Memetic Computing manuscript No.
(will be inserted by the editor)

ABC-Miner+: Constructing Markov Blanket Classifiers with Ant Colony Algorithms

Khalid M. Salama · Alex A. Freitas

15-09-2013

Abstract ABC-Miner is a Bayesian classification algorithm based on the Ant Colony Optimization (ACO) meta-heuristic. The algorithm learns Bayesian network Augmented Naïve-Bayes (BAN) classifiers, where the class node is the parent of all the nodes representing the input variables. However, this assumes the existence of a dependency relationship between the class variable and *all* the input variables, and this relationship is always a type of "causal" (rather than "effect") relationship, which restricts the flexibility of the algorithm to learn. In this paper, we extended the ABC-Miner algorithm to be able to learn the *Markov blanket* of the class variable. Such a produced model has a more flexible Bayesian network classifier structure, where it is not necessary to have a (direct) dependency relationship between the class variable and each of the input variables, and the dependency between the class and the input variables varies from "causal" to "effect" relationships. In this context, we propose two algorithms: ABC-Miner+$_1$, in which the dependency relationships between the class and the input variables are defined in a separate phase before the dependency relationships among the input variables are defined, and ABC-Miner+$_2$, in which the two types of dependency relationships in the Markov blanket classifier are discovered in a single integrated process. Empirical evaluations on 33 UCI benchmark datasets show that our extended algorithms outperform the original version in terms of predictive accuracy, model size and computational time. Moreover, they have shown a very competitive performance against other well-known classification algorithms in the literature.

Keywords: Ant Colony Optimization (ACO), Data Mining, Classification, Bayesian Networks, Markov Blanket Classifiers .

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

1 Introduction

Ant Colony Optimization (ACO) is a meta-heuristic for solving combinatorial optimization problems, inspired by the observation of the behavior of biological ant colonies [14]. One of the fields in which ACO has been successfully applied is data mining, which involves finding hidden patterns and constructing analytical models from real-world datasets [53]. Classification is one of the widely studied data mining tasks, where the aim is to discover, from labeled cases (instances), a model that can be used to predict the class of unlabeled cases. There are many types of classification methods [53], but in this paper we focus on Bayesian network (BN) classifiers.

BN classifiers model the (in)dependency-relationships between the input domain variables given the class variable by means of a probabilistic network [18], which is used to predict the class of a case by computing the class with the highest posterior probability given the case's predictor attribute values. Since learning the optimal BN structure from a dataset is $\mathcal{NP}$-hard [5,6], stochastic heuristic search algorithms – such as ACO – can be a good alternative to build high-quality models, in terms of predictive accuracy and network size, within an acceptable computational time. Developing ACO-based algorithms to learn BN classifiers is the research topic addressed in this work.

We have recently introduced ABC-Miner [45,49], as an Ant-based Bayesian Classification algorithm that learns the structure of a Bayesian network Augmented Naïve-Bayes (BAN), where the class node is the parent of all the input variables, and at most k parents are allowed for each variable in the network. The ABC-Miner algorithm showed predictive effectiveness compared to other Bayesian classification algorithms, namely: Naïve-Bayes, TAN and GBN [45, 49]. However, the BAN structure produced by ABC-Miner has two limitations. First, it assumes there is a dependency relationship between the class and *all* the input variables. However, this assumption is unrealistic and may harm the classification effectiveness in the domains where there are redundant or irrelevant input variables to the class variable prediction. Second, this dependency relationship between the class and the input variables is only specified as a "causal" relationship, i.e., the class variable is always a parent to the input variables. This restricts the flexibility of the algorithm to discover other structures in constructing BN classification models, where the relationship between the class and an input variable is an" effect" relationship — i.e., the class variable can be a child to some input variables.

In this paper, we extend our ABC-Miner algorithm to learn more flexible BN classifier structures, where it is not necessary to have a (direct) dependency relationship between the class variable and each of the input variables. In addition, we allow the dependency between the class and the input variables to vary from "causal" to "effect" relationships, where the class variable can be a parent or a child of an input node. The produced model is called the *Markov blanket* (MB) of the class variable. Such a model specifies a more effective posterior probability distribution of the class variable given the subset of input variables that are relevant to the class prediction, which avoids

the negative effect of the redundant and irrelevant input variables to the target class. In this context we propose two variations of ACO-based algorithms for learning Markov blanket classifiers. The first algorithm is ABC-Miner+$_1$, which executes in two phases. The first phase is dedicated to construct a structure where only the relationship types (if any exists) between the class and the input variables are defined, while in the second phase the dependency relationships among the input variables, according to the previously discovered structure, are defined. The second algorithm, ABC-Miner+$_2$, utilizes an integrated approach, where the whole MB structure – including the dependency relationships between the class and the input variables, and among the input variables – is constructed in a synergic fashion.

The present paper is an extended version of the NICSO 2013 workshop paper [48], where ABC-Miner+$_1$, the two-phase ACO algorithm for learning MB classifiers, was introduced. We build on the work described in [48] in four ways. First, we introduce the novel ABC-Miner+$_2$ algorithm, which discovers the dependency structure of the MB classifier in a single integrated phase. Second, in order to mitigate possible training-phase overfitting, we propose two new ideas: 1) randomly changing the validation set during the training phase at each iteration; and 2) introducing a new penalty component in the model quality evaluation function according to the number of class variable parents to limit it and avoid producing overfitted structures. Third, we use two different types of classification measures to evaluate the quality of the candidate model constructed during the training phase: accuracy and probabilistic accuracy. Fourth, in terms of empirical evaluations, the number of datasets used in the experimental evaluation is increased from 18 to 33, and we compare our proposed algorithms with various well-known classification algorithms.

Note that we use the word "causal" in a loose sense in this work, simply to refer to a direction of the dependency relationship between two variables. The issue of whether or not Bayesian networks learned from observational data represent truly causal knowledge is controversial (depending on how we define causality) [41], and is out of the scope of this paper.

The rest of the paper is structured as follows. The next section gives some background on the two related areas of this research, namely BN classifiers and ACO. Then, we briefly review the ABC-Miner algorithm in Section 3, to make this paper more self-contained. In Section 4, we discuss the motivation behind our target task, which is learning Markov blanket classifiers, and the difference between directly aiming at that target and learning General Bayesian Networks (GBNs) and then extracting the class variable's Markov Blanket. We introduced our two ACO-based algorithms, ABC-Miner+$_1$ and ABC-Miner+$_2$, in Section 5, along with the quality evaluation functions used in our algorithms for building the MB classification models and the overfitting mitigation techniques. Our experimental methodology is presented in Section 6, followed by the computational results and their analysis in Section 7. Finally, we conclude with general remarks in Section 8.

2 Background

2.1 Bayesian Networks

Bayesian Networks (BNs) are a statistically sound method for representing probabilistic (in)dependencies among variables and using those (in)dependencies for probabilistic inferences [9]. In a BN, nodes represent variables (features or attributes) and edges represent dependencies among variables. The set of nodes and edges forms a DAG (Directed Acyclic Graph). Each node in that DAG is also associated with a Conditional Probability Table (CPT), which specifies the probability for each value of its variable given the values of all the variables that are parents of that node. The set of CPTs are the set of parameters Θ of the BN. The joint probability distribution of the set of variables $\mathbf{X} = \{X_1, X_2, X_3, ..., X_n\}$ in a BN given its DAG structure G and its set of parameters Θ is given by the following factorized formula:

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

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

Learning a BN from a dataset involves two steps: learning the DAG structure, and then learning the set of parameters Θ. The DAG structure learning step is usually considered the most difficult one, since parameter learning can be performed by estimating the relative frequency of each variable value directly from the dataset.

Broadly speaking, there are two approaches for learning the DAG structure of a BN. The first one is often called the *CI-based* (Conditional Independence-based, or constraint-based) approach [23, 9]. This approach is based on iteratively using some kind of statistical independence test (e.g. the Chi-squared test) to detect whether a certain set of variables is (in)dependent – possibly given other variables. Two issues with this approach are that the results of such statistical tests are quite sensitive to the value of a (ad-hoc) user-specified significance level value and the iterative use of such tests leads to the well-known problem of multiple statistical hypothesis testing.

By contrast, the second approach, usually called the scoring-based approach, uses a scoring function to evaluate each DAG structure G with respect to the dataset $\mathbf{D}$ at hand [23, 9]. The basic idea is to find the DAG structure G that best fits the dataset $\mathbf{D}$ in terms of $P(\mathbf{D}|G)$. This idea is implemented by using a search method (usually a greedy one) G that maximizes the value of a given scoring function. This approach has been more popular in the data mining and machine learning fields, possibly because it views the problem of BN learning as a well-defined optimization task (avoiding difficult issues of multiple hypothesis testing), where various types of search methods can be employed [4]. K2, MDL, KL, BDEu and several scoring functions can be used for this scoring-maximization task [7, 24, 55].

For a more detailed review on learning BNs, we recommend the very comprehensive review by Daly et al. [9] as well as [24,23].

2.2 Bayesian Network Classifiers

A general-purpose BN can compute the probability of any set of variables' values given any other set of variables' values. By contrast, a BN classifier is built specifically for the classification task, i.e., to compute the probability of each value of the class variable for an instance given the values of all the other variables of that instance. More formally, a BN classifier computes the posterior probability of each value (label) l of the class variable C given an instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ using a DAG structure G and parameters Θ, then labels this case with the class value having the highest posterior probability, as show in the following formulas:

$$C(\mathbf{x}) = \underset{\forall\ l \in C}{\arg\max}\, P(C = l | \mathbf{x} = x_1, x_2, ..., x_n, BNC), \tag{2}$$

and according to the Bayes' theorem,

$$\overbrace{P(C = l | \mathbf{x} = x_1, x_2, ..., x_n)}^{\text{posterior probability}} \propto \overbrace{P(C = l)}^{\text{prior probability}} \prod_{i=1}^{n} \overbrace{P(x_i | \mathbf{Parents}(X_i))}^{\text{likelihood}}, \tag{3}$$

where $\propto$ denotes the proportionality relationship. The above formulas refer to a typical type of BN classifier, where the class variable is a parent (cause) node to all the input variables (predictor attributes). As will be seen later, this is not the case in all types of BN classifiers.

