Swarm Intelligence manuscript No.
(will be inserted by the editor)

Learning Bayesian network classifiers using ant colony optimization

Khalid M. Salama · Alex A. Freitas

01-06-2013

Abstract Bayesian networks are knowledge representation tools that model the (in)dependency-relationships among variables for probabilistic reasoning. Classification with Bayesian networks aims to compute the class with the highest probability given a case. This special kind is referred to as Bayesian network classifiers. Since learning the Bayesian network structure from a dataset can be viewed as an optimization problem, heuristic search algorithms may be applied to build high-quality networks in medium- or large-scale problems, as exhaustive search is often feasible only for small problems. In this paper, we present our new algorithm, ABC-Miner, and propose several extensions to it. ABC-Miner uses ant colony optimization for learning the structure of Bayesian network classifiers. We report extended computational results comparing the performance of our algorithm with 8 other classification algorithms, namely 6 variations of well-known Bayesian network classifiers, *c*Ant-Miner for discovering classification rules, and a support vector machine algorithm.

Keywords: Ant Colony Optimization (ACO), Data Mining, Classification, Bayesian Network Classifiers.

1 Introduction

Data mining is an active research area involving the development and analysis of algorithms for extracting interesting knowledge (or patterns) from real-world datasets. Classification is a central problem in the data mining field, where the goal is to build, from labelled cases (also called instances), a model (classifier) used to predict the class of unlabelled cases [23]. While the literature includes several approaches for tackling this problem, such as decision tree

School of Computing, University of Kent,
Canterbury, CT2 7NF, UK
E-mail: kms39@kent.ac.uk, A.A.Freitas@kent.ac.uk

construction, artificial neural network training and classification rule induction [23,43], in this paper we focus on a type of probabilistic graphical modeling, namely Bayesian network classification.

Naïve-Bayes, as discussed in [28], is the simplest type of Bayesian classifier used in the literature. Despite making a strong simplifying assumption – namely, that the attributes are independent of each other given the class label – Naïve-Bayes classifiers showed effective predictive performance [28]. Nonetheless, since the attribute-independency assumption is not realistic in many datasets, there have been many attempts to improve the performance of Naïve-Bayes by extending it to more sophisticated types of probabilistic graphical models, such as Tree Augmented Naïve-Bayes (TANs), Bayesian Networks Augmented Naïve-Bayes (BANs) and General Bayesian Networks (GBNs) [5,18]. In general, the search methods used to build such types of models are greedy and deterministic, and therefore prone to get stuck in local optima. Since learning the optimal Bayesian network (BN) structure from a dataset is $\mathcal{NP}$-hard [6], stochastic heuristic search algorithms may be applied on medium and large size problems where exhaustive search is not feasible. Such stochastic algorithms perform a more global search that is less likely to get stuck into local maxima to build high-quality Bayesian network classifiers in an acceptable computational time.

Ant colony optimization (ACO) [15] is a meta-heuristic inspired by the way real ants find the shortest route between two locations. ACO has been successful in solving several types of optimization problems, in particular combinatorial optimization problems, including the induction of classification rules in data mining [29,31–33,39] and the construction of general-purpose Bayesian networks [3,11,36,44]. However, to the best of our knowledge, ACO has not been used for learning Bayesian network classifiers before the introduction of the ABC-Miner algorithm.

The present paper is an extended version of the ANTS 2012 conference paper [41], where ABC-Miner, an ant-based Bayesian classification algorithm, was introduced. We build on the work described in [41] in four ways. First, a new flexible way of selecting the number of dependencies per variable is introduced. Second, a different pheromone update strategy is applied to better balance exploration and exploitation. Third, the number of datasets used in the experimental evaluation is increased from 15 to 25. Fourth, several more comparisons are performed between ABC-Miner and other algorithms to enrich the analysis of our results.

The rest of the paper is organized as follows. ACO is described in Section 2. In Section 3, we give an overview on the basic concepts of Bayesian Networks, then we describe various BN classifiers. Section 4 discusses the related work on the use of ACO algorithms for building BNs in the literature. In Section 5, we describe our novel ABC-Miner algorithm and how to tackle our learning problem using the ACO meta-heuristic. Sections 6 and 7 explain our experimental methodology and results, respectively. We conclude with some general remarks and directions for future research in Section 8.

2 Ant colony optimization

Inspired by the behaviour of natural ant colonies, Dorigo et al. [12,13,15] have defined Ant Colony Optimization (ACO) as a meta-heuristic for combinatorial optimization problems. The basic principle of ACO is that a population of artificial ants cooperates with each other to find the best path in a graph, analogously to the way that natural ants cooperate to find the shortest path between two points like their nest and a food source [1,7,14].

In ACO, each artificial ant constructs a candidate solution to the target problem, represented by a combination of solution components in the search space. Ants cooperate via indirect communication, by depositing pheromone on the selected solution components for a candidate solution. The amount of pheromone deposited is proportional to the quality of that solution, which influences the probability with which other ants will use that solution's components when constructing their solution. This contributes to the global search aspect of ACO algorithms. In addition, the probability of an ant choosing a solution component also depends on the value of a heuristic function that measures the desirability of that component.

The global search aspect is also induced by the fact that a population of ants searches for the best solution in parallel, exploring, thus, possibly different regions of the search space at each iteration of the algorithm. As a result of this global search, ACO is less likely to get trapped into local optima than conventional greedy algorithms, which increases the chances of finding a near-optimal solution in the search space.

In order to design an ACO algorithm to solve a specific type of problem, the following components of the algorithm should be carefully specified:

- **Construction graph** - This graph defines the search space to be explored by the ACO algorithm. Each ant incrementally constructs a candidate solution by visiting different components in the graph.
- **Heuristic function** - This is a function that uses some problem-specific information to evaluate the quality of each solution component available in the construction graph. The value of this function influences an ant's choice of which components will be used for constructing its candidate solution.
- **State transition rule** - This is a probabilistic transition rule that determines how each ant decides which solution component will be visited next. This rule is based on both the heuristic function value η and the pheromone amount τ associated with the solution components.
- **Quality evaluation measure** - This is a problem-specific function used to evaluate the quality of a solution constructed by an ant. The higher the quality of a solution constructed by an ant, the more pheromone will be deposited on the construction graph components used in that solution.
- **Pheromone update strategy** - This involves formulas for pheromone reinforcement and evaporation. Pheromone reinforcement is applied on components used in the constructed solution, while pheromone evaporation is applied on the whole construction graph to avoid stagnation and premature convergence. The goal of the pheromone update strategy is to direct ants

in future iterations of the ACO algorithm to solution components found in high quality solutions.
- **Local search** - An optional procedure to improve the quality of a constructed solution. This can be performed on each constructed candidate solution, or less often (e.g. performed just on the best solution in the colony at each iteration) in order to reduce computational time.

Section 4 discusses the related applications of the ACO meta-heuristic in the field of learning Bayesian networks. But first, background on Bayesian networks is given in the next section.

3 Bayesian networks

3.1 Overview on Bayesian networks

Bayesian networks (BNs) are powerful tools for knowledge representation and inference that encode variables dependence and independence relationships. More precisely, BNs represent a model of the joint probability distribution of n random variables $\mathbf{X} = \{X_1, X_2, X_3, ..., X_n\}$. A BN is represented as a directed acyclic graph (DAG), where nodes represent variables and edges represent statistical dependencies between the variables. Moreover, a set of conditional probability tables (CPTs), one for each variable, is computed to represent the parameters Θ of the network. The graphical structure of the network along with its parameters specifies a joint probability distribution over the set of variables $\mathbf{X}$, as shown in the following formula:

$$p(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} p(X_i | \mathbf{Pa}(X_i), \Theta, G), \quad (1)$$

where G is the DAG that represents the structure of the BN, and $\mathbf{Pa}(X_i)$ are the parents of variable X_i in G.

The process of learning (or constructing) a Bayesian network from a dataset $\mathbf{D}$ can be divided into two phases, namely learning the network structure, and then learning the parameters of the network. The latter phase is, compared to the former, a relatively easy phase, where a conditional probability table (CPT) is computed for each variable with respect to its parent variables. The CPT of variable X_i encodes the likelihood of each value of this variable given each combination of values of $\mathbf{Pa}(X_i)$ in the graph G, and the marginal likelihood of the dataset $\mathbf{D}$ given a structure G is denoted by $P(\mathbf{D}|G)$. The purpose of the network structure-learning phase is to find G that maximizes $P(\mathbf{D}|G)$ for a given $\mathbf{D}$. A popular approach to that phase consists of using a scoring function, f, that evaluates each candidate G with respect to $\mathbf{D}$, searching for the best network structure according to that score. Various scoring metrics for learning a BN structure have been proposed in the literature [8, 21, 45].

A well-known greedy and deterministic algorithm for learning a BN structure is Algorithm B [2]. This algorithm is initialized with an empty DAG

(i.e., an edge-less graph structure) and iteratively adds, to the current network structure, the edge that leads to the maximum increase in the scoring function f, subject to the constraint that no directed cycles are included in the graph. The algorithm stops when there are no more valid edges to be added, or when adding any valid edge does not increase the value of the scoring function.

K2, a function based on uniform prior scoring, is one of the most used scoring functions for learning and evaluating Bayesian networks [8], and has been the basis of much subsequent work [45]. K2 measures the joint probability of a Bayesian network G given a database $\mathbf{D}$, using the following formula:

$$P(G, \mathbf{D}) = P(G) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!, \tag{2}$$

where r_i is the number of possible values of the variable x_i, q_i is the number of possible configurations (instantiations) for the variables in $\mathbf{Pa}(x_i)$, N_{ijk} is the number of cases in $\mathbf{D}$ in which variable x_i has its kth value and $Pa(x_i)$ is instantiated to its jth value, and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

For further information about Bayesian networks, the reader is referred to [8,19,21], which provide detailed discussions of the subject.

