press, Cambridge

Kumaresan N (2011) Optimal control for stochastic linear quadratic singular periodic neuro Takagi-Sugeno (T-S) fuzzy system with singular cost using ant colony programming. Appl Math Model 35(8):3797–3808

Lanzi P (2008) Learning classifier systems: then and now. Evol Intell 1:63–82

Martens D, Baesens B, Fawcett T (2011) Editorial survey: swarm intelligence for data mining. Mach Learn 82:1–42

Martens D, De Backer M, Vanthienen J, Snoeck M, Baesens B (2007) Classification with ant colony optimization. IEEE Trans Evol Comput 11:651–665

Mernik M, Heering J, Sloane AM (2005) When and how to develop domain-specific languages. ACM Comput Surv 37(4):316–344

Mernik M, Črepinšek M, Kosar T, Rebernak D, Žumer V (2004) Grammar-based systems: definition and examples. Informatica 28(3):245–255

Mullen RJ, Monekosso D, Barman S, Remagnino P (2009) A review of ant algorithms. Expert Syst Appl 36:9608–9617

Olmo JL, Romero JR, Ventura S (2010) A grammar based ant programming algorithm for mining classification rules. In: IEEE congress on evolutionary computation (IEEE CEC), pp 225–232

Olmo JL, Romero JR, Ventura S (2011) Using ant programming guided by grammar for building rule-based classifiers. IEEE Trans Syst Man Cybern Part B Cybern 41(6):1585–1599

Otero F, Freitas AA, Johnson C (2008) cAnt-Miner: an ant colony classification algorithm to cope with continuous attributes. In: International conference on Swarm Intelligence (ANTS). LNCS, vol 5217, pp 48–59

Otero FEB, Freitas AA, Johnson CG (2009) Handling continuous attributes in ant colony classification algorithms. In: IEEE symposium on computational intelligence and data mining (IEEE CIDM), pp 225–231

Parpinelli R, Freitas AA, Lopes HS (2002) Data mining with an ant colony optimization algorithm. IEEE Transactions Evol Comput 6:321–332

Roux O, Fonlupt C (2000) Ant programming: or how to use ants for automatic programming. In: International conference on Swarm Intelligence (ANTS), pp 121–129

Rudokaite-Margelevičiene D, Pranevičius H, Margelevičius M (2006) Data classification using Dirichlet mixtures. Inf Technol Control 35:157–166

Salama K, Abdelbar A, Freitas A (2011) Multiple pheromone types and other extensions to the ant-miner classification rule discovery algorithm. Swarm Intell 5:149–182

Salehi-Abari A, White T (2008) Enhanced generalized ant programming (EGAP). In: Genetic and evolutionary computation conference (GECCO), pp 111–118

Salehi-Abari A, White T (2009) The uphill battle of ant programming vs. genetic programming. In: International joint conference on computational intelligence (IJCCI), pp 171–176

Shirakawa S, Ogino S, Nagao T (2008) Dynamic ant programming for automatic construction of programs. IEEE Trans Electr Electron Eng 3(5):540–548

Stützle T, Hoos HH (2000) MAX–MIN ant system. Future Gener Comput Syst 16:889–914

Torácio A (2009) Multiobjective particle swarm optimization in classification-rule learning, chap. 3. Springer, Berlin, pp 37–64

Ventura S, Romero C, Zafra A, Delgado JA, Hervás C (2007) JCLEC: a java framework for evolutionary computation. Soft Comput 12(4):381–392