Among the many types of BN classifiers, the simplest one is Naïve-Bayes (NB) [15,18]. NB makes the assumption that the input variables are independent of each other given the class variable, so its network has no edges connecting input variables. This reduces the posterior probability formula to:

$$P(C = l | \mathbf{x} = x_1, x_2, ..., x_n) \propto P(C = l) \prod_{i=1}^{n} P(x_i | l) \tag{4}$$

Since the assumption of independence among input variables is usually not realistic, many extensions of NB have been proposed. As discussed in [28,56], those extensions can be broadly divided into three main approaches: 1) applying NB to a subset of the input variables [31,34,27]; 2) extending the network structure of NB [3,4,19]; and 3) building local models based on different subsets of the dataset [25,21,33,4,30,50]. The current work is related to the first two types of extensions.

The first approach, called *feature selection* [31,35], consists of selecting a subset of relevant variables (features) from the dataset for use in model construction. This is in contrast to conventional NB, where all the input variables have some effect on computing the posterior probability of the class, and so

redundant, strongly-correlated, and irrelevant variables may degrade the predictive performance of NB. Several feature selection methods were introduced in the literature, such as Backward Sequential Elimination (BSE) method [31], Forward Sequential Selection (FSS) [34], and Evolutional Naïve-Bayes (ENB) [27].

The second approach to improve NB is extending its structure to represent the dependency relationships between the input variables given the class variable, adding edges edges between the input nodes in the network. In practice, imposing restrictions on the number of parents that an input node can have in the BN classifier is important for at least two reasons. First, finding the optimal structure of a BN (classifier) is $\mathcal{NP}$-hard [5,6], thus restricting the number of dependencies would reduce the search space and make the problem computationally tractable. Second, including too many dependency relationships in the BN classifier may result in producing a complex classification model that is prone to overfitting the training set and does not generalize well on the test set. Besides, BN models with a large number of edges tend to be less comprehensible and harder to be interpreted by the user. Figure 1 illustrates the various structural types of BN classifiers.

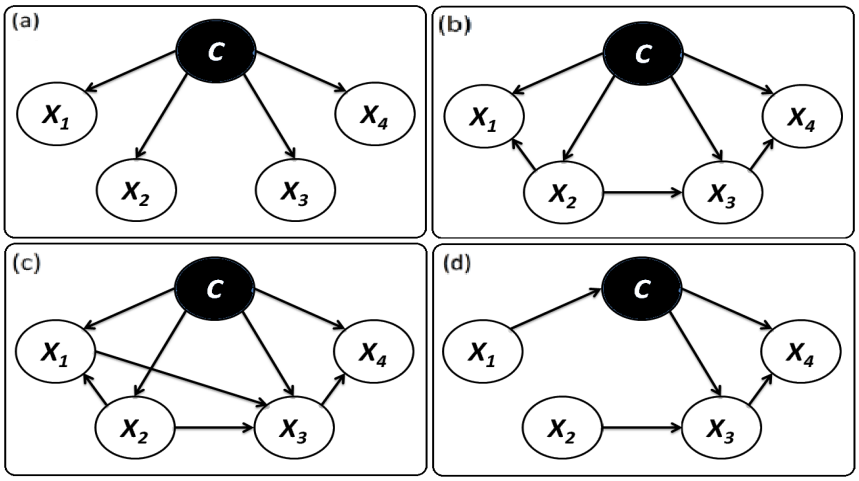

Fig. 1 Different types of Bayesian classifiers are presented: (a) Naïve-Bayes, where all the input variables have only the class variable as a parent. (b) TAN, where a variable can have one parent besides the class variable. (c) BAN, where a variable can have multiple parents beside the class variable. (d) MBC, where the class variable can have both parent and child variables.

A simple extension of NB is the Tree Augmented Naïve-Bayes (TAN), which allows a node in a BN to have one parent, in addition to the class variable [18]. This produces a BN with a tree-like structure. The Chow-Liu tree (CL-Tree) [3,4], and the SuperParent TAN (SP-TAN) algorithms are examples of well-known TAN classifier learning algorithms. A more elaborated extension

of NB is the Bayesian Network Augmented Naïve-Bayes (BAN). 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. Another variation of the Chow-Liu algorithm can be utilized to build BANs as well [3,4]. Note that if $k = 1$ a BAN becomes a TAN. The original version of our ABC-Miner algorithm produces BN classifiers with a BAN structure [49]. Since, the Markov blanket classifier (MBC), Figure 1 (d), is the focus of the current work; it is discussed separately in Section 4.

2.3 Ant Colony Optimization

Ant colony optimization (ACO) is a meta-heuristic inspired by the behavior of natural ant colonies [14,13,12]. Although each individual ant has a simple behavior, the ants in a colony cooperate with each other to solve complex optimization problems, resulting in an emergent intelligent behavior at the level of the colony. The pseudo-code of a typical ACO algorithm is shown, at a high level of abstraction, in Algorithm 1.

Algorithm 1 Pseudo-code of basic ACO algorithm.

```
Begin ACO
ConstructionGraph ← Problem_definition;
Initialize();
best ← φ; /* best solution found so far */
repeat
    current ← ant.ConstructSolution()
    ApplyLocalSearch(current)
    if Quality(current) > Quality(best) then
        best ← current;
    end if
    ant.UpdatePheromone(current);
until termination_condition
return best;
End
```

The first step of Algorithm 1 is the definition of the construction graph, whose nodes represent the components to be used to construct a candidate solution to the target problem. In the repeat-until loop, first each ant incrementally constructs a candidate solution by following a path in the graph, choosing which components are added to the current candidate solution in a heuristic manner (see below). Then, a local search procedure is applied to the just-constructed candidate solution, in order to try to improve it. Next, the quality of the current candidate solution is evaluated, and the quality of the best solution constructed so far by the algorithm is tracked and saved in the "best" variable. In the last step of the repeat-until loop, the current ant deposits pheromone on the construction graph components that were included in its constructed candidate solution.

Importantly, the amount of pheromone deposited on each graph component is proportional to the quality of the current candidate solution, which is measured by a predefined quality evaluation function. In future iterations, ants will be attracted by larger amounts of pheromone. Hence, the deposit of pheromone acts as a positive feedback mechanism, which encourages the ants to prefer solution components that were often used to produce good solutions in the previous iterations of the search. The algorithm also incorporates a pheromone evaporation strategy (not shown in the high-level pseudo-code of Algorithm 1), where the amount of pheromone in each solution component decreases gradually over time. This makes the search to put more emphasis on the quality of solutions constructed in recent iterations, rather than in early iterations, helping convergence to good solutions.

In order to design an ACO algorithm, one has to specify not only a construction graph (representing solution components) and a quality-evaluation function, but also the state transition formula used by each ant to decide which component should be added next to the current solution. A typical state transition formula consists of the product of two factors: the heuristic value η and the pheromone amount τ associated with each candidate solution component. The value of η is usually computed by a predefined "local" heuristic function that measures the quality of a solution component by itself, regardless of the quality of the entire solution using that component, and regardless of the previous history of the search. By contrast, the value of τ is given by the amount of pheromone accumulated on a solution component, taking into account the history of the search – i.e., solution components that were used to build better solution in the past accumulate more pheromone, as mentioned earlier. Typically, an ant chooses which component to add next to a candidate solution with a probability proportional to the product of the heuristic value η and the pheromone amount τ for that component. In addition, one also has to specify specific formulas for pheromone updating (based on the quality function) and pheromone evaporation, of course. All these design decisions will be specified in the context of our proposed ACO algorithm in later sections.

It is important to note that, unlike the conventional local, greedy search methods which are commonly used in search and optimization problems, an ACO algorithm performs a global search for near-optimal solutions in the search space. The global search stems from using a population of ants which is initially spread across different regions of the search space and which iteratively cooperate with each other (based on the positive feedback associated with depositing more pheromone on better solution components) in order to converge to a near-optimal solution.

ACO has been effectively used for learning *general-purpose* BNs [54, 10, 8, 42], as well as different types of classification models [40, 43, 37, 39, 44, 51]. Moreover, ABC-Miner, recently introduced by the authors [45, 49], is the first ACO algorithm for learning BN *classifiers* with a BAN structure, and it was shown to outperform several BN classification algorithms. Thus, we extend the ant-based ABC-Miner algorithm to learn the more advanced structure of MB classifiers. The authors have also introduced a clustering-based Bayesian multi-

net classification algorithm [47], which uses ACO to cluster the dataset into subsets, then builds several local BN classifiers. However, this paper addresses the problem of building BN classifiers with a very different approach.

3 An Overview of the ABC-Miner Algorithm

In ABC-Miner, the decision components in the construction graph (which are available for each ant to construct its candidate solution) are all the edges of the form $X \rightarrow Y$ where $X \neq Y$ and X, Y belong to the set of input variables. Each edge $X \rightarrow Y$ indicates that the value of variable Y depends (probabilistically) on the value of variable X. ABC-Miner selects edges from the construction graph in order to build a BAN classifier. Algorithm 2 outlines the ABC-Miner procedures.

Algorithm 2 Pseudo-code of ABC-Miner.

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

Each ant builds a candidate BAN classifier by executing the $CreateSolution()$ procedure in line 7 of Algorithm 2. That procedure is described in in Algorithm 3. Before starting to build its BAN classifier, each ant automatically selects the maximum number of parent variables (in addition to the class variable, which is a parent of all other variables) for each variable in the network [45, 49] (line 2).

In Algorithm 3, an ant starts with the network structure of Naïve-Bayes (line 3), where every variable has only the class variable as its parent. Then the ant expands that structure into a BAN structure by adding one edge at

Algorithm 3 Pseudo-code of solution creation procedure by an ant.

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

a time to the network (lines 4 to 8). The edge selected to be added in the current step is determined probabilistically. More precisely, the probability of each edge being selected is given by the product of the pheromone amount in the edge (reflecting the usefulness of that edge for constructing good BANs in previous iterations of the search) times the heuristic function value of that edge – measured by its conditional mutual information [45, 49] (line 5). However, an edge can be added to the current BAN structure only if its inclusion does not produce a directed cycle and does not violate the constraint on the maximum number of parents k (chosen by the current ant) for the node that the edge is pointing to. When an edge is added to the current BAN, all invalid edges are removed from the construction graph (line 7). An ant keeps adding edges to the current BAN as long as there are valid edges in the construction graph. Then, the CPT of each variable is computed, in order to create a complete BAN classifier (line 9). That classifier is then evaluated, and all edges become available again for constructing other BAN classifiers (line 8 in Algorithm 2).