3.2 Bayesian networks for classification

BN classifiers are a special kind of BNs where the class attribute is treated as a special variable in the network. The purpose is to compute the posterior probability of each value c in the class variable C given a case x (an instance of the input attributes $\mathbf{X}$) using network G, then label this case with the class value having the highest posterior probability. This goal can be stated by the following formulas:

$$C(x) = \underset{\forall\ c \in C}{\arg\max}\, P(C = c | \mathbf{X} = x, G) \tag{3}$$

$$\overbrace{P(C = c | \mathbf{X} = x, G)}^{\text{posterior probability}} \propto \overbrace{P(\mathbf{X} = x | C = c, G)}^{\text{likelihood}} \times \overbrace{P(C = c)}^{\text{prior probability}} \tag{4}$$

where $\propto$ is the mathematical symbol for the proportionality relationship. The following are various types of BN classifiers studied in the literature, as illustrated in Figure 1.

- **Naïve-Bayes** - As mentioned earlier, this classifier has the simplest type of BN structure where the class node is the only parent node of all other nodes. Despite its simplicity, Naïve-Bayes has surprisingly outperformed several other more elaborated classifiers in many experiments, especially where the attributes are not strongly correlated – since in such cases the attribute-independency assumption is not so problematic [18].

- **Tree Augmented Naïve-Bayes (TAN)** - This classifier is a simple and natural extension of Naïve-Bayes. A TAN classifier allows a node in a BN to have one parent, in addition the class variable [18]. This produces a BN with a tree-like structure. One of the most used methods for learning a TAN is a variation of the well-known Chow-Liu algorithm [4], which works as follows. First, it computes the conditional mutual information $I(X, Y|\mathbf{C})$ between each pair of variables X, Y given class variable $\mathbf{C}$. Then it builds a complete undirected graph (i.e., a graph where every pair of variables is connected by an undirected edge) and finds the maximum weighted spanning tree for that complete graph, where the weight of edge $X \rightarrow Y$ is annotated with $I(X, Y|\mathbf{C})$. Next, it chooses a root variable and sets the direction of all edges to be outwards of it. Finally, it adds one edge from the class node to each of the other variables.
- **BN Augmented Naïve-Bayes (BAN)** - This type of classifier is more elaborated (and more computationally expensive to learn) than both Naïve-Bayes and TAN. In a BAN, either there are no restrictions on the number of parents of a node or, more commonly, there is a maximum number of k parents (k-dependencies) that a node can have. In other words, while each node in TAN can have only one parent besides the class node, each node in BAN can have (up to) k parents besides the class node. Another variation of the Chow-Liu algorithm is utilized to build BANs as well [5]. Note that if $k = 1$ a BAN becomes a TAN.
- **General Bayesian Network (GBN)** - This type of BN classifier is quite different from the others described earlier, since a GBN treats the class variable node as an ordinary node (which can have both parent and child nodes) during the process of network-structure construction [4]. In essence, the system builds a general-purpose Bayesian network, finds the *Markov blanket* of the class node, deletes all the other nodes outside that blanket and uses the resulting network structure as a Bayesian classifier. A *Markov blanket* of a node n is the union of n's parents, n's children, and the parents of n's children.

Friedman et al. [18] provided an excellent study of these algorithms. A comprehensive investigation and comparisons of various Bayesian classifiers was done by Cheng and Greiner [4, 5]. A relatively recent survey on improving Naïve-Bayes for classification is found in [24].

4 Related work on ant colony optimization

The use of the ACO meta-heuristic in the classification task in data mining is not new. Since the Ant-Miner algorithm was proposed by Parpinelli et al. in [33], many variations of that algorithm have been proposed, extending or improving the original algorithm in several ways. Some examples are AntMiner+ [29], cAnt-Miner [32], and multi-pheromone Ant-Miner [37–40]. See also [31] for a recent survey of Ant-Miner and its variations. The original Ant-Miner and all of its extensions are based on the paradigm of rule induction, i.e., they

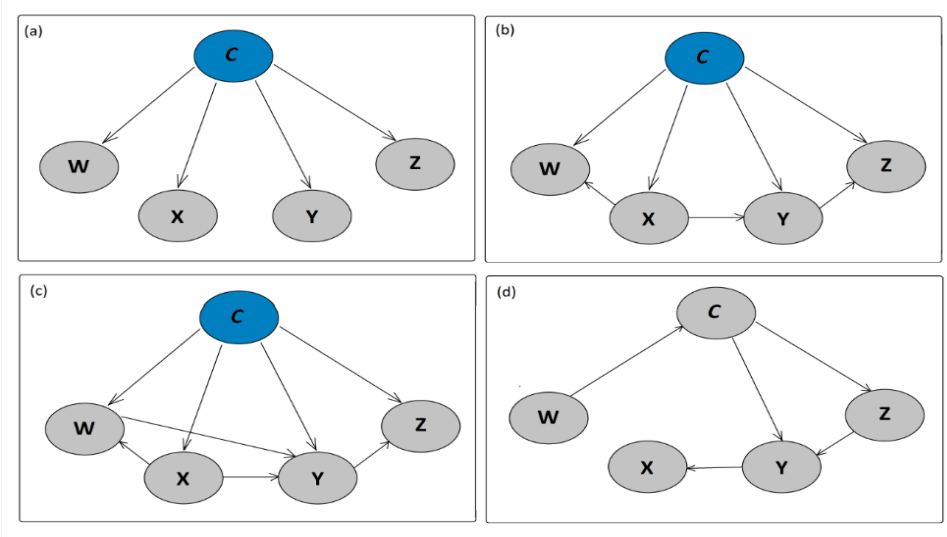

Fig. 1 Different types of Bayesian classifiers are presented: (a) Naïve-Bayes, where all the input variables have only the class variable as a parent. (b) TAN, where a variable can have one parent beside the class variable. (c) BAN, where a variable can have multiple parent beside the class variable. (d) GBN, where the class variable is treated the same as any other variable, and any dependencies between variables are possible.

learn a classification model that consists of a list or a set of classification rules, where each rule has the form: **IF** $<$ *Antecedent* $>$ **THEN** $<$ *Class* $>$. By contrast, this paper introduces an ACO-based classifier based on the paradigm of Bayesian network classifiers.

Several ACO algorithms have been proposed for learning the structure of Bayesian networks, including ACO-B [3], MMACO [35,36], ACO-E [10,11] and CHAINACO - K2ACO [44]. However, none of the above algorithms tackled the problem of building BN classifiers. To the best of our knowledge, ABC-Miner is the first ACO algorithm that was designed specifically for learning Bayesian network classifiers, as opposed to standard Bayesian networks (BNs). Next, we review the main characteristics of ACO algorithms for learning standard BNs.

The first ACO for learning BNs, called ACO-B, was proposed by Campos et al. [3]. In ACO-B, each ant adds one edge at a time to a candidate BN structure. After the BN construction, it deposits pheromone on the corresponding part of the construction graph in proportion to the quality of the constructed BN structure. The function used for measuring the quality of a candidate BN structure is the K2 scoring function [8], which is also the scoring function used as the heuristic function during the process of BN structure construction.

Pinto et al. proposed a hybrid method, called MMACO, which combines ideas from the Max-Min Parents and Children (MMPC) algorithm and ACO [35,36]. MMACO works in two phases. First, the MMPC algorithm is employed to construct the skeleton of the network (which is a graph with undirected

edges). Second, ACO is used to orientate the edges in that skeleton. MMACO uses the BDEu [21] scoring function both as the local heuristic function and as the BN-quality measure used to deposit pheromone.

Daly et al. proposed an ACO-based algorithm for learning a BN by carrying out a search in the space of BN equivalence classes. Their algorithm is called ACO-E [10,11]. An equivalence class consists of all network structures where changing the orientation (the direction of a dependency relationship) of one or more edges in a BN leads to a network with the same value of the scoring metric. Focusing on the equivalence classes of BNs has some advantages, as described by the authors in [10,11], but the focus on equivalence classes also makes the ACO algorithm more complex than a BN-learning ACO algorithm that does not need to compute equivalence classes.

Yanghui et al. proposed two new ACO algorithms for learning BN structures, named CHAINACO and K2ACO [44]. In essence, these algorithms use ACO to find the best node (variable) ordering, taking into account the dependencies between the variables. Finding the best node ordering is important because a variable can have, as parents, only variables that occur earlier in the ordered variable list, so that different orderings produce different models, with differerent degrees of predictive accuracy. The quality of node orderings is also based on the aforementioned K2 scoring function.

Junzhong et al. recently proposed HACO-B; a hybrid method that combines dependency analysis, ACO, and simulated annealing [25]. First, this method uses order-zero independence tests with a self-adjusting threshold value to reduce the size of the search space. Second, Bayesian network models are generated by using an ACO algorithm, where a new heuristic function is proposed to improve the search efficiency. Moreover, simulated annealing is used with HACO-B as a local search to optimize the BN structures constructed by the ants.

It should be emphasized that the previously discussed ACO algorithms were designed to build conventional, *general-purpose* BNs. In other words, the major algorithmic components of such ACO algorithms, such as the heuristic function and the network-quality scoring function, were designed to build general-purpose BNs, rather than for building BN *classifiers* as addressed in this paper.

5 A novel ACO algorithm for learning BN classifiers

5.1 The ABC-Miner algorithm

A major step in the design of an ACO algorithm is to specify the construction graph, consisting of candidate solution components. In the case of our target problem, the nodes in the construction graph represent the variables (attributes) of the underlying dataset, and an edge of the form $X \rightarrow Y$ indicates that the value of variable Y depends (probabilistically) on the value of variable X. Hence, the solution components are all the edges of the form

$X \rightarrow Y$ where $X \neq Y$ and X, Y belong to the set of attributes of the underlying dataset. A constructed candidate solution, consisting of a given set of selected edges (including edges from the class variable to every other variable), represents a BN classifier.

In the initialization step of ABC-Miner, the amount of pheromone assigned to each solution component in the construction graph is initialized with the value $1/|All\_Edges|$, where All_Edges is the set of all edges that are available to be selected by the ants at the start of the search. More precisely All_Edges is computed by assuming a fully connected graph for all the predictor variables (i.e., a graph with edges between every pair of predictor variables). Note that edges connecting the class variable to other variables are not included in All_Edges because these edges are not available for selection by the ants – rather, edges pointing from the class variable to each other variable are always already included in a candidate solution. Furthermore, the heuristic value assigned to each edge $X \rightarrow Y$ is given by the following conditional mutual information formula, which is also used as the heuristic function of ABC-Miner:

$$I(X, Y|C) = \sum_{c \in C} p(c) \sum_{x \in X} \sum_{y \in Y} p(x, y|c) \log_2 \frac{p(x, y|c)}{p(x|c)p(y|c)}, \tag{5}$$

where C is the class variable. $p(x, y|c)$ is the conditional probability of value $x \in X$ and $y \in Y$ given class value c, $p(x|c)$ is the conditional probability of x given c, $p(y|c)$ is the conditional probability of y given c and $p(c)$ is the prior probability of the value c of the class variable. Conditional mutual information is essentially a non-linear measure of the correlation between two variables given a third one. The value of this heuristic function is computed for each edge only once in the initialization step of the algorithm, and remains fixed throughout the execution of the algorithm.

The overall process of ABC-Miner is illustrated in Algorithm 1. At each iteration of the repeat-until loop (lines 5-22), first each ant_i creates a candidate solution, i.e., a Bayesian network classifier (BNC_i). Once all ants have constructed their candidate solutions (line 8), local search is applied to the best solution BNC_{tbest} produced by the ant colony at the current iteration t (line 15). The candidate solution produced by that local search operation is then used to deposit pheromones in the construction graph (line 16). Next, the algorithm compares the quality of the current iteration's best solution BNC_{tbest} with the best-so-far solution BNC_{bsf} and updates BNC_{bsf} if BNC_{tbest} is better than it (the if statement in lines 17-20). This set of steps is repeated until the same solution is generated for a number of consecutive trials specified by the `conv_iterations` parameter (indicating convergence) or until `max_iterations` is reached. The values of `conv_iterations`, `max_iterations` and `colony_size` are user-specified parameters (see Section 5 for parameter settings). Solution construction, solution quality evaluation and pheromone updating are discussed in detail in the following subsections.

Algorithm 1 Pseudo-code of ABC-Miner.

```
1: Begin
2: BNC_bsf = φ; t = 1;
3: InitializePheromones();
4: InitializeHeuristicValues();
5: repeat
6:       BNC_tbest = φ; Q_tbest = 0;
7:       for i = 1 → colony_size do
8:             BNC_i = CreateSolution(ant_i);
9:             Q_i = ComputeQuality(BNC_i);
10:            if Q_i > Q_tbest then
11:                  BNC_tbest = BNC_i;
12:                  Q_tbest = Q_i;
13:            end if
14:      end for
15:      PerformLocalSearch(BNC_tbest);
16:      UpdatePheromone();
17:      if Q_tbest > Q_bsf then
18:            BNC_bsf = BNC_tbest;
19:            Q_bsf = Q_tbest;
20:      end if
21:      t = t + 1;
22: until t = max_iterations or Convergence(conv_iterations);
23: return BNC_bsf;
24: End
```

5.2 Variable k-dependencies value selection

In order to build the structure of a Bayesian network (classifier), the maximum number of parents (dependencies) for the node (variable) should be specified. Instead of asking the user to guess the optimal number of dependencies (at most k parents for each node), the number of dependencies is determined by the ants in ABC-Miner in a self-adaptive manner. We allow each variable to have its own number of parents during the network construction.

Before creating each candidate solution, the ant selects the maximum number of dependencies k_i for each variable i independently, for the candidate network structure being constructed. That is, for n variables, the ant has n values for the k parents limit.

The possible values for the number of dependencies for each node are included as solution components in the construction graph. More precisely, the user has to specify the value of the `max_parents` parameter, and all integer values from 1 to that number are included as solution components in the construction graph. At each iteration, the iteration's best ant_i updates the pheromone on the value k_i based on the quality of BNC_{tbest}, which was constructed with the value k_i for variable i. This kind of pheromone updating tends to lead to the accumulation of pheromone in the best k_i value for each variable i.

5.3 Solution creation

The process of creating a new candidate solution is described in Algorithm 2. Each ant begins with a very simple network structure (line 2), where the only existing edges are edges connecting the class node (parent node) to each of the variables (child nodes). This is the network structure of a Naïve-Bayes classifier. Next, each ant incrementally expands that structure into a Bayesian Augmented Naïve-Bayes (BAN) structure, by adding one edge at a time to the current network (lines 5-6). The selection of the edge to be added at each step is based on the usual probabilistic state transition formula

$$p_{ij} = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{a}^{I} \sum_{b}^{J} [\tau_{ab}(t)]^{\alpha} \cdot [\eta_{ab}]^{\beta}}, \tag{6}$$

where p_{ij} is the probability of selecting the edge $i \rightarrow j$, $\tau_{ij}(t)$ is the amount of pheromone associated with edge $i \rightarrow j$ at iteration t and η_{ij} is the heuristic information for edge $i \rightarrow j$ computed using conditional mutual information (equation 5). The edge $a \rightarrow b$ represents a valid edge (see below) among the set of available edges. The exponents α and β represent the relative importance of the pheromone (τ) and heuristic information (η), respectively. Note that edges available for selection are directed, i.e., $i \rightarrow j \neq j \rightarrow i$.

In addition, ABC-Miner uses the "ants with personality" approach, where each ant_i is allowed to have its own *personality* by allowing it to have its own values of the parameters α_i and β_i, increasing the diversity in the ants' search strategies – see [38] for details.

Algorithm 2 Pseudo-code of Solution Creation Procedure by an Ant.

```
1: Begin CreateSolution(ant)
2: BNC ← Naïve-Bayes structure;
3: k_list = ant.SelectMaxParentsForEachVariable();
4: while GetValidEdges() ≠ φ do
5:       {i → j} = ant.SelectEdgeProbablistically();
6:       BNC = BNC ∪ {i → j};
7:       RemoveInvalidEdges(BNC, k_j);
8: end while
9: BNC.LearnParameters();
10: return BNC;
11: End
```

An edge $i \rightarrow j$ is deemed valid to be added to the network structure being constructed if the inclusion of that edge satisfies two criteria, namely: it does not create a directed cycle, and it does not violate the upper limit of k_j parents (chosen by the current ant) for the child variable j. Once a valid edge has been added to the BN classifier, all invalid edges are eliminated from the construction graph (line 7). Invalid edges are the ones whose inclusion would cause a direct cycle in the constructed BN classifier or would exceed the pre-specified maximum number of k_i parents for node i in the network.

The ant keeps adding edges to the current BN classifier until no valid edges are available. After constructing the network structure associated with BNC_i, the parameters Θ of that classifier are learnt by calculating the CPT for each variable (line 9), based on the network structure. This completes the construction of a candidate solution. Next, the quality of the solution is evaluated, and all the edges become available again for the next ant to construct another candidate solution.

5.4 Quality evaluation

Recall that conventional, general-purpose BN algorithms normally are designed to maximize the quality of a network structure in terms of representing the probabilistic dependencies among all attributes of the underlying dataset. That is, general-purpose BN algorithms do not recognize any special class attribute. They treat attributes (both class and predictor attributes) in the same way, and they are used to answer all types of inference queries about any set of variables, which may or may not include the class variable.

By contrast, a BN classifier (BNC) is only concerned about the queries regarding the class variable, and so our algorithm was designed to build a BNC that maximizes the predictive accuracy when predicting the target class variable. In other words, BNC learning algorithms try to find a network structure that can be used to calculate the probability of a class value given a case with predictor variables, and then the system predicts, to the current case, the class value with the highest calculated probability. Hence, the scoring functions of conventional, general-purpose BNs are not consistent with the objective of building a classifier [18].

Taking into account the aforementioned argument, ABC-Miner evaluates the quality of a constructed network directly as a classifier, measuring the predictive performance of that classifier. More precisely, we use the *accuracy* measure, a simple and yet popular measure of predictive performance, to evaluate the constructed model. The *accuracy* measure is computed as:

$$Accuracy = \frac{|Correctly\_Classified\_Cases|}{|Validation\_Set|} \tag{7}$$

5.5 Local search

At each iteration, local search is applied to the best BN classifier (denoted BNC_{tbest}) constructed in that iteration to improve the predictive accuracy of the classifier. In essence, the local search procedure works as follows. It tentatively removes one edge at a time from the constructed BNC in a reverse order (removing last the edge that was added to the network first). If that removal improves the quality of the BN classifier, this edge is removed permanently from the network, otherwise it is added once again. This process continues

until all the edges are tested to be removed from the BN classifier, and then the BN classifier with the highest quality is returned.

5.6 Pheromone update

Once the current iteration's best solution BNC_{tbest} has been improved by the local search procedure, pheromone is deposited on the solution components (network edges) used by that ant. The solution components that have their amount of pheromone increased include both the edges in the constructed BNC structure and the values of the number of parents selected by that ant during BNC construction.