ABC-Miner evaluates the quality of a candidate BAN classifier using a measure of predictive accuracy [45, 49], since the BAN will be used only for predicting the value of a specific class attribute. This is in contrast to general-purpose BN learning algorithms, whose solution-quality function does not distinguish between the input (predictor) and the class attributes. As shown in Algorithm 2, only the colony's iteration best solution BAN_{tbest} undergoes local search and is used for updating pheromone (lines 14 and 15). The best-so-far solution BAN_{bsf} is saved during the search and returned as the output of the algorithm.

4 Learning Markov Blanket Classifiers

4.1 Target and Motivation

The motivation behind our proposed extension is the following. As discussed in Section 2.2, two different limitations can be concluded from conventional TAN and BAN, which are the commonly used structures of the BN classifier that extends Naïve-Bayes. First, it assumes that the class variable has dependency

relationships with *all* the input variables (attributes), which means that the state of each input variable affects the posterior probability of the class values, and consequently the class prediction. This assumption is not necessarily valid in all applications domains. In some domains, some attributes are irrelevant, or at least not directly related, to the prediction of the target class. Including these irrelevant attributes in the computation of the posterior probability of the class values, according to Equation 3, can be disadvantageous, and may lead to incorrect predictions.

Second, the relationship between the class and all the input variables is always a type of "causal" relationship, that is, the class variable can only be a parent of an input variable. This is noticed in ABC-Miner's BAN creation procedure; it starts with a Naïve-Bayes structure where the class variable is fixed to be the parent of all the input variables (Algorithm 3, line 3). Such a property limits the flexibility of the algorithm to learn. Nonetheless, in real-world domains, some input variables are "causes" (parents) of the class variable, whereas others are "effects" (children) of the same class variable. For example, in a cancer diagnosis domain, the state of the *smoker* variable can be considered a cause of the state of the *Cancer* class variable, while the state of the *X-Ray* variable can be considered an effect of the class variable.

Our proposed ACO algorithms learn Markov blanket classifiers, which have the most flexible and elaborated BN structure, as shown Figure 1(d). The process of learning the class variable's MB structure performs an embedded feature selection with respect to the target class variable, by including only the input variables that contain the relevant information with respect to the target class prediction. It is not necessary to have a dependency relationship between the class variable and each of the input variables. This means that an input variable may not have a direct connection (edge) to the class node in the network, or an input variable may not even be presented in the network. In addition, the algorithm allows (up to) k dependency relationships to be defined for an input variable in the MB classifier, to relax the (unrealistic) independency assumption of Naïve-Bayes.

Moreover, our proposed algorithms allow the type of dependency (edge) between the class and the input variables to vary from "causal" to "effect" relationships in the MB classifier, where the class variable can be a parent or a child of an input node, unlike the BAN structures produced by ABC-Miner. The advantage of allowing this kind of edges in the BN model is the possibility of capturing new conditional (in)dependency relationships. For example, if X and Y are input variables that are unconditionally independent of the class variable C, then X and Y should be parents to C. This kind of (in)dependency relationship cannot be modeled by a BAN structure. Such a flexible MB classifier structure should better represent the class posterior probability distribution according to the dependency relationships, and lead to higher classification accuracy.

Formally, given the set of variables $\mathbf{X}$ and target class variable C, a Markov blanket for C is the smallest subset $\mathbf{S}$ of the input variables $\mathbf{X}$, such that C is independent of $\mathbf{X} - \mathbf{S}$, conditional on the variables in $\mathbf{S}$. Since the Markov

condition is assumed in a BN, every node X_i in $\mathbf{X}$ is independent of its non-descendants and non-parents in the network, conditional on its parents. Therefore, the variable subset $\mathbf{S}$, which composes the Markov blanket of the class variable C (MBC), is the union of C's parents, C's children, and the parents of C's children (Figure 1(d)). These are basically the variables on which the class variable is dependent — i.e., the information about the nodes in the Markov blanket of the class variable affects the posterior probability calculation of the class [32], as follows:

$$P(C = l|\mathbf{x}) \propto P(C = l|\mathbf{Parents}(C)) \prod_{v=1}^{|\mathbf{M}|} P(x_v|\mathbf{Parents}(X_v), MBC), \quad (5)$$

where $X_v \in \mathbf{M}$, where $\mathbf{M}$ the subset of the input variables that have the class as parent, and $\mathbf{Parents}(C) \cup \mathbf{M} = \mathbf{S}$.

4.2 Learning MB Classifiers vs. GBNs

It is worth mentioning that a straightforward known approach to discover the Markov blanket of a class variable is the use of General Bayesian Networks (GBNs) [3,4]. In essence, any BN learning algorithm can be used to construct a *general-purpose* BN. Then one can find the Markov blanket of the class node, delete all the other nodes outside that blanket and use the resulting network structure as a Bayesian classifier. However, such an approach is very different from the approach employed in this work, which focuses only on directly constructing an MB classifier, on a number of aspects. First, the algorithms utilized for learning GBNs do not treat the class variable as a special node in the network, i.e., the algorithm may add dependency-relationships (edges) that are irrelevant to the class posterior probability calculation, because the target is constructing a general BN. On the other hand, the search space of the class variable's MB gets much smaller during the construction process. For example, if the edge $X \leftarrow C$ is added between the input variable X and the target class variable C in the MB classifier being constructed, then an edge from any input variable X to Y becomes invalid (irrelevant) to be added in the network, which is unlike GBN algorithms. This is described in more details in Section 6.

Moreover, the GBN algorithms are oriented toward estimating any marginal probability distribution. Unlike our proposed algorithms, which are specifically focused on the task of estimating the conditional probabilities of the class attribute given the set of input variable values in constructing the MB classifier. This implies that a GBN does not perform well in the classification task compared to a MB classifier, and other types of BN classifiers, that are constructed with the classification purpose in mind. Such a purpose (building a BN classifier rather than a general BN) has an implication on different design aspects.

For instance, ACO-B [10], an ACO algorithm for learning GBNs, used the K2 scoring function as the heuristic information and solution quality evaluation. The aim is to increase the general inference capabilities of the constructed

BN, with no respect to a specific target (class) variable. On the other hand, an ACO algorithm for learning BN classifiers (such as ABC-Miner and the two algorithms proposed in the current paper) would use a classification-based evaluation function, which measures the quality of the constructed model directly as a classifier, with respect to predicting the class value. Section 5.6 discusses the quality evaluation functions used in our work.

5 The Proposed ACO-based Algorithms

5.1 Construction Graphs

The aim of our proposed ACO algorithms is to discover the variable dependency structure of the Markov blanket, given a training set, with respect to the target class variable. At first glance, one would suggest that the decision (solution) components in the ACO construction graph, by which the ants would construct the candidate solution (MBC structure), are the possible dependency relationships between the variables of the domain. However, with a closer look at the problem, we can define two different types of these relationships; the relationships between the class and the input variables, and the relationships among the input variables. The reason we distinguished between the two types of the relationships is that we think that the latter type of relationship should be discovered based on a complete definition of the former type. Hence, for constructing a candidate MB classifier structure, the class-input variables dependency relationships should be completely defined before the input variable-variable dependency relationships are discovered. Otherwise, a part the search process would be wasted by adding irrelevant decision components to the solution being constructed.

For example, let us assume that both types of dependency relationships are available in the search space (construction graph). An ant would select the edge $X \leftarrow Y$ to define a dependency relationship between the two input variables X and Y in the MBC structure. However, the search process would finish constructing the MBC structure without adding the edge $C \leftarrow Y$ between the class variable C and the input variable Y, which is necessary to make the previously added dependency relationship $X \leftarrow Y$ relevant to calculating the posterior probability of the class variable (see Section 4.1). Therefore, adding edges between input variables before defining all the edges between the class and the input variable would introduce waste of time in the search process.

According to the previous reasoning, we define two different construction graphs, one for each type of dependency relationships. The ACO algorithm uses the first construction graph to completely define the relationships between the class and the input variables, then it uses the second construction graph to define the relationships among the input variables based on the previously defined structure. A graphical representation of the two construction graphs is illustrated in Figure 2.

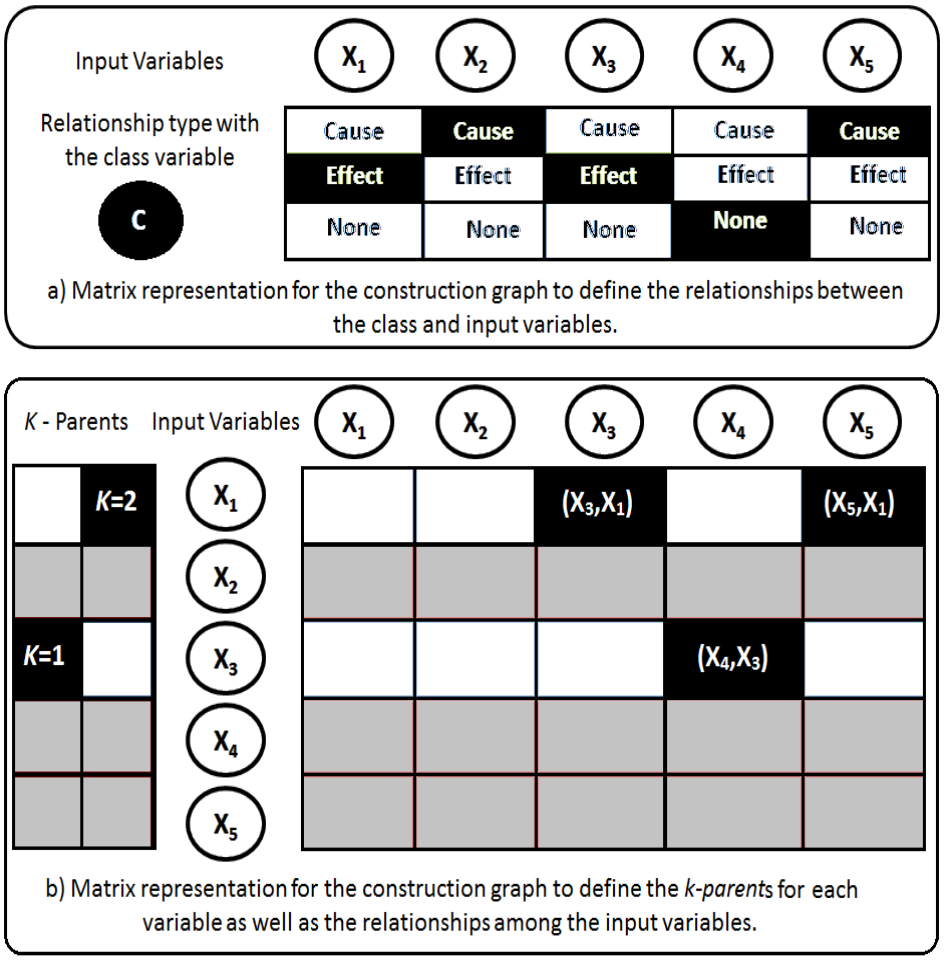

Fig. 2 Matrix representations for the construction graphs to define the two types of dependency relationships: a) between the class and the input variables; and b) among the input variables, where the variables on the columns are the parents. Invalid components are shown in light grey.