Pheromone update is carried out to affect the probability of selecting the solution components for building further candidate solutions according to the quality of the previously constructed ones, and consists of pheromone deposit and evaporation. In the pheromone deposit phase, an amount of pheromone is deposited depending on the quality of two solutions: the iteration-best BNC_{tbest} and the best-so-far BNC_{bsf}. The contribution of the two solutions is weighted as follows:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \phi_1.Q_{tbest}(t) + \phi_2.Q_{bsf}(t), \tag{8}$$

where ϕ_1 and ϕ_2 represent the *intensity* of the pheromone to be deposited in iteration t according to the quality of the iteration-best and best-so-far solutions respectively. They are calculated as

$$\phi_1 = \frac{max\_iterations - t}{max\_iterations}, \quad \phi_2 = \frac{t}{max\_iterations} \tag{9}$$

Hence, in the early iterations, more weight is given to the iteration-best rather than to the best-so-far solutions. This is applied in order to introduce search diversion. However, as the number of elapsed iterations increases, the quality of the best-so-far increases, which gains more weight in directing the search, leading to convergence. Note that $\phi_1 + \phi_2$ equals 1 at any iteration t.

Pheromone normalization (to simulate evaporation as in [33]) is then applied to all pheromone values τ_{ij} in the construction graph by dividing each τ_{ij} by the total amount of pheromone on all the edges in the construction graph. This normalization is also carried out for the solution components representing the maximum number of dependencies.

5.7 ABC-Miner vs. ACO-B

As mentioned in Section 4, ACO-B is the first ant-based algorithm for learning the structure of *general-purpose* BNs. On the other hand, our proposed ABC-Miner algorithm learns the structure of BN *classifiers*. Although both use the ACO meta-heuristic, essential aspects of the algorithms are different due to the different targets. First, ACO-B does not distinguish between the

predictor variables and the class variable in the construction graph. So, all the edges (representing variable dependencies) between the variables including the class variable are available for selection in constructing the BN structure. On the other hand, ABC-Miner treats the class variable as a special class, that is a *cause* variable to all other input variables, as the goal of the inference mechanism for classification is to compute $P(C|\mathbf{X})$. This restricts each variable in the constructed BN classifier to have the class variable as its parent, and makes the construction graph include the edges only between the input variables (Section 4.1). Second, the initial candidate solution in ACO-B is an edge-less network structure, while an ant starts with a Naïve-Bayes network structure in ABC-Miner (Section 5.3). Third, the heuristic information function used in the ACO-B algorithm is K2, which does not consider the class variable as a special node in the network. On the other hand, ABC-Miner uses Conditional Mutual Information, which measures the correlation between two variables given the class variable. Fourth, the use of the quality evaluation measure differs (Section 5.3). ACO-B learns a general-purpose BN to answer any probabilistic query about any single or group of variables, via maximizing K2. On the other hand, ABC-Miner learns a BN classifier that is only concerned about the queries regarding the class variable; the probability of a class value given a case with predictor variables. Thus, ABC-Miner evaluates the quality of constructed candidate solutions directly as classifiers, using the accuracy measure (Section 5.4). Moreover, ABC-Miner introduces a dynamic and flexible way for selecting the k-dependencies in building the network structure, whereas in ACO-B the value of the parameter k-dependencies should be specified by the user prior the algorithm's execution (Section 5.2). The ABC-Miner algorithm uses a state transition formula that employs the "ants with personality" approach for increasing search diversity (Section 5.3). In addition, a new pheromone update strategy is introduced to achieve a good trade-off between exploration and exploitation with the flow of the algorithm's iterations (Section 5.6).

6 Experimental methodology

6.1 Comparative evaluations

We compare the predictive accuracy of two variations of ABC-Miner (with and without local search) with the predictive accuracies of eight other algorithms, namely Naïve-Bayes, TAN, GBN, SP-TAN, GHC-K2, GHC-A, cAnt-Miner and SVM. Naïve-Bayes, TAN and GBN are well-known and widely used for learning BN classifiers. In our experiments, we used WEKA [43] implementations for these algorithms. We also included in our experiments another well-known, more sophisticated variation of a TAN leaning algorithm: The SuperParent (SP-TAN) algorithm [27]. SP-TAN is a greedy search algorithm in which the edge associated with the highest accuracy improvement is selected at each step. A recent empirical comparison showed that the SP-TAN

algorithm outperformed many other BN classification algorithms in the literature [24]. Moreover, we implemented a variation of Algorithm B that uses a greedy hill-climbing (GHC) approach to learn BN classifiers. The algorithm starts with a Naïve-Bayes BN, a network where all the input nodes have only the class node as a parent. Then the algorithm iteratively adds the edge that leads to the maximum increase of the quality of the BN classifier being constructed. Our proposed ABC-Miner is compared with two variations of this implemented greedy hill-climbing algorithm. The first is GHC-K2, which employs the GHC approach to maximize the K2 scoring function (see equation 2) of the constructed network. The second is GHC-Acc, which employs the hill-climbing approach to maximize the predictive accuracy (see equation 7). Table 1 presents the main properties of the used algorithms.

Table 1 Summary of the BN classifier learning algorithms used in the experiments

Algorithm	Type	Search Strategy	Optimization
Naïve-Bayes	Deterministic	-	-
TAN	Deterministic	Finding Max. Spanning Tree	Cond. Mutual Info.
GBN	Deterministic	Greedy Hill Climbing	K2 Function
SP-TAN	Deterministic	Greedy Hill Climbing	Predictive Accuracy
GHC-K2	Deterministic	Greedy Hill Climbing	K2 Function
GHC-Acc	Deterministic	Greedy Hill Climbing	Predictive Accuracy
ABC-Miner	Stochastic	Ant Colony Optimization	Predictive Accuracy
ABC-Miner$^-$	Stochastic	ACO without local Search	Predictive Accuracy

The reason behind the two different implementations of GHC is to examine the effect of the choice of predictive accuracy or the K2 scoring function (as an optimization objective) on the quality of the produced BN classifiers. Note that both GBN and our implemented GHC-K2 are greedy algorithms that optimize the K2 scoring function, yet GBN builds a *general* BN, and GHC-K2 builds a BN where all the nodes have the class node as a parent. The pseudocode of the GHC method is presented in Algorithm 3.

Besides, we run experiments using the ABC-Miner algorithm without the local search procedure, in order to isolate its effect from the performance of the algorithm. This algorithm is denoted in Table 1 by ABC-Miner$^-$; it does not apply the local search operation in any iteration of the algorithm execution.

In addition, we compare our proposed ABC-Miner with another ant-based algorithm for classification, *c*Ant-Miner [32], which is a recent extension of the original Ant-Miner algorithm that handles continuous attributes dynamically (during the algorithm's run), without discretizing attributes in a data preprocessing step. Note that, as mentioned in Section 4, Ant-Miner (and also *c*Ant-Miner) tackle the classification problem by constructing a list of "IF-THEN" classification rules, while ABC-Miner builds Bayesian networks.

Algorithm 3 Pseudo-code of GHC.

```
1: Begin GHC
2: BNC ← Naïve-Bayes structure;
3: k ← input;
4: stop = false;
5: t = 1;
6: repeat
7:       {i → j} = SelectEdgeWithMaxIncreaseInQuality();
8:       if {i → j} ≠ φ then
9:             BNC = BNC ∪ {i → j};
10:            RemoveInvalidEdgesFromSearchSpace();
11:      else
12:            Stop = true;
13:      end if
14:      t = t + 1;
15: until t = max_iterations or Stop
16: return BNC
17: End
```

Finally, we compare our ant-based Bayesian classification approach against a state-of-the-art classification algorithm: Support Vector Machine (SVM) [9]. SVM uses a completely different approach for representing classifiers, compared to the Bayesian network representation. An SVM maps the cases into a higher-dimensional feature space and then finds the best hyperplane for separating cases of different classes — best hyperplane is the one with the greatest possible gap separating cases of different classes in the data space. Although SVM often produces very good results in terms of predictive accuracy, it has the disadvantage that the produced classification model can hardly be interpreted by users, that is, it produces a black-box model. This is in contrast to the probabilistic graphic models produced by Bayesian network classifiers, which, in principle, can be interpreted by users.

6.2 Experimental setup

The experiments were carried out using the *stratified* ten-fold cross validation procedure. In essence, a dataset is divided into ten mutually exclusive partitions (folds), with approximately the same number of cases in each partition. Then each classification algorithm is run ten times, where each time a different partition is used as the test set and the other nine partitions are used as the training set. The results (accuracy rate on the test set) are then averaged and reported as the accuracy rate of the classifier. Since ABC-Miner is a stochastic algorithm, we run it ten times – using a different random seed to initialize the search each time – for each of the ten iterations of the cross-validation procedure (i.e., 100 runs in total, for each dataset). In the case of the deterministic algorithms, each one is run just once for each iteration of the cross-validation procedure. Results and analysis are presented in the next subsection.

The parameter configuration used in our experiments is shown in Table 2. Note that for the greedy (hill-climbing) algorithms (GBN, GHC-K2 and GHC-Acc), we refer to `max_iterations` as the maximum number of solution evaluations that the algorithm performs during the hill-climbing search. It is set to 500, which is equal to `max_iterations` multiplied by `colony_size`, which is used for ABC-Miner. For the sake of fair comparison, we limit each Bayesian classification algorithm to the same fixed number of solution evaluations to construct the BN classifier. However, the maximum number might not be utilized completely. Note that the ACO-based algorithms might use a smaller number of iterations if they converge earlier and the greedy algorithms might also stop earlier if they get stuck in a local optimum.

Table 2 Parameter settings used in experiments

	Parameter	Value
ABC-Miner	`max_iterations`	100
	`colony_size`	5
	`conv_ iterations`	10
	`max_parents`	3
Hill Climbing	`max_iterations`	500
	number of parents	3

Unlike ABC-Miner, the number of parents (k-dependencies) for the variables must be specified for the hill-climbing Bayesian classification algorithms. We set it to 3, which is the maximum number of parents used in our ABC-Miner algorithm, where the number of parents is selected dynamically at each iteration. Note that for SP-TAN (as for any TAN learning algorithm), besides the class node being a parent for all the input variables, the number of parents for each node in the network is at most one.

Concerning the cAnt-Miner algorithm, the parameter configuration used in our experiments is the same as the original implementation shown in [32]. The type of SVM algorithm used in the experiments was SMO, as implemented in the WEKA data mining tool, with polynomial function kernel and default parameters [43].

6.3 Datasets

The performance of ABC-Miner was evaluated using 25 public-domain datasets from the well-known UCI (University of California at Irvine) dataset repository [42]. The main characteristics of the datasets are shown in Table 3.

Datasets having continuous attributes were discretized in a pre-processing step, applying the C4.5-Disc algorithm [26] on the training set (i.e., without using the test set). Briefly, C4.5-Disc works as follows. For each continuous

Table 3 Description of datasets used in experimental results

Dataset	Cases	Attributes	Classes
balance scale	625	4	3
breast cancer (wisconsin)	286	9	2
car evaluation	1,728	6	4
chess (rook vs. pawn)	3,196	36	2
contraceptive method choice	1,473	9	3
statlog credit (australian)	690	14	2
statlog credit (german)	1,000	20	2
dermatology	366	33	6
glass	214	10	7
hayes-roth	160	4	3
heart (cleveland)	303	12	3
hepatitis	155	19	2
ionosphere	351	34	2
iris	150	4	3
monks	432	6	2
mushrooms	8,124	22	2
nursery	12,960	8	5
parkinsons	197	23	2
pima diabetes	768	8	2
post-operative patient	90	8	3
soybean	307	35	19
tic-tac-to	958	9	2
voting records	435	16	2
wine	178	13	3
zoo	101	17	7

attribute, a two-attribute dataset is constructed. The first attribute of the constructed dataset contains the values of the numeric attribute to be discretized, and the second is the class attribute. The C4.5 decision tree generation algorithm is then applied to this reduced dataset. Thus, C4.5 constructs a decision tree in which all internal nodes refer to the attribute being discretized. Each path from the root to a leaf node in the tree corresponds to the definition of a categorical interval.

7 Results and analysis

7.1 Predictive accuracy

Tables 4 reports the mean and the standard error ($mean \pm standard\ error$) of predictive accuracy values obtained by ten-fold cross validation for the 25 datasets, for the deterministic algorithms and for the (stochastic) ACO algorithms. The highest accuracy for each dataset (across all the algorithms) is shown in bold face. The predictive accuracy results comparing ABC-Miner and SVM are discussed separately later in this section, where we also briefly illustrate the potential for Bayesian network classifiers to be interpreted by users.

Table 4 Predictive accuracy % (*mean* ± *standard error*). The highest accuracy for each dataset is shown in bold face.

Dataset	Naïve-Bayes	TAN	GBN	GHC-K2	SP-TAN	GHC-Acc	ABC-Miner	ABC-Miner^{-}	*c*Ant-Miner
bal	**91.2 ± 0.8**	84.4 ± 0.9	85.6 ± 0.6	83.7 ± 0.8	78.7 ± 0.8	85.2 ± 0.9	85.6 ± 0.9	85.6 ± 0.9	86.5 ± 0.9
bcw	92.1 ± 0.9	**95.4 ± 0.9**	93.8 ± 0.9	94.2 ± 0.9	94.4 ± 0.9	94.2 ± 0.9	**95.4 ± 0.9**	**95.4 ± 0.9**	92.5 ± 0.9
car	85.3 ± 0.9	93.6 ± 0.6	86.2 ± 0.9	95.6 ± 0.6	95.6 ± 1.2	96.8 ± 0.5	**97.2 ± 0.3**	**97.2 ± 0.3**	87.8 ± 0.3
chess	88.2 ± 1.2	92.5 ± 1.2	89.8 ± 0.9	91.8 ± 0.9	94.6 ± 0.9	95.6 ± 1.2	**97.6 ± 0.9**	95.9 ± 0.6	96.5 ± 0.6
cmc	52.2 ± 1.2	49.8 ± 1.2	49.8 ± 1.2	67.1 ± 0.6	58.3 ± 1.2	**68.5 ± 0.8**	68.0 ± 0.6	67.5 ± 0.4	54.1 ± 0.4
crd-a	77.5 ± 1.2	85.1 ± 0.9	85.7 ± 0.9	85.1 ± 0.6	84.2 ± 0.9	86.5 ± 0.6	**87.8 ± 0.9**	**87.8 ± 0.9**	83.3 ± 1.2
crd-g	**75.6 ± 0.9**	73.7 ± 1.2	**75.6 ± 1.2**	**75.6 ± 0.9**	73.5 ± 0.9	72.1 ± 0.9	73.7 ± 0.9	73.7 ± 0.9	69.8 ± 1.2
drm	96.2 ± 0.6	97.8 ± 0.9	97.2 ± 0.9	98.6 ± 0.6	98.6 ± 3.5	98.1 ± 0.6	**99.1 ± 0.9**	98.6 ± 1.2	98.1 ± 3.1
glass	74.2 ± 1.2	78.4 ± 0.9	80.7 ± 0.9	90.4 ± 0.9	72.9 ± 0.9	**94.6 ± 0.9**	93.4 ± 1.2	93.4 ± 1.2	83.2 ± 2.8
hay	80.0 ± 2.8	77.9 ± 3.1	**83.1 ± 3.5**	80.0 ± 2.8	80.4 ± 2.8	82.1 ± 2.8	80.0 ± 3.1	80.0 ± 3.1	82.1 ± 1.6
hrt-c	62.7 ± 2.2	68.8 ± 2.5	66.7 ± 2.2	83.1 ± 2.2	69.2 ± 3.1	**83.8 ± 3.1**	**83.8 ± 3.1**	83.1 ± 2.8	79.7 ± 0.3
hep	81.9 ± 1.6	**78.0 ± 1.6**	83.2 ± 2.1	78.0 ± 1.6	77.3 ± 2.5	77.3 ± 2.5	**78.0 ± 1.6**	**78.0 ± 1.6**	**78.2 ± 0.6**
iono	82.6 ± 0.6	90.5 ± 0.9	90.5 ± 0.6	91.7 ± 0.6	91.2 ± 0.3	91.7 ± 0.3	**94.5 ± 0.3**	**94.5 ± 0.3**	93.2 ± 1.2
iris	**96.2 ± 1.5**	94.2 ± 1.8	92.9 ± 0.8	**96.2 ± 0.8**	92.1 ± 0.8	94.2 ± 0.8	**96.2 ± 0.6**	**96.2 ± 0.6**	94.1 ± 0.3
monk	**61.6 ± 0.6**	58.8 ± 0.6	**61.6 ± 0.9**	49.5 ± 0.9	59.3 ± 0.6	58.8 ± 0.6	53.8 ± 0.9	52.0 ± 1.2	60.6 ± 0.9
mush	95.8 ± 0.9	98.2 ± 0.6	96.1 ± 1.2	96.9 ± 0.9	98.2 ± 0.9	97.5 ± 0.9	**98.8 ± 0.6**	98.0 ± 0.3	97.1 ± 2.2
nurs	90.1 ± 0.9	94.3 ± 0.9	90.1 ± 0.9	96.5 ± 0.6	96.5 ± 0.6	97.0 ± 0.6	**98.2 ± 0.9**	98.0 ± 0.9	97.0 ± 1.5
park	84.5 ± 2.5	91.7 ± 2.2	84.5 ± 2.5	84.5 ± 2.2	92.1 ± 2.5	83.4 ± 2.5	**94.2 ± 2.5**	92.3 ± 2.2	**94.2 ± 1.9**
pima	75.4 ± 1.2	77.8 ± 1.5	76.2 ± 1.5	76.2 ± 1.5	77.8 ± 1.5	77.8 ± 1.5	77.8 ± 1.5	77.8 ± 1.5	**78.9 ± 0.9**
pop	68.2 ± 0.6	64.1 ± 0.6	66.6 ± 0.6	71.6 ± 0.9	75.5 ± 0.9	**78.3 ± 0.6**	76.8 ± 0.9	76.8 ± 0.9	71.7 ± 0.9
soy	91.4 ± 1.2	**95.6 ± 1.2**	93.2 ± 0.9	94 ± 1.2	94.4 ± 1.2	94.2 ± 1.2	**95.6 ± 1.2**	95.3 ± 0.9	93.8 ± 0.9
ttt	70.3 ± 0.3	76.6 ± 0.3	70.3 ± 0.6	71.2 ± 0.6	80.6 ± 0.6	72.5 ± 0.6	**86.4 ± 0.3**	84.9 ± 0.3	79.7 ± 0.3
vot	90.3 ± 0.9	92.1 ± 0.9	90.3 ± 0.6	91.7 ± 1.2	93.1 ± 1.2	90.0 ± 1.2	**94.6 ± 1.2**	93.7 ± 1.1	92.6 ± 1.1
wine	95.6 ± 1.2	97.3 ± 0.9	97.3 ± 0.9	98.5 ± 0.9	98.5 ± 0.9	98.1 ± 0.9	98.5 ± 1.5	98.5 ± 1.5	98.1 ± 1.5
zoo	94.2 ± 0.6	97.3 ± 0.3	95.1 ± 0.9	97.3 ± 0.3	**98.0 ± 0.3**	**98.0 ± 0.3**	**98.0 ± 0.1**	**98.0 ± 0.1**	**98.0 ± 0.1**

As can be observed in Table 4, ABC-Miner achieved the highest predictive accuracy amongst all algorithms in 16 datasets; followed by ABC-Miner$^-$, which achieved the best predictive accuracy in seven datasets. *c*Ant-Miner, the ACO algorithm for classification rule discovery, achieved the highest predictive accuracy in four datasets. Naïve-Bayes achieved the highest predictive accuracy in four datasets, while TAN in three datasets and GBN in three datasets. SP-TAN, the extended TAN learning algorithm, achieved the best predictive accuracy in 1 dataset. GHC-K2 and GHC-Acc, which were customized for the experiments, achieved the best predictive accuracy amongst all the algorithms in two and five datasets, respectively.