Figure 2(a) is a matrix representation of the construction graph for defining the dependency relationships between the class and the input variables. In essence, for each input variable X, an ant has to choose between three decision components, each represents a relationship type between X and the class variable C. If the "cause" relationship type is chosen, the edge $X \rightarrow C$ is added to the MBC structure, where the input variable X becomes a parent node to the class variable C. If the "effect" relationship type is chosen, the edge $C \rightarrow X$ is added to the MBC structure, where the input variable X becomes a child node to the class variable C. However, if the decision component representing the "none" relationship type is selected, there should not

be an edge (direct dependency relationship) between the class and the input variables. The black-highlighted decision components in Figure 2(a) represent an example of the relationship types selected. Figure 3(a) shows the output structure of this selection.

The dependency relationships among the input variables can be defined using the construction graph shown in Figure 2(b). Similar to ABC-Miner, the left side of the construction graph represents the allowed number of parents that a variable can have in the network structure. The idea is that instead of having the user selecting the optimum number of the (at most) k parents for each node, this selection is carried out by the ACO algorithm in a self-adaptive manner [49]. The selection of k_i value, for the input variable X_i, is done probabilistically from a list of available numbers. The user only specifies the `max_parents` parameter, and all the integer values from 1 to this parameter are available for the ant to use in the structure construction. More precisely, if the number of variables is n, the ant would have n values for the k parents limit, one for each variable, where variable i is restricted to have (at most) k_i parents during the network construction. Later, the ant updates the pheromone on the value k_i $(i = 1, 2, ..., n)$ after solution creation according to the quality of this MB classifier, which used value k_i for variable i in the process of solution construction. This pheromone amount influences the selection probability of this value by subsequent ants, leading to convergence on a near-optimal value of k_i dependencies for each variable i.

The right side of the construction graph in Figure 2(b) includes the decision components representing the valid dependency relationships that can occur between the input variables, where the variables on the columns are parents. In other words, the search space contains all the edges $X \rightarrow Y$ where $X \neq Y$ and X, Y belongs to the input variables. The validity of an edge in this construction graph depends on the primary structure that defines the edges between the class and the input variables constructed using the first construction graph. For example, according to Figure 2(a), since X_2 and X_5 are parents to the class variable C, then no edge can be added from any input variable to X_2 and X_5. The invalid components Figure 2(b) are shown in light gray, based on the selected components in Figure 2(a). On the other hand, an example of a selected set of edges among the input variables in 2(b) is in black, and the complete output structure of a MB classifier based on all the component selections in Figure 2 is shown in Figure 3(b).

5.2 The Two-Phase ABC-Miner+$_1$

The first proposed ACO algorithm for learning MB classifiers is ABC-Miner+$_1$, which executes in two sequential phases, as shown in Algorithm 4. In the first phase (lines 4 from to 20), it finds the dependency relationship types between the class variable and each of the input variables. Then, in the second phase (line 21), it finds the dependency relationships among the input variables, based on the structure discovered in the previous phase. Note that each phase

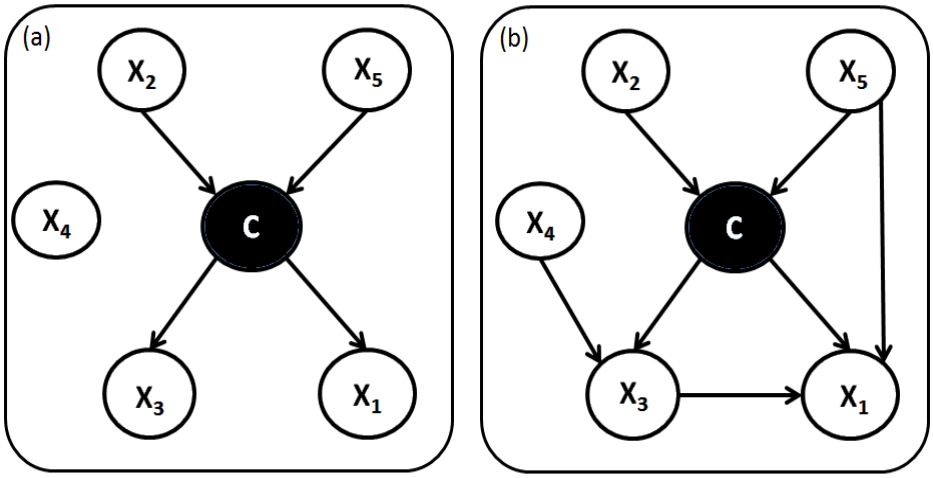

Fig. 3 Example of output structures defining the two types of dependency relationships: a) between the class and the input variables; and b) among the input variables.

is considered a completely separate ACO procedure, where the output of phase one is fixed during the execution of phase two. In the first phase, the product is a primary BN structure STR that contains only the edges between the input variables and the class variable, if any exists, and does not contain edges between the input variables.

Algorithm 4 Pseudo-code of ABC-Miner+$_1$.

Begin
$MBC_{final} = \phi$; $STR_{bsf} = \phi$;
$Initialize()$; $t = 1$;
repeat
  $sets = trainingSet.Split()$; /* split into learning and validation sets */
  $learningSet = sets[0]$; $validationSet = sets[1]$;
  $STR_{tbest} = \phi$; /* an empty network structure */
  for $i = 1 \rightarrow$ `colony_size` **do**
    $STR_i = FindRelationshipTypes(ant_i)$; /* create a candidate solution */
    $LearnParameters(STR_i, learningSet)$;
    if $Quality(STR_i, validationSet) > Quality(STR_{tbest}, validationSet)$ **then**
      $STR_{tbest} = STR_i$;
    end if
  end for
  $UpdatePheromone(STR_{tbest})$;
  if $Quality(STR_{tbest}, validationSet) > Quality(STR_{bsf}, validationSet)$ **then**
    $STR_{bsf} = STR_{tbest}$;
  end if
  $t = t + 1$;
until $t =$ `max_iterations` **or** $Convergence($`conv_iterations`$)$;
$STR_{final} = PerformLocalSearch(STR_{bsf})$;
$MBC_{final} = ExecuteABCMiner(STR_{final})$; /* extend the final structure */
return MBC_{final};
End

In essence, each ant_i in the colony constructs a candidate primary structure STR_i, using the $FindRelationshipTypes()$ method (line 9). The construction is performed by probabilistically selecting a relationship type for each input variable with respect to the class variable from the construction graph (as shown in Figure 2 (a)) according to the pheromone amounts associated with the decision components. Then, the parameters are learnt for the structure using the learning set (subset of the training set) to produce a BN classifier. The predictive quality of the produced model is evaluated on the validation set (a subset of the training set with no intersection with the learning set) and compared with the iteration-best candidate solution STR_{tbest} (lines 11 to 13). The pheromone update is performed according to the quality of STR_{tbest}, in order to influence the component selection of the ants during the construction of the subsequent candidate solutions (line 15).

After that, the quality of the iteration-best solution STR_{tbest} is compared to the quality of the best-so-far solution STR_{bsf} (lines 16 to 18), in order to keep track of the best constructed primary structure. 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 (line 20). `conv_iterations`, `max_iterations` and `colony_size` are user-specified parameters. The best-so-far STR_{bsf} structure undergoes local search, and the optimized primary STR_{final} structure is produced to be used in the next phase. Note that the BN structure discovered in the first step contains no edges between the input variables, as shown in Figure 3(a). The quality evaluation, local search, pheromone update, and training set split procedures are discussed in the following subsections.

In the second phase, the best constructed and optimized STR_{final} structure of the BN classifier is extended to a complete Markov blanket of the class variable, by finding the dependency relationships among the input variables. To include this type of edge in the network structure, we execute the original ABC-Miner algorithm in this phase (line 22). However, in the context of ABC-Miner+$_1$, the solution creation procedure (Algorithm 3) of ABC-Miner starts with the STR_{final} structure constructed in the previous phase, rather than a Naïve-Bayes structure as in the ABC-Miner algorithm (Algorithm 3, line 3). Therefore, the output of the procedure is a MB classifier, rather than a BAN.

5.3 The Integrated ABC-Miner+$_2$

The second proposed ACO algorithm, ABC-Miner+$_2$, employs an integrated approach for building MB classifiers, where the relationships between the class and the input variables on one hand, and the relationships among the input variables on the other hand, are discovered in synergic fashion using a single phase, as shown in Algorithm 5.

Algorithm 5 Pseudo-code of ABC-Miner+$_2$.

```
1: Begin
2: MBC_bsf = φ;
3: Initialize(); t = 1;
4: repeat
5:     sets = trainingSet.RandomSplit(); /* split into learning and validation sets */
6:     learningSet = sets[0]; validationSet = sets[1];
7:     STR_tbest = φ; MBC_tbest = φ;
8:     for i = 1 → colony_size do
9:         STR_i = FindRelationshipTypes(ant_i); /* wrt the class variable */
10:        LearnParameters(STR_i, learningSet);
11:        if Quality(STR_i, validationSet) > Quality(STR_tbest, validationSet) then
12:            STR_tbest = STR_i;
13:        end if
14:    end for
15:    for i = 1 → colony_size do
16:        MBC_i = CreateSolution(ant_i, STR_tbest); /* edges among input variables */
17:        if Quality(MBC_i, validationSet) > Quality(MBC_tbest, validationSet) then
18:            MBC_tbest = MBC_i;
19:        end if
20:    end for
21:    MBC_tbest = PerformLocalSearch(MBC_tbest);
22:    UpdatePheromone(MBC_tbest);
23:    if Quality(MBC_tbest, validationSet) > Quality(MBC_bsf, validationSet) then
24:        MBC_bsf = MBC_tbest;
25:    end if
26:    t = t + 1;
27: until t = max_iterations or Convergence(conv_iterations);
28: return MBC_bsf;
29: End
```

Recall that the two-phase ABC-Miner+$_1$ algorithm finishes the construction of the primary BN structure (STR) that defines the edges between the class and the input nodes, and fix this structure during the process of the second phase, which extends this structure by finding the edges between the input nodes to complete the construction of a MB classifier (MBC). We can notice that the ABC-Miner+$_1$ has two separate repeat-until loops representing the search process of the ACO algorithms: 1) the one that produces the primary structure STR (lines from 4 to 20 in Algorithm 4); and 2) the one that extends the primary structure to a complete MBC (in the execution of ABC-Miner, line 22 in Algorithm 4). By contrast, the ABC-Miner+$_2$ algorithm consists of one integrated repeat-until loop (lines from 4 to 27 in Algorithm 5), and in each iteration of that loop a complete candidate MB classifier is constructed.

In each iteration of ABC-Miner+$_2$, shown in Algorithm 5, the integrated construction of a candidate MBC is carried out in two steps, as follows. First, the ant colony responsible for creating candidate primary structures (using the construction graph shown in Figure 2(a)) constructs several candidate STR solutions, using the $FindRelationshipTypes()$ method, and selects the iteration-best STR_{tbest} to go to the second step (lines 8 to 14). Then, the ant colony responsible for extending a primary structure STR to a complete MB classifier MBC (using the construction graph shown in Figure 2(b)),

extends the best discovered structure STR_{tbest} in the previous step, using the $CreateSolution()$ method, to construct several candidate complete MBC solutions (lines 15 to 20).

The key integration aspect in this approach is that the pheromone update is not performed on the two construction graphs until the complete MBC solution is constructed, local searched and evaluated. More precisely, in the first phase of the ABC-Miner+$_1$ algorithm, the pheromone feedback of the ants is performed in each iteration according to the quality of the primary structure (line 15 in Algorithm 4), rather than a complete MB classifier. That is, the first phase optimizes the primary structure, which defines the dependency relationships between the class and the input variables, independently of the possible dependency relationships that can be defined among the input variables. On the other hand, in the integrated approach of ABC-Miner+$_2$, the pheromone feedback is performed according to the quality (tested on a validation set) of the complete MB classifier (line 22 in Algorithm 5). The pheromone is updated on the decision components in the two construction graphs, introducing a relationship between the quality of the input variable-input variable dependency selection and the input variable-class dependency selections. Therefore, the optimization of the primary structure is also dependent on (or a part of) the optimization of the whole MB classifier structure.

5.4 A Note Regarding the Execution Time

The execution of the procedure shown in Algorithm 3, which adds the edges between the input variables, is more efficient (faster) in the context of our two proposed ACO algorithms (line 22 in Algorithm 4 and line 16 in Algorithm 5) than in the original ABC-Miner algorithm, which can be explained as follows. First, the search space of this procedure is smaller in the context of Algorithms 4 and 5 than the search space in context of the original ABC-Miner. The reason is that, in ABC-Miner, the initial structure is the Naïve-Bayes' structure, where all the input variables are children of the class variable, so all the candidate edges between the input variables are available for selection by an ant (i.e., any variable can be a parent to any other variable).

On the other hand, in the proposed algorithms, the initial structure has some input variables as parents of the class variable, and others are not even related to the class variable. In this case, the candidate edges available for selection to be added to the network are only the edges that satisfy two conditions, namely: the edge is connecting two input variables (rather than connecting an input variable to the class), and the edge is pointing to a child node of the class node. The algorithm does not consider adding edges between the class variable's parents because these edges do not affect the predictions (posterior probability calculation) of the MB classifier. For example, in Figure 2 (a), the valid edges to be added (white and black boxes) are the ones that are pointing to the variables X_1 and X_3, since they are the only child nodes to (effect of) the class variable, according to the previously constructed example

structure, as shown in Figure 3. Hence, line 22 in Algorithm 4 and line 16 in Algorithm 5 call a somewhat modified version of Algorithm 3, where the only modified lines are the BAN initialization in line 3 and the implementation of the $GetValidEdges()$ procedure.

Second, in the MB models produced by our proposed ACO algorithms, the size of the CPT for the variables that do not have the class variable as parent is relatively smaller compared to the CPT of the BN classifiers produced by ABC-Miner, where the class node has to be a parent to all the variables, besides their other parents. Smaller CPT size leads to less computational time.

Note that both ABC-Miner+$_1$ and ABC-Miner+$_2$ perform the same (maximum) number of solution quality evaluations, which represents the computational budget of each algorithm, and is equal to `max_iterations` *times* `colony_size` *times* 2. In the sequential two-phase ABC-Miner+$_1$ (Algorithm 4), the first half of the budget is utilized in the first phase (from line 4 to line 20), while the second half of the budget is utilized in the second phase (line 22). In ABC-Miner+$_2$ (Algorithm 5), the whole computational budget is utilized in one, integrated phase (from line 4 to 27).

5.5 Local Search

In the context of the two phase ABC-Miner+$_1$ algorithm, local search is performed in two different positions of the algorithm. The first position is at the end of the first phase, where the local search is performed once on the best-so-far primary structure STR_{bsf} (line 21 in Algorithm 4), to produce the optimized STR_{final} that goes to phase two. In this procedure, the algorithm tries a different relationship type for each variable with respect to the class variable. For example, if the relationship type between the input variable X_i and the class variable C in the STR_{bsf} is "cause", the algorithm temporarily changes it: one time to "effect", and another time to "none". Then we calculate the classification accuracy of each temporary BN structure (using a validation set), and select the one with the highest quality. We perform the same operations iteratively to each input variable $X_i \in \mathbf{X}$. The best locally optimized structure STR_{final} goes to the second phase.

In the second phase, the conventional local search procedure of ABC-Miner[49], embedded in the execution of line 22 of Algorithm 4, is performed after each iteration. The procedure tentatively removes one edge between two input variables at a time from the constructed MBC in a reverse order (removing last the edge that was added to the network first). If that removal improves the quality of the MB classifier, this edge is removed permanently from the network; otherwise it is added once again. This process continues until all the edges between the input variables are tested to be removed from the MB classifier, and then the MBC_{final} with the highest quality is returned. Note that the local search of the second phase does not affect the primary structure (the edges between the class and the input variables) discovered and optimized in the first phase.

In the integrated ABC-Miner+$_2$, the local search procedure is performed in one position in the algorithm (line 21 in Algorithm 5). In each iteration t, the iteration-best constructed MB classifier MBC_{tbest} undergoes the local search as follows. First, the edges between the input variables are temporarily removed, one edge at a time, in a reversed order, where the edges whose removal improves the quality of the MBC_{tbest} (tested on a validation set) are removed permanently from the structure. Next, the edges between the class and the input variables are tested to be removed, that is, for each variable that has a dependency relationship to the class variable other that "none", we temporarily change it to "none". This is performed for one variable at a time for each valid variable, where the change is kept if it improves the quality of the MB classifier.

We can notice that the second part of this local search procedure (involving the edges between the class and the input variables) is simpler than its corresponding procedure in ABC-Miner+$_1$ that optimizes the primary structure STR_{bsf}, since only the "none" relationship type is tried for only a subset of variables (which have edges with the class), rather than trying the other two relationship types for each variable. This is intended for two reasons. The first reason is to reduce the computational time, since this procedure is called in each iteration in ABC-Miner+$_2$, (line 21 in Algorithm 5), unlike ABC-Miner+$_1$, which calls the local search procedure only once on the best-so-far constructed primary structure (line 20 in Algorithm 4).

The second reason is that changing the relationship type from "cause" to "effect" or vice versa, after adding a dependency relationship among the input variables, may introduce irrelevant edges in the MBC structure. For example, suppose that X and Y are input variables, and there is a dependency relationship between them: $X \rightarrow Y$. Suppose that the relationship type between Y and the class variable C is "effect", i.e., the edge is $C \rightarrow Y$. Now suppose we change the relationship type from "effect" to "cause", i.e., the edge becomes $C \leftarrow Y$. In this case, the dependency relationship $X \rightarrow Y$ between X and Y becomes irrelevant to the class variable prediction and outside the scope of the class variable's Markov blanket.

5.6 Solution Quality Evaluation

As discussed in Section 4.2, GBN algorithms do not recognize any special target class variable. They treat all the variables of the dataset 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 is only concerned about the queries regarding the class variable, and so the BN classification algorithm should be designed to build a model that maximizes its effectiveness concerning the prediction of the target class variable.

Taking into account the aforementioned argument, ABC-Miner evaluates the quality of a candidate BN during the training phase directly as a classifier,

using the *accuracy* measure as the indicator of the predictive performance of that classifier. Accuracy is a simple and yet popular predictive performance measure, computed as:

$$Accuracy = \frac{|Correct|}{|VSet|}, \tag{6}$$

where $Correct$ is the set of the correctly classified instances, and $VSet$ is the current validation set.

However, the conventional accuracy measure discards the fact that a BN classifier does not only predict a class label of any given instance, it also associates a probability to this predicted class label. This probability represents the confidence of the BN classifier regarding its prediction.

More precisely, suppose we have two candidate BN classifiers BNC_1 and BNC_2, which are used to classify instance X^l, where l is the correct class label of the instance. Now let us assume that the posterior probabilities of the class C given instance X and each BN classifier are $(C = l|X^l, BNC_1)$=0.9 and $(C = l|X^l, BNC_2)$=0.55. If we used accuracy as the predictive performance measure, instance X^l will be counted as 1 correctly classified instance for both BN classifiers, and the performance of the two classifiers would be equal (with respect to the classification of instance X). However, this discards the fact that BNC_1 produced a higher posterior probability of the correct class label than the one produced by BNC_2. Therefore, during the training phase, BNC_1 should be considered to have a higher quality than BNC_2, and consequently should receive a better pheromone feedback.