Table 5 shows the results of the statistical significance tests according to the non-parametric Friedman test with Holm's post-hoc test [16,20], where $\alpha = 0.05$, with respect to the average rank. The average rank for a given algorithm g is obtained by first computing the rank of g on each dataset individually. The individual ranks are then averaged across all datasets to obtain the overall average rank. Note that the lower the value of the rank, the better the algorithm. ABC-Miner obtained the best overall ranking: 2.6.

For each algorithm, the first column in Table 5 after the algorithms' names shows its average rank, the second column shows the p-value (adjusted on the Holm critical value) of the statistical test when its average rank is compared to the average rank of ABC-Miner (the control algorithm with the best rank). An entry where the p-value is less than 0.05 indicates that the control algorithm, ABC-Miner, is better than its corresponding algorithm at the statistical significance level of 0.05.

Table 5 The non-parametric Friedman statistical test results with Holm's post-hoc test.

Algorithm	Average Rank	Adjusted p-value
ABC-Miner (control)	2.6	–
ABC-Miner$^-$	3.2	0.3801
GHC-Acc	4.6	0.0782
*c*Ant-Miner	4.7	0.0048
SP-TAN	5.0	0.0016
GHC-K2	5.4	2.1E-4
TAN	5.9	1.8E-5
GBN	6.4	5.4E-7
Naïve-Bayes	7.1	5.3E-9

Employing ACO with local search for learning a Bayesian network classifier structure, along with the use of predictive accuracy as the objective function to be optimized, produced the best results. To analyze the effects of different components of ABC-Miner on the effectiveness of the algorithm, we carry out the following comparisons:

- **GHC-Acc vs. GHC-K2**: To isolate the effect of the objective function, we compare the same search strategy (GHC) with two different objective functions. The use of predicative accuracy performs better than the use of K2 when learning a BN classifier, even without using ACO as a search strategy. GHC-Acc reached higher accuracy than GHC-K2 in 16 datasets and obtained a better overall accuracy ranking that is statistically significant at the 5% level.
- **ABC-Miner vs. GHC-Acc**: The results show that the use of an ACO meta-heuristic performs better than the use of a greedy hill-climbing search in building BN classifiers, even when both have the same objective function: predictive accuracy. ABC-Miner reached higher accuracy than GHC-Acc in 17 datasets and obtained a better overall accuracy ranking. However, the statistical test shows that the ranking of the ACO-based algorithm is not significant at the level of 5%.
- **ABC-Miner vs. GHC-K2**: Combining ACO with accuracy as an objective function obtained much better results than the greedy search with K2 as objective function. ABC-Miner reached higher predictive accuracy results than GHC-K2 in 21 datasets. The overall accuracy ranking obtained by ABC-Miner is statistically better than the result obtained by GHC-K2 at the significance level of 5%.
- **TAN vs. SP-TAN**: Using predictive accuracy to select the edge to add as a parent in the TAN learning procedure makes the SP-TAN algorithm perform better than the conventional TAN learning algorithm, which selects edges using the conditional mutual information between variables. SP-TAN reached higher accuracy than TAN in 15 datasets, and a higher accuracy ranking than TAN that is statistically significant at the 5% level.
- **GHC-Acc vs. SP-TAN**: Although GHC-Acc obtained a slightly better overall ranking than SP-TAN, this difference is not statistically significant at the 5% level. GHC-Acc builds a BN classifier in which a node can have up to 3 parents, and it reached higher accuracy than SP-TAN in only 13 datasets out of 25, while in a TAN a node can have only 1 parent.
- **ABC-Miner vs. ABC-Miner$^-$**: Using local search has improved the effectiveness of the algorithm, since ABC-Miner obtained a better accuracy ranking compared to ABC-Miner$^-$. However, this difference is not statistically significant, since the use of the local search improved the performance of ABC-Miner compared to ABC-Miner$^-$ only in 10 datasets.
- **ABC-Miner vs. *c*Ant-Miner**: The ACO algorithm for learning BN classifiers performs better than its corresponding ACO algorithm for learning classification rules in 17 datasets, obtaining a better overall accuracy ranking that is statistically significant at the 5% level.

According to the aforementioned comparisons, we conclude that the use of the ACO-meta heuristics outperformed GHC in learning BN classifiers. This result is statistically significant as shown in the results' comparisons of ABC-Miner and ABC-Miner− against GHC-K2, SP-TAN, GBN and TAN. The use of predictive accuracy has the highest impact in increasing the effectiveness

of the algorithm, whether it is used with ACO or GHC. This claim is statistically supported by the comparisons of ABC-Miner vs. GHC-K2, GHC-Acc vs. GHC-K2 and SP-TAN vs. TAN, since the use of predictive accuracy as an objective function with any search algorithm outperforms its corresponding search algorithm using another conventional scoring function (such as K2 and conditional mutual information) with statistical significance. On the other hand, using local search has a relatively low impact on the predictive performance, which is seen in the comparison of ABC-Miner against ABC-Miner−, which shows no statistically significant difference. Combining ACO as a search strategy while utilizing local search, along with the predictive accuracy as an objective function, produced the best results.

Finally, we compare the predictive performance of ABC-Miner to SVM as follows. The results for the two algorithms were obtained using 20-fold cross-validation using the same fold partitioning. Consequently, for a given dataset, the results for each fold, for the two algorithms, can be considered a matched pair. Therefore, for each individual dataset, we apply a two-tailed Wilcoxon matched-pairs statistical test to the results of the 20 folds for the two algorithms to determine if there is a statistically significant difference for that dataset. Although the use of 10-fold cross validation is more common, we performed the statistical evaluation using the results of the 20-fold cross validation to increase the power of the test.

Table 6 shows the results of these Wilcoxon tests for each of the 25 datasets. The result shown in bold-face is the highest one between the two algorithms for a given dataset. A result shown with underline indicates a statistically significant difference at the 5% level. Accordingly, ABC-Miner reached higher predictive accuracy results than SVM in 12 datasets; in 10 of which the results were significantly better at the 5% level. On the other hand, SVM performed better than ABC-Miner in 13 datasets; in 6 of which the results were significantly better at the 5% level.

Broadly speaking, Support Vector Machines (SVMs) often obtain somewhat higher predictive accuracies than other types of classification algorithms, although this is not true for all types of datasets (as shown in Table 6). However, the classifier built by an SVM is in general a "black box" from the perspective of users — i.e., the output of an SVM algorithm can hardly be interpreted by users. By contrast, BN classifiers produce a graphical model of the dependencies between variables that can be directly interpreted by users, which is an advantage in many application domains. For a review of the importance of comprehensible classification models, see [17,22,30,34].

A sample of the graphical models produced by ABC-Miner from two datasets used in our experiments is shown in Figure 2. As shown, ABC-Miner produces BAN classifiers, where the class variable is the parent of all other input variables (the predictor variables), and each predictor variable usually has more than one parent node besides the class node.

In general, parent variables with a high number of children can be considered to be important variables; the values of those parent variables influence the values of several other related variables. For instance, in the Car Evalua-

Table 6 ABC-Miner vs. SVM predictive accuracy % (*mean* ± *standard error*). The result shown in bold-face is the higher one between the two algorithms for a given dataset. The underlined result indicates a statistically significant difference at $\alpha = 0.05$, according to a two-tailed Wilcoxon matched-pairs statistical test.

Dataset	ABC-Miner	SVM
bal	84.7 ± 0.7	**88.4 ± 0.9**
bcw	95.1 ± 0.6	**97.8 ± 1.1**
car	**97.4 ± 0.2**	94.4 ± 0.6
chess	**95.8 ± 0.5**	95.2 ± 0.7
cmc	**66.7 ± 0.4**	51.3 ± 0.9
crd-a	**87.6 ± 0.6**	84.9 ± 0.8
crd-g	73.5 ± 0.8	**75.4 ± 0.7**
drm	**98.4 ± 0.9**	97.4 ± 1.2
glass	**93.6 ± 0.8**	82.4 ± 0.7
hay	80.8 ± 1.8	**81.9 ± 1.2**
hrt-c	**83.5 ± 2.1**	73.2 ± 1.6
hep	**77.8 ± 1.2**	74.2 ± 1.6
iono	**94.8 ± 0.2**	92.1 ± 0.4
iris	96.1 ± 0.4	**96.4 ± 0.3**
monk	52.3 ± 0.8	**63.6 ± 1.2**
mush	97.8 ± 0.2	**100 ± 0.0**
nurs	**97.9 ± 0.6**	93.4 ± 0.9
park	92.4 ± 1.8	**94.7 ± 1.6**
pima	**77.6 ± 0.8**	74.7 ± 1.2
pop	**76.6 ± 0.6**	72.4 ± 0.9
soy	95.6 ± 0.6	**97.2 ± 0.8**
ttt	84.8 ± 0.2	**96.1 ± 1.1**
vot	94.3 ± 1.1	**95.6 ± 1.2**
wine	98.3 ± 1.2	**98.8 ± 0.6**
zoo	98.1 ± 0.4	**98.5 ± 0.6**

tion dataset, the value of the "safety" variable directly influence the values of its three child variables; and in the Nursery dataset, the "housing" variable is a parent of four variables. The effect of a parent variable on the state of its child variables is quantified by the CPT computed for each of those children.