We have addressed such an issue in our ACO algorithms for learning MB classifiers, where we used *probabilistic accuracy* to evaluate the quality of the candidate constructed MBC solutions and perform pheromone update. As shown in Equation 7, the probabilistic accuracy will count each of the classified instances in the validation set according to the posterior probability of its correct class label produced by the candidate MB classifier.

$$Probabilistic\ Accuracy = \frac{\sum_{j}^{|VSet|} P(C = l|X_j^l)}{|VSet|}, \tag{7}$$

where X_j^l is the j-th instance in the validation set, and l is the correct class label of instance X_j. Note that the probabilistic accuracy (Equation 7) is only the first component of the quality measure used to evaluate a candidate MB classifier. The second component, which concerns mitigating overfitting, is described in the next subsection.

As for pheromone update, we use the same procedure employed by the ABC-Miner algorithm [49]. Concerning pheromone deposit, the pheromone amount is increased on each decision component according to the quality of two constructed solutions: the iteration-best $tbest$ and the best-so-far bsf using a weighted reinforcement strategy. The strategy aims at focusing on the exploration aspect during the early iterations in the algorithms by giving more

weight to the *tbest* solution in depositing pheromone. Gradually, the procedure moves to exploitation as the search progresses by giving more weight to the bsf in depositing pheromone, leading to convergence towards a good solution. Then, the pheromone amounts are normalized to simulate pheromone evaporation [49].

5.7 Overfitting Mitigation

A key objective of a classification algorithm is to learn models with good generalization capabilities, i.e., models that are able to accurately predict the class labels of previously unknown instances. In fact, the available instances in the dataset used for building and validating a classification model are considered just a sample of a larger population of possible instances in the application domain of concern. The aim is to build a model that can generalize over the current dataset to correctly fit the data of the whole application domain as much as possible [22,52].

Overfitting occurs when the induced model (classifier) reflects good classification performance (fit) on the training (in-sample) data used for in the learning process, yet shows bad predictive performance (generalization) involving new/testing data. This is in general a property of complex classifiers that have a high degree of freedom — i.e., a BN classifier with a large number of dependency relationships (edges). In general, overfitting can occur due to two factors. First, if the classification algorithm used the whole training dataset for both learning (building the model) and validating the model's accuracy. In this case, the generalization ability would not be tested during the training phase, since the model would be validated on the same instances used to learn that model. The quality of the induced classifier might be due to a high (undesirable) fit on the training set, and the same classifier might show a bad predictive accuracy on the unseen test (out-of-sample) dataset. Second, allowing the classification algorithm to build too complex models might lead it to learn an over-tuned model that "memorizes" the training data, which harms its generalization ability. Learning, in the sense of using a set of instances to build a classifier that can predict the class of unseen instances, needs induction, which refers to discovering a general hypothesis (classification model) that can describe both the sample instances at hand and the whole population of the instances in the application domain as well [38,52].

In the context of our proposed algorithms, we tried to mitigate the overfitting problem, to produced well-generalized MB classification models, as follows. First, in order not to allow building complex BN structures, we limited the maximum number of parents that each node can have in the network to 3 (as applied in ABC-Miner [45,49]). However, learning MBC models is more prone to overfitting than learning BANs, since in the class variable MB structure, the class can have input variables as parents. In an extreme case, if the class variable would have *all* the input variables as its parents, the produced model would be perfectly memorizing the training data in the parameters of

the constructed BMC. More precisely, one big CPT would be built for the class variable that stores the probability of each class label given each combination of the input variables, which is an undesirable behavior.

We have noticed that, the more parent nodes the class variable has, the more it is prone to overfitting. Therefore, we introduced a penalty component to the quality evaluation formula, which affects the pheromone update procedure, based on the number of parents of the class variables, as follows:

$$Quality(MBC) = ProbabAcc(MBC, validation\_set) - Penalty(MBC), \quad (8)$$

$$Penalty(MBC) = \frac{1}{2^{(n-p)}}, \quad (9)$$

where $ProbabAcc()$ refers to the probabilistic accuracy measure that is shown in Equation 7, p is the number of parent nodes to the class variables, and n is the total number of the input variables. Note that Equation 8 represents the complete formula for evaluating a candidate MBC solution quality to perform pheromone update.

As shown in Equation 9, the penalty value can range from 0 to 1 (the same range as the probabilistic accuracy), according to the number of parents that the class variable has. If all the input variables are parents to the class node (i.e., $p = n$) – which is totally undesirable, the penalty value would be the maximum (equals 1), and the total quality value of the model according to Equation 8 would be less than or equal to 0. However, if the $p = n - 1$, the penalty would be reduced to its half. This means that, according to Equation 9, adding one more parent to the class variable would double the penalty value, as it would be increasing the possibility of the MBC model to overfit the training data.

Second, instead of having the algorithm build the MB classifier over the whole training set, the training set is split into two mutually exclusive parts: 1) the learning set, which contains 75% of the training set and is used to build a candidate MBC (i.e., BN parameter learning, as shown in Algorithm 4, line 10 and Algorithm 5, lines 10 and 16); and 2) the validation set, which contains 25% of the training set and is used to evaluate the quality of the constructed MBC for pheromone update. Such a strategy is common in many classification algorithms. However, these two parts are usually fixed during the whole training process, which may lead to optimize the model construction over the validation set. To avoid that the model overfits a fixed validation set, we propose randomly changing the partitioning of the learning/validation sets in each iteration, in order to push the generalization ability of the constructed models over any part of the training set. In our experiments, we utilize this idea in the two versions of ABC-Miner+ (as shown in Algorithm 4 and 5, lines 5 to 6), as well as in the original ABC-Miner algorithm.

6 Experimental Methodology

6.1 Comparative Evaluations

We compare the predictive accuracy of our proposed ACO algorithms for learning MB classifiers with our previously introduced ABC-Miner that learns BAN classifiers, as well as three other widely used Bayesian Network (BN) classifiers, namely Naïve-Bayes, Tree Augmented Naïve-Bayes (TAN) and General Bayesian Network (GBN). A variation of the Chow-Liu (CL) tree algorithm [3] is used for building TANs, as follows. First, it computes the conditional mutual information $I(X, Y|C)$ between each pair of variables X and Y given class variable C. Then it builds a complete undirected graph connecting all the input variables to find the maximum weighted spanning tree from the graph, where the weight of the edge $X \rightarrow Y$ is annotated with $I(X, Y|C)$. After that, 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 to complete a TAN classifier. As for the construction of GBNs, Algorithm-B [2] is used to build a general Bayesian network. The algorithm utilizes a greedy search to optimize the K2 scoring function for the Bayesian network. Then, the Markov blanket of the class node is extracted from the BN to be used as a MB classifier. We implemented in the experiments another version of the GBN learning algorithm, which optimizes the predictive accuracy, denoted as GHC-Acc. That is, it starts with an edge-less BN structure, and incrementally adds the edge which leads to the highest increase in classification accuracy on the validation set. Table 1 presents the main properties of the used BN classification algorithms.

Table 1 Summary of the BN classification algorithms used in the experiments.

Algorithm	Type	Search Strategy	Output	Optimization
Naïve-Bayes	Determ.	-	NB	-
CL-Tree	Determ.	Max. Spanning Tree	TAN	Cond. Mut. Info.
Algorithm-B	Determ.	Greedy Hill Clim.	GBN	K2 Function
GHC-Acc	Determ.	Greedy Hill Clim.	GBN	Predictive Acc.
ABC-Miner	Stoch.	Ant Colony Optim.	BAN	Predictive Acc.
ABC-Miner+$_1$	Stoch.	Ant Colony Optim.	MBC	Probabilistic Acc.
ABC-Miner+$_2$	Stoch.	Ant Colony Optim.	MBC	Probabilistic Acc.

Besides the BN classification algorithms, we compare our proposed ACO algorithms with three well-known classification algorithms: Ripper, C4.5, and SVM [53]. Ripper is a classification rule induction algorithm, which learns a classification model that consists of a list of classification rules, where each rule has the form: "IF (Antecedent) THEN (Class)". C4.5 builds classification decision trees, where the internal nodes are the input attribute values of the dataset, and the leaf nodes are the classes to be predicted. An SVM maps

the instances into a higher-dimensional feature space and then finds the best hyper-plane for separating instances of different classes – where the best hyper-plane is the one with the greatest possible margin (a gap separating instances of different classes in the data space). Note that SVMs have the disadvantage of producing black-box classification models that can hardly be interpreted by users [53,17,16], unlike the other types of classifiers used in our experiments.

6.2 Experimental Setup

The experiments were carried out using stratified 10-fold cross validation [53]. For the stochastic ACO-based algorithms, we run each algorithm 10 times – using a different random seed to initialize the search each time – for each cross-validation fold. In the case of the deterministic algorithms, each is run just once for each fold.

The parameter configuration used in our experiments is shown in Table 2. Note that the `max_iterations` parameter refers to the maximum number of iterations used in ABC-Miner and ABC-Miner+$_2$. However, in the case of the two-phase ABC-Miner+$_1$, each phase is allocated half of the total maximum number of iterations (i.e. 500 iterations in our experiments). On the other hand, for the greedy Algorithm-B and GHC-Acc algorithms, we refer to `max_iterations` as the maximum number of solution evaluations that the algorithm performs during the hill-climbing search to build a GBN. It is set to 1000, which is equal to the value of `max_iterations` multiplied by `colony_size` used for our stochastic ant-based algorithms. For the sake of fair comparison, we limit each algorithm to the same fixed number of solution evaluations to construct the model. However, the maximum number might not be utilized completely; ACO-based algorithms might only use a smaller number of iterations if they converged earlier and the greedy-based algorithms might also stop earlier if they get stuck in a local optimum.

Unlike our ant-based algorithms, the number of parents (k-dependencies) must be specified for Algorithm-B and GHC-Acc. We set it to 3, which is the maximum number of parents used in our ACO algorithm, where the number of parents is selected dynamically at each iteration (see Section 5.1). Note also that we have employed the two overfitting mitigation procedures (i.e., adding the penalty component in the quality evaluation function and randomly splitting the training set into a learning set and a validation set each iteration), discussed in Section 5.7, in both versions of the ABC-Miner+ algorithm. As for the other classification algorithms, in our experiments, we used WEKA [53] implementations for Rippper (JRip), C4.5 (J48), and SVM (SMO), each with its default parameter settings.

6.3 Evaluation Datasets

The performance of ABC-Miner+$_1$ and ABC-Miner+$_2$ was evaluated using 33 public-domain datasets from the University of California at Irvine UCI dataset

Table 2 Parameter settings of ABC-Miner+ used in experiments.

Parameter	Value
`max_iterations`	100
`colony_size`	10
`conv_iterations`	10
`max_parents`	3

repository [1]. The main characteristics of the datasets, such as number of instances, number of predictor attributes and number of classes are shown in Table 3.

Datasets having continuous attributes were discretized in a pre-processing step, using the well-known C4.5-Disc algorithm [53], applied only to the training set of each dataset (i.e., the cut-points of the intervals were computed by C4.5-Disc using the training data only, and the computed cut-points were used to discretize both the training and the test sets).

7 Computational Results

7.1 Predictive Performance

Table 4 reports the mean and the standard error ($mean \pm standard\ error$) of the predictive accuracy values obtained by 10-fold cross validation for the 33 datasets, where the highest accuracy for each dataset is shown in bold face. The last row shows the average rank of each algorithm in terms of predictive accuracy. 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.

As shown in Table 4, our proposed ACO-based algorithm for learning MB classifiers using the integrated approach, ABC-Miner+$_2$, obtained the best overall rank in terms of predictive accuracy of 2.7, and achieved the best predictive results in 10 datasets out of 33. SVM came in the second place by obtaining 3.2 as an overall rank, and achieved the best predictive results in 10 datasets as well. The two-phase ACO algorithm for learning MB classifiers, ABC-Miner+$_1$, came in the third place with overall rank of 3.5, and achieved the best predictive results in 6 datasets. In the fourth place came the original ABC-Miner algorithm, which learns BAN classifiers, with overall rank of 4.5, and achieved the best results in one dataset. C4.5 and Ripper came in fifth and the sixth place, respectively, by obtaining 4.8 and 5.9 overall rank, respectively. C4.5 achieved the best predictive results in 4 datasets, while Ripper obtained the best result in 6 dataset.

Table 5 shows the critical values' results of the statistical significance tests according to the non-parametric Friedman test with Holm's post-hoc test [11,

Table 3 Description of datasets used in the experiments.

Dataset	Cases	Attributes	Classes
abalone	4177	8	29
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
ecoli	336	8	8
glass	214	10	7
hayes-roth	160	4	3
heart (cleveland)	303	12	3
heart (statlog)	270	13	2
hepatitis	155	19	2
ionosphere	351	34	2
iris	150	4	3
lung cancer	32	56	3
monks	432	6	2
mushrooms	8,124	22	2
nursery	12,960	8	5
parkinsons	197	23	2
page Blocks classification	5,473	10	5
pima diabetes	768	8	2
post-operative patient	90	8	3
segmentation	2,310	19	7
soybean	307	35	19
SPECT heart	267	22	2
tic-tac-to	958	9	2
voting records	435	16	2
wine	178	13	3
yeast	1,484	8	10
zoo	101	17	7

20], which is used for comparing multiple algorithms on multiple datasets [26]. The results show the statistical comparison between the results of all the used algorithms with respect to the results of our two proposed algorithms: ABC-Miner+$_1$ and ABC-Miner+$_2$. We performed the Friedman test using the freely available Java program suggested by Garcia et al. in [20], which applies the test with two different statistical significance levels: $\alpha = 0.05$ and $\alpha = 0.1$. The values shown are the adjusted Holm p-values, where a double underlined valued indicates that the difference between the predictive results of

the two corresponding algorithms in the table is statistically significant at 5% level, and a single underlined value indicates that the difference is statistically significant at 10% level. A result shown with no underlines indicates that there is no statistically significant difference between predictive performances of the corresponding algorithms.

Table 4 Predictive accuracy % (*mean* ± *standard error*) results.

Dataset	Naïve-B	Cl-Tree	Algo-B	GHC-Acc	ABC-Miner	JRip	J48	SVM	ABC-Miner+$_1$	ABC-Miner+$_2$
abl	86.1 ± 2.5	87.2 ± 1.6	88.3 ± 1.3	88.7 ± 2.7	85.6 ± 2.7	**92.6 ± 2.5**	**92.6 ± 2.1**	89.1 ± 1.2	88.7 ± 1.5	89.3 ± 1.8
bal	**91.2 ± 0.8**	84.4 ± 0.9	85.6 ± 0.6	83.4 ± 0.9	84.7 ± 0.7	77.6 ± 1.8	80.2 ± 1.2	88.4 ± 0.9	85.6 ± 0.9	86.8 ± 0.7
bcw	92.1 ± 1.4	95.4 ± 1.2	93.8 ± 2.7	94.6 ± 1.4	95.4 ± 2.5	94.7 ± 1.5	95.1 ± 2.5	**97.7 ± 1.7**	95.4 ± 1.5	97.6 ± 2.7
car	85.3 ± 2.2	93.6 ± 1.2	86.2 ± 1.3	92.1 ± 2.1	97.2 ± 2.3	90.6 ± 1.2	94.3 ± 2.8	94.1 ± 2.5	98.1 ± 1.1	**98.6 ± 2.4**
chess	88.2 ± 1.5	92.5 ± 1.4	89.8 ± 2.2	95.1 ± 2.2	97.6 ± 2.5	98.1 ± 1.5	**98.4 ± 2.3**	95.1 ± 1.8	96.3 ± 2.7	97.6 ± 2.5
cmc	52.2 ± 2.1	56.4 ± 2.2	54.6 ± 2.8	60.4 ± 2.8	66.4 ± 2.7	60.1 ± 1.1	64.2 ± 2.4	60.2 ± 2.2	**67.8 ± 2.8**	65.3 ± 2.2
crd-a	77.5 ± 2.6	85.1 ± 1.8	85.7 ± 1.4	85.2 ± 2.4	86.8 ± 2.6	85.7 ± 2.5	**91.9 ± 1.2**	85.5 ± 1.1	84.2 ± 2.1	87.1 ± 2.6
crd-g	75.6 ± 1.6	73.7 ± 1.8	75.6 ± 1.2	74.8 ± 2.3	73.7 ± 2.8	73.2 ± 1.8	75.7 ± 1.6	75.2 ± 2.3	75.8 ± 2.7	**77.6 ± 1.1**
drm	96.2 ± 1.5	97.8 ± 1.1	97.2 ± 1.6	96.0 ± 2.6	98.6 ± 2.6	94.8 ± 2.7	96.1 ± 1.8	97.2 ± 2.1	**99.0 ± 1.3**	97.7 ± 2.8
ecoli	86.5 ± 2.2	84.2 ± 2.7	82.6 ± 2.6	84.1 ± 1.4	87.4 ± 1.2	84.3 ± 1.7	85.7 ± 2.8	87.6 ± 1.6	**90.4 ± 1.2**	**90.4 ± 2.4**
glass	74.2 ± 1.1	78.4 ± 0.9	80.7 ± 1.2	80.9 ± 2.3	82.6 ± 2.7	75.7 ± 1.4	82.2 ± 1.2	82.6 ± 1.2	85.3 ± 2.4	**85.8 ± 1.5**
hay	80.0 ± 1.4	77.9 ± 1.7	83.1 ± 2.7	**84.2 ± 2.1**	80.2 ± 2.3	**84.2 ± 2.1**	83.3 ± 2.3	81.6 ± 2.7	82.4 ± 2.1	**84.2 ± 1.4**
hrt-c	62.7 ± 2.1	68.8 ± 2.5	66.7 ± 1.7	76.8 ± 2.6	83.8 ± 2.1	67.1 ± 2.1	70.9 ± 2.3	72.6 ± 1.8	**84.2 ± 2.5**	81.8 ± 1.6
hrt-s	83.4 ± 1.7	83.3 ± 2.4	81.5 ± 1.8	72.4 ± 2.7	85.6 ± 2.4	76.6 ± 1.6	75.5 ± 1.8	84.1 ± 2.3	**88.4 ± 2.7**	86.2 ± 2.6
hep	71.9 ± 1.6	78.0 ± 1.6	67.3 ± 2.1	74.2 ± 1.6	77.8 ± 1.2	72.7 ± 1.4	68.9 ± 1.2	74.2 ± 1.6	**78.6 ± 0.9**	78.0 ± 1.6
iono	82.6 ± 0.6	90.5 ± 1.5	90.5 ± 1.5	92.8 ± 1.2	94.5 ± 1.7	92.8 ± 1.3	93.1 ± 2.7	**95.8 ± 1.7**	94.9 ± 1.3	94.9 ± 1.1
iris	96.2 ± 2.8	94.2 ± 1.2	92.9 ± 2.2	96.2 ± 2.8	96.2 ± 1.7	95.6 ± 1.8	95.0 ± 2.5	**96.6 ± 2.1**	96.2 ± 1.1	96.2 ± 2.5
lung	72.5 ± 2.6	75.4 ± 2.7	63.2 ± 1.8	74.8 ± 2.2	74.8 ± 2.8	58.7 ± 1.3	75.7 ± 1.5	**79.3 ± 2.2**	74.8 ± 1.2	74.3 ± 2.2
monk	61.6 ± 2.3	58.8 ± 2.4	61.6 ± 2.1	61.6 ± 1.1	61.8 ± 1.1	61.4 ± 1.2	62.2 ± 1.6	63.3 ± 1.3	62.2 ± 1.2	**64.6 ± 2.7**
mush	95.8 ± 1.7	98.2 ± 1.5	96.1 ± 2.8	98.0 ± 2.3	98.8 ± 2.4	**100 ± 2.3**	**100 ± 1.7**	**100 ± 2.4**	99.9 ± 2.3	99.9 ± 1.3
nurs	90.1 ± 1.7	94.3 ± 1.6	92.7 ± 2.7	93.1 ± 2.6	98.0 ± 1.4	96.5 ± 2.8	97.1 ± 1.8	93.2 ± 2.7	97.7 ± 1.2	**98.2 ± 2.5**
park	84.5 ± 2.2	91.7 ± 1.7	84.5 ± 1.2	88.2 ± 2.5	94.2 ± 2.1	98.5 ± 1.8	96.2 ± 2.4	98.2 ± 1.4	98.5 ± 2.4	**98.7 ± 2.8**
pbc	93.5 ± 1.7	96.1 ± 1.4	95.6 ± 1.1	94.2 ± 2.4	96.5 ± 2.1	97.4 ± 1.5	98.2 ± 2.6	**98.7 ± 2.3**	97.8 ± 2.2	96.5 ± 2.4
pima	75.4 ± 1.2	77.8 ± 1.5	76.2 ± 1.5	76.2 ± 1.5	78.9 ± 1.9	**79.9 ± 1.9**	79.5 ± 1.6	75.7 ± 1.2	78.9 ± 1.9	78.9 ± 1.9
pop	68.2 ± 2.3	64.1 ± 2.2	66.6 ± 2.1	71.1 ± 2.1	**74.3 ± 2.4**	71.1 ± 1.7	71.1 ± 1.7	72.2 ± 1.7	70.3 ± 1.7	72.5 ± 1.2
seg	91.6 ± 1.4	94.8 ± 2.6	94.1 ± 2.7	94.0 ± 1.2	94.5 ± 2.1	94.7 ± 2.4	96.3 ± 2.5	97.8 ± 2.7	95.8 ± 2.8	**971. ± 1.3**
soy	91.4 ± 1.7	95.6 ± 2.4	93.2 ± 2.2	92.1 ± 1.7	95.6 ± 1.1	91.5 ± 1.7	95.1 ± 2.3	**97.6 ± 1.4**	95.6 ± 1.3	95.6 ± 1.2
SPECT	73.7 ± 2.7	72.1 ± 1.4	74.0 ± 2.6	76.5 ± 2.6	76.5 ± 2.6	76.7 ± 2.2	74.4 ± 1.1	78.5 ± 1.5	78.8 ± 2.4	**79.6 ± 2.6**
ttt	70.3 ± 1.5	76.6 ± 2.5	74.3 ± 2.1	82.5 ± 2.5	86.4 ± 2.4	**97.8 ± 1.5**	85.7 ± 2.3	96.3 ± 2.3	86.8 ± 1.4	89.2 ± 2.3
vot	90.3 ± 1.6	92.1 ± 1.5	90.3 ± 1.8	91.4 ± 1.7	94.6 ± 1.5	**96.5 ± 2.1**	96.3 ± 2.6	95.8 ± 2.2	95.6 ± 2.7	95.6 ± 1.5
wine	95.6 ± 1.4	97.3 ± 1.3	97.3 ± 1.3	98.8 ± 2.2	98.4 ± 1.7	97.6 ± 1.1	95.8 ± 2.4	**98.9 ± 1.7**	98.0 ± 1.3	98.4 ± 1.1
yeast	59.7 ± 2.7	61.2 ± 1.8	60.2 ± 2.2	61.5 ± 1.8	62.6 ± 2.4	63.8 ± 1.5	64.2 ± 1.2	**64.6 ± 1.7**	64.2 ± 1.6	64.5 ± 1.7
zoo	94.2 ± 1.2	97.3 ± 1.3	95.1 ± 1.8	96.2 ± 1.2	97.0 ± 1.7	94.2 ± 2.5	96.1 ± 1.4	**97.5 ± 2.2**	97.0 ± 1.2	97.0 ± 2.1
Rank	8.6	6.8	8.0	6.7	4.5	5.9	4.8	3.2	3.5	2.7

Table 5 The non-parametric Friedman statistical test results (adjusted p-values) with Holm's post-hoc correction.

Algorithm	ABC-Miner+$_1$	ABC-Miner+$_2$
Naïve-Bayes	7.8E-1	1.9E-13
CL-Tree	3.3E-4	1.2E-6
Algorithm-B	3.4E-7	2.4E-10
GHC-Acc	0.0023	1.4E-5
ABC-Miner	1.0	0.0946
Ripper	0.0681	0.0012
C4.5	1.0	0.08012
SVM	1.0	1.0

7.2 Model Simplicity

Table 6 reports the model size results – in terms of the number of edges – of the models produced by the BN classification algorithms, as the average of the 10-fold cross validation experiments. The last row in the table shows the average rank of results for the model sizes over all the datasets, where the lower the rank, the better the algorithm. Note that we do not report the size results for Naïve-Bayes, since it is obvious that it produces the smallest model size; a Naïve-Bayes model has a number of edges that is equal to the number of input variables in the dataset.

As shown in Table 6, TAN, as expected, produces the smallest BN models in general, since the number of parents for each node in a TAN is restricted to one, besides the class parent. The reason why in some cases the ACO-based algorithm produces smaller models is due to the local search procedure that might remove edges. We can also see that, in most of the datasets, the MB classifier structures produced by our proposed ACO algorithms are smaller than the BAN models produced by ABC-Miner. More precisely, ABC-Miner+$_1$ and ABC-Miner+$_2$ obtained overall rankings of 2.3 and 3.1, respectively, while ABC-Miner obtained overall ranking of 4.3 (coming in the last place).

7.3 Execution Time

The execution time results are shown in Table 7. The running time (in seconds) of each algorithm in each dataset is reported. Besides, the ratio of the running time of each algorithm to the ABC-Miner algorithm (as a baseline) is shown in the column with "ratio" header in bold face. The last row in the table reports the average ratio of the running time of each algorithm to ABC-Miner across all the datasets. All the experiments were performed with an Intel dual-core i7

Table 6 Model size (*mean* ± *standard error*) results in terms of number of edges.

Dataset	CL-Tree	Algo-B	GHC-Acc	ABC-Miner	ABC-Miner+$_1$	ABC-Miner+$_2$
abl	8.0 ± 0.0	18.9 ± 2.2	12.9 ± 1.8	28.5 ± 2.4	21.7 ± 2.4	24.5 ± 2.7
bal	4.0 ± 0.0	3.7 ± 2.5	3.6 ± 3.2	3.4 ± 3.2	3.4 ± 2.1	4.1 ± 2.2
bcw	9.0 ± 0.0	18.7 ± 2.8	16.2 ± 2.4	23.6 ± 2.8	14.8 ± 1.4	20.4 ± 1.7
car	6.0 ± 0.0	13.6 ± 2.7	13.1 ± 1.2	15.5 ± 2.7	11.2 ± 2.1	13.3 ± 2.7
chess	36.0 ± 0.0	63.4 ± 2.3	65.7 ± 2.3	71.7 ± 1.7	61.8 ± 2.4	58.6 ± 2.8
cmc	9.0 ± 0.0	20.6 ± 2.2	12.6 ± 1.2	18.7 ± 1.7	10.4 ± 1.7	14.3 ± 1.5
crd-a	14.0 ± 0.0	17.4 ± 1.4	15.2 ± 2.7	23.6 ± 1.7	15.3 ± 1.6	16.6 ± 1.7
crd-g	20.0 ± 0.0	34.6 ± 1.2	30.4 ± 1.3	28.2 ± 2.5	18.5 ± 2.6	18.6 ± 1.4
drm	33.0 ± 0.0	22.5 ± 2.4	23.2 ± 2.7	34.6 ± 1.6	26.6 ± 2.1	33.8 ± 2.7
ecoli	8.0 ± 0.0	5.6 ± 1.2	9.8 ± 1.8	15.6 ± 2.6	17.2 ± 1.8	17.9 ± 1.8
glass	10.0 ± 0.0	8.2 ± 1.1	10.3 ± 1.5	15.5 ± 2.7	11.4 ± 2.8	10.2 ± 2.8
hay	4.0 ± 0.0	2.8 ± 2.4	3.1 ± 1.7	12.7 ± 2.6	7.3 ± 1.3	8.6 ± 1.7
hrt-c	12.0 ± 0.0	21.8 ± 1.6	20.4 ± 2.6	18.4 ± 1.5	21.3 ± 1.6	20.6 ± 1.1
hrt-s	13.0 ± 0.0	22.6 ± 2.4	19.6 ± 1.1	26.4 ± 1.8	26.9 ± 2.3	22.4 ± 2.6
hep	15.0 ± 0.0	24.8 ± 1.2	20.4 ± 1.2	26.9 ± 1.6	22.5 ± 2.4	24.6 ± 2.2
iono	34.0 ± 0.0	31.2 ± 1.4	36.7 ± 1.6	43.8 ± 1.8	41.6 ± 2.5	45.5 ± 1.3
iris	4.0 ± 0.0	3.2 ± 2.1	3.2 ± 2.4	4.9 ± 2.5	3.2 ± 1.8	3.4 ± 2.7
lung	56.0 ± 0.0	38.6 ± 2.3	34.8 ± 1.3	44.5 ± 1.7	36.7 ± 2.5	34.9 ± 2.4
monk	6.0 ± 0.0	15.8 ± 1.4	8.6 ± 1.3	17.4 ± 1.7	9.3 ± 1.2	12.4 ± 1.8
mush	22.0 ± 0.0	29.7 ± 1.3	18.3 ± 1.3	38.4 ± 1.2	27.9 ± 1.5	31.5 ± 1.6
nurs	8.0 ± 0.0	10.5 ± 2.3	12.4 ± 2.2	22.6 ± 2.7	14.6 ± 2.5	14.2 ± 2.8
park	23.0 ± 0.0	9.4 ± 1.2	14.9 ± 2.2	25.6 ± 1.8	19.7 ± 1.5	19.6 ± 1.2
pbc	10.0 ± 0.0	25.6 ± 1.4	19.4 ± 1.4	29.9 ± 2.6	21.7 ± 1.6	23.4 ± 1.2
pima	8.0 ± 0.0	14.2 ± 2.2	9.2 ± 2.7	10.5 ± 1.8	7.3 ± 2.3	7.0 ± 1.8
pop	8.0 ± 0.0	13.4 ± 1.1	11.5 ± 2.4	14.7 ± 1.6	8.2 ± 1.2	9.1 ± 1.2
seg	19.0 ± 0.0	28.8 ± 2.3	25.2 ± 1.3	39.9 ± 1.5	23.7 ± 2.5	28.9 ± 1.6
soy	35.0 ± 0.0	21.7 ± 2.2	20.5 ± 2.2	21.8 ± 2.3	28.6 ± 1.3	22.7 ± 2.5
SPECT	22.0 ± 0.0	19.6 ± 1.7	19.4 ± 1.5	23.1 ± 1.7	25.9 ± 2.2	24.7 ± 2.2
ttt	9.0 ± 0.0	20.2 ± 2.1	13.1 ± 1.5	15.8 ± 1.8	10.6 ± 2.5	14.2 ± 1.5
vot	16.0 ± 0.0	25.5 ± 1.8	22.6 ± 2.8	34.6 ± 1.5	15.6 ± 1.8	18.9 ± 1.6
wine	13.0 ± 0.0	7.3 ± 1.3	8.7 ± 2.4	9.8 ± 1