Note that in general the value of the class variable for a given case is more dependent on the values of the case's input variables that have several children in the BN classifier than on the values of the case's variables with no children in the BN classifier – like the "persons" and the "social" variables in the Car Evaluation and the Nursery datasets, respectively. Changing the value of an input variable with several children can have a relatively large effect on the

class posterior probability of the case, compared to the relatively small effect caused by changing the value of an input variable with no children, according to Equations 1 and 4.

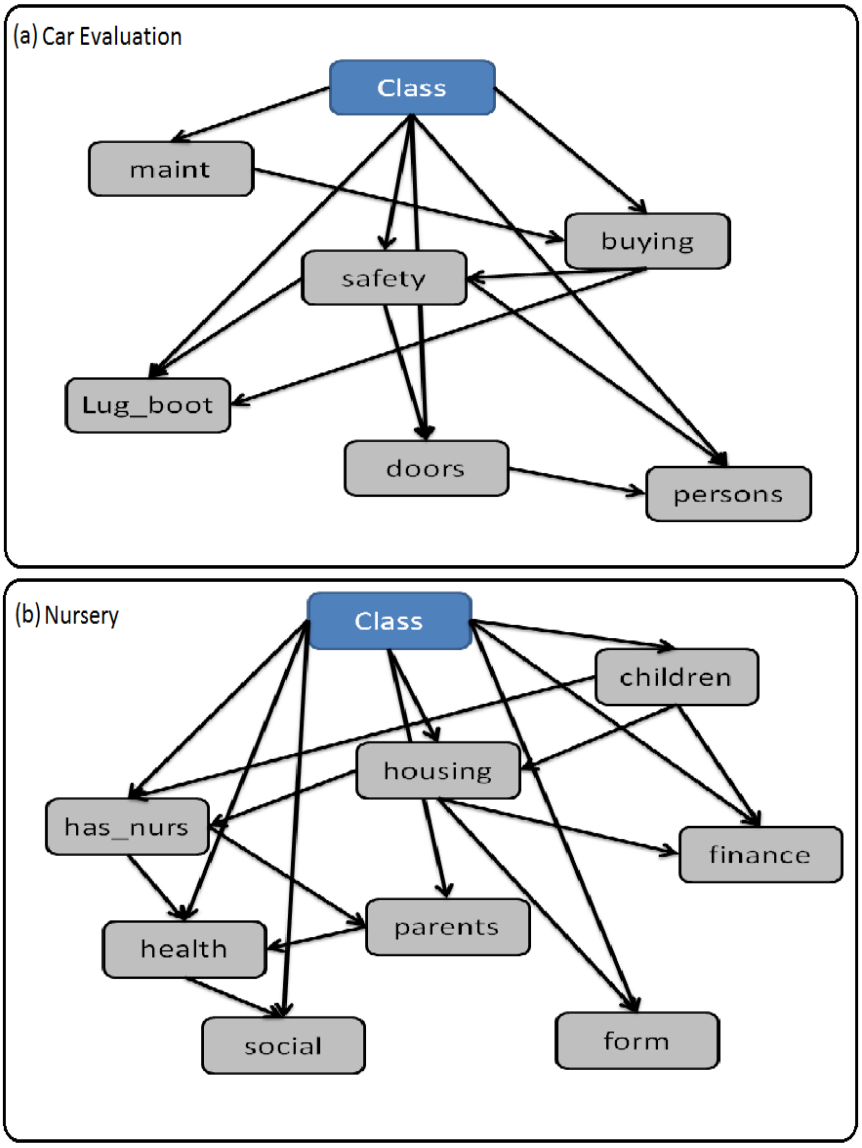

Fig. 2 Bayesian network classifiers output from running ABC-Miner on two datasets (a) Car Evaluation and (b) Nursery.

7.2 Computational time

The running time (in seconds) of each algorithm (averaged over the 10 cross-validation folds) is reported in Table 7 for all datasets. The ratio between each algorithm's running time and ABC-Miner's running time is reported in the column with "Ratio" header.

According to the running times, the use of the predictive accuracy function leads to a longer execution time than the use of K2 as an objective function. This is expected, since K2 is a decomposable measure [8]. This is shown when comparing GHC-K2's and GBN's running time to GHC-Acc and ABC-Miner. Comparing GHC-Acc to ABC-Miner, the ratio of GHC-Acc's time to ABC-Miner's is 1.39, although both were given the same computational budget (i.e., both evaluate the same maximum number of candidate solutions). The fact that GHC-Acc takes more time than ABC-Miner can be explained by the possibility that ACO converges before reaching the maximum number of iterations. On the other hand, the use of the local search procedure increases the running time by about 40%. In addition, overall, ABC-Miner is slightly faster than *c*Ant-Miner.

As expected, TAN is by far the fastest algorithm in Table 7, but its predictive performance is much worse than ABC-Miner (Section 7.1). TAN does not offer a good trade-off between accuracy and speed, since predictive accuracy is usually considered much more important than the running time in the classification task of data mining. SP-TAN, the second best in terms of computational time, provides a good trade-off between the predictive accuracy and the running time of the algorithm.

Table 7 Running time (in seconds).

Dataset	TAN		GBN		GHC-K2		SP-TAN		GHC-Acc		*c*Ant-Miner		ABC-Miner$^-$		ABC-Miner
	time	ratio	time	ratio	time	ratio	time	ratio	time	ratio	time	ratio			
bal	21	0.01	1500	0.58	1000	0.38	1200	0.46	2000	0.77	3200	1.23	2000	0.77	2600
bcw	34	0.02	1200	0.6	900	0.45	1100	0.55	2500	1.25	3000	1.5	1500	0.75	2000
car	27	0.03	700	0.7	1300	1.3	650	0.65	5700	5.7	400	0.4	900	0.9	1000
chess	590	0.02	6800	0.18	7000	0.19	4400	0.12	67000	1.81	3000	0.08	25000	0.68	37000
cmc	78	0.01	2100	0.39	800	0.15	1200	0.22	5200	0.96	6600	1.22	5000	0.93	5400
crd-a	21	0.01	2200	1	800	0.36	2000	0.91	1100	0.5	4500	2.05	1100	0.5	2200
crd-g	36	0.01	1400	0.31	900	0.2	400	0.09	1300	0.29	6100	1.36	4300	0.96	4500
drm	26	0.01	500	0.18	700	0.25	550	0.2	4600	1.64	2000	0.71	2500	0.89	2800
glass	8	0	400	0.08	300	0.06	100	0.02	700	0.14	7200	1.44	4000	0.8	5000
hay	6	0.02	15	0.05	10	0.03	10	0.03	200	0.67	250	0.83	100	0.33	300
hrt-c	21	0.01	400	0.1	100	0.02	150	0.04	2200	0.52	4800	1.14	3200	0.76	4200
hep	1	0.01	50	0.25	70	0.35	20	0.1	100	0.5	150	0.75	100	0.5	200
iono	13	0	450	0.17	380	0.14	250	0.09	2200	0.81	4200	1.56	2400	0.89	2700
iris	1	0.01	10	0.1	10	0.1	10	0.1	100	1	80	0.8	10	0.1	100
monk	70	0.02	470	0.11	380	0.09	220	0.05	3000	0.7	4000	0.93	600	0.14	4300
mush	180	0.01	6000	0.4	8000	0.53	2000	0.13	18000	1.2	12000	0.8	10000	0.67	15000
nurs	90	0.01	6500	0.76	4000	0.47	6500	0.76	9000	1.06	9000	1.06	5000	0.59	8500
park	26	0.01	400	0.19	300	0.14	160	0.08	6000	2.86	3800	1.81	200	0.1	2100
pima	36	0.01	120	0.04	280	0.08	120	0.04	3000	0.88	3600	1.06	1800	0.53	3400
pop	1	0	14	0.02	10	0.01	10	0.01	100	0.14	800	1.14	400	0.57	700
soy	24	0.02	320	0.25	190	0.15	120	0.09	4200	3.23	1500	1.15	500	0.38	1300
ttt	24	0.01	3800	1.46	1900	0.73	2300	0.88	2300	0.88	1800	0.69	1800	0.69	2600
vot	35	0.01	500	0.12	700	0.17	200	0.05	1800	0.44	4000	0.98	1600	0.39	4100
wine	27	0.11	200	0.8	300	1.2	100	0.4	700	2.8	450	1.8	100	0.4	250
zoo	12	0.06	110	0.55	70	0.35	110	0.55	800	4	200	1	10	0.05	200
avg. ratio		0.02		0.37		0.31		0.26		1.39		1.12		0.57	

8 Concluding remarks

In this paper, we presented and extended ABC-Miner, an ant-based algorithm for learning Bayesian network classifiers. Empirical results showed that ABC-Miner significantly outperforms the well-known Naïve-Bayes, TAN, SP-TAN and GBN algorithms in terms of predictive accuracy. The results also show that the use of predictive accuracy as an optimization objective function for building a BN classifier leads to higher predictive performance than the K2 scoring function that is used to build general BNs. The ACO meta-heuristic was shown to be better than greedy hill-climbing in the search process even if both used predictive accuracy as a quality measure for the constructed BN classifier. However, using predictive accuracy as an objective function caused the highest impact in improving the results. On the other hand, the automatic selection of the maximum number of k-parents value for each node makes ABC-Miner more adaptive and autonomous than conventional algorithms for learning BN classifiers, where the k-dependencies value is set by the user and fixed for all nodes of the BN during the entire run of the algorithm.

As future work, we would like to explore the effect of using different scoring functions for computing the heuristic value used by ABC-Miner, as well as exploring other measures to evaluate the quality of a constructed BN classifier during the search process. Another research direction is to use ant colony optimization to construct Bayesian multi-nets for classification.

References

1. Bonabeau, E., Dorigo, M., Theraulaz., G. : Swarm Intelligence: From natural to artificial systems. *Oxford University Press*, New York, NY, USA, (1999).
2. Buntine, W. : Theory refinement on Bayesian networks. *17th Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA, USA, pp. 52–60 (1991).
3. Campos, L.M. , Gámez, J.A. , Puerta, J.M. : Ant Colony Optimization for Learning Bayesian Networks. *Journal of Approximate Reasoning*, Vol. 31, pp. 291-311 (2002).
4. Cheng, J. and Greiner, R. : Comparing Bayesian Network Classifiers. *15th Annual Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA, USA, pp. 101–108 (1999).
5. Cheng, J. and Greiner, R. : Learning Bayesian Belief Network Classifiers: Algorithms and System. *14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, Springer, London, UK, pp. 141–151 (2001).
6. Chickering, D., Geiger, M., Heckerman, D. : Learning Bayesian Networks is NP-Hard. *Advanced Technologies Division, Microsoft Corporation, Redmond, WA, Technical Report*, (1994).
7. Colorni, A., Dorigo, M., Maniezzo, V. : Distributed Optimization by Ant Colonies. *1st European Conference on Artificial Life*, MIT Press, Cambridge, MA, pp. 134–142 (1992).
8. Cooper, G.F. and Herskovits, E. : A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, Vol. 9, pp. 309–348 (1992).
9. Cristianini, N., and Shawe-Taylor, J. : An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. *Cambridge University Press*, Cambridge, UK (2000).
10. Daly, R., Shen, Q., Aitken, S. : Using Ant Colony Optimization in Learning Bayesian Network Equivalence Classes. *UK Workshop on Computational Intelligence (UKCI)*, AAAI Press, Palo Alto, CA, USA, pp. 111–118 (2006).

11. Daly, R. and Shen, Q. : Learning Bayesian Network Equivalence Classes with Ant Colony Optimization. *Journal of Artificial Intelligence Research*, Vol. 35, pp. 391–447 (2009).
12. Dorigo, M. Di Caro, G. : The Ant Colony Optimization Meta-Heuristic. *New Ideas in Optimization*, McGraw-Hill, Vol. 2 pp. 11–32 (1999).
13. Dorigo, M., Di Caro, G., Gambardella, L. M. : *Ant Algorithms for Discrete Optimization. Artificial Life*, Vol. 5(2), pp. 137–172. (1999)
14. Dorigo, M., Maniezzo, V., Colorni, A. : Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, Vol. 26, pp. 29–41 (1996).
15. Dorigo, M. and Stützle, T. : Ant Colony Optimization. *MIT Press*, Cambridge, MA, USA, (2004).
16. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Datasets. *Journal of Machine Learning Research*, Vol. 7, pp. 1–30 (2006).
17. Freitas, A.A., Wieser, D.C. and Apweiler, R. : On the Importance of Comprehensible Classification Models for Protein Function Prediction. *ACM/IEEE Transactions on Computational Biology and Bioinformatics* Vol. 7, pp. 172–182 (2010).
18. Friedman, N., Geiger, D., Goldszmidt, M. : Bayesian Network Classifiers. *Machine Learning*, Vol. 29, pp. 131–161 (1997).
19. Friedman, N., and Goldszmidt, M. : Learning Bayesian Networks with Local Structure. *Learning in Graphical Models, Norwell, MA: Kluwer*, pp. 421–460 (1998).
20. García, S. and Herrera, F. : An Extension on Statistical Comparisons of Classifiers over Multiple Datasets for all Pairwise Comparisons. *Journal of Machine Learning Research*, Vol. 9, pp. 2677–2694 (2008).
21. Heckerman, D., Geiger, D., Chickering, D.M. : Learning Bayesian Networks: the Combination of Knowledge and Statistical Data. *Machine Learning*, Vol. 20, pp. 197–244 (1995).
22. Huysmans, J., Dejaeger, K., Mues, C., Vanthienen, J., and Baesens, B. : An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems* Vol. 51, pp. 141–154. (2011).
23. Jaiwei, H., & Kamber, M. : Data Mining: Concepts and Techniques. *Morgan Kaufmann*, San Francisco, CA, USA, (2006)
24. Jiang, L., Wang, D., Cai, Z., Yan., X. : Survey of Improving Naive Bayes for Classification. *International Conference on Advanced Data Mining and Applications*, Springer Heidelberg, LNCS 4632, pp. 134–145 (2007).
25. Ji, J., Hu, R., Zhang, H., Liu, C. : A Hybrid Method for Learning Bayesian Networks based on Ant Colony Optimization. *Applied Soft Computing*, Vol. 11, pp. 3373–3384 (2011).
26. Kohavi, R. and Sahami, M. : Error-based and Entropy-based Discretization of Continuous Features. *International ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, AAAI Press, Palo Alto, CA, USA, pp. 114–119. (1996).
27. Keogh, E., Pazzani, M.: Learning Augmented Bayesian Classifiers: A Comparison of Distribution-based and Classification-based Approaches. *International Workshop on Artificial Intelligence and Statistics*, Morgan Kaufmann, San Francisco, CA, USA pp. 225–230 (1999).
28. Langley, P., Iba, W. and Thompson, K. : An Analysis of Bayesian Classifiers. *10th National Conference on Artificial Intelligence (AAAI-92)*, AAAI Press, Palo Alto, CA, USA, pp. 223–228 (1992).
29. Martens, D., Backer, M.D., Haesen, R., Vanthienen, J., Snoeck, M., Baesens, B. : Classification with Ant Colony Optimization. *IEEE Transactions on Evolutionary Computation*, Vol. 11, pp. 651–665 (2007).
30. Marteens, D., Vanthienen, J., Verbeke, W., and Baesens, B. : Performance of Classification Models from a User Perspective. *Decision Support Systems* Vol. 51, pp. 782–793 (2011)
31. Martens, D. , Baesens, B., Fawcett T. : Editorial Survey: Swarm Intelligence for Data Mining. *Machine Learning*, Vol. 82, pp. 1–42 (2011).
32. Otero, F., Freitas, A.A., Johnson, C.G. : cAnt-Miner: an Ant Colony Classification Algorithm to Cope with Continuous Attributes. *Sixth International Conference on Ant Colony Optimization and Swarm Intelligence*, Springer Heidelberg, Germany, LNCS 5217, pp. 48–59 (2008).

33. Parpinelli, R.S., Lopes, H.S., Freitas, A.A. : Data Mining with an Ant Colony Optimization Algorithm. *IEEE Transactions on Evolutionary Computation*, Vol. 6(4), pp. 321–332 (2002).
34. Pazzani, M.J., Mani, S., and Shankle, W.R. : Acceptance of Rules Generated by Machine Learning among Medical Experts. *Methods of Information in Medicine*, Vol. 40, pp. 380–385 (2001).
35. Pinto, P.C., Ngele, A., Dejori, M., Runkler, T.A., Costa, J.M. : Learning of Bayesian networks by a local discovery ant colony algorithm. *IEEE World Congress on Computational Intelligence*, IEEE Press, Piscataway, NJ, USA, pp. 2741–2748 (2008).
36. Pinto, P.C., Ngele, A., Dejori, M., Runkler, T.A., Costa, J.M. : Using a Local Discovery Ant Algorithm for Bayesian Network Structure Learning. *IEEE Transactions on Evolutionary Computation*, Vol. 13, pp. 767–779 (2009).
37. Salama, K. M., and Abdelbar, A. M. : Extensions to the Ant-Miner Classification Rule Discovery Algorithm. *7th International Conference on Swarm Intelligence (ANTS 2010)*, Springer Heidelberg, LNCS 6234, pp. 43–50 (2010),
38. Salama, K.M., Abdelbar, A.M., Freitas A.A. : Multiple Pheromone Types and Other Extensions to the Ant-Miner Classification Rule Discovery Algorithm. *Swarm Intelligence*, Vol. 5, pp. 149–182 (2011).
39. Salama, K. M. and Abdelbar, A. M. Exploring Different Rule Quality Evaluation Functions in ACO-based Classification Algorithms. *IEEE Symposium on Swarm Intelligence (SIS)*, IEEE Press, Piscataway, NJ, USA, pp. 1–8 (2011).
40. Salama, K.M., Abdelbar, A. M., Otero, F.E, Freitas, A.A. : Utilizing Multiple Pheromones in an Ant-based Algorithm for Continuous-Attribute Classification Rule Discovery. *Applied Soft Computing*, Vol. 13, pp. 667–675 (2013),
41. Salama, K.M. and Freitas A.A. : ABC-Miner: an Ant-based Bayesian Classification Algorithm. *8th International Conference on Swarm Intelligence (ANTS 2012)*, Springer Heidelberg, LNCS 7461, pp. 13–24 (2012).
42. UCI Repository of Machine Learning Databases. Retrieved Oct 2011 from, URL:http://www.ics.uci.edu/ mlearn/MLRepository.html
43. Witten, H., and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques, Second Edition. *Morgan Kauffman*, San Francisco, CA, USA (2005).
44. Yanghui, Wu., McCall, J., Corne, D. : Two Novel Ant Colony Optimization Approaches for Bayesian Network Structure Learning. *IEEE World Congress on Evolutionary Computation (CEC 2010)*, IEEE Press, Piscataway, NJ, USA, pp. 1–7 (2010).
45. Yang, S. : Comparison of Score Metrics for Bayesian Network Learning. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, Vol. 32, pp. 419–428 (2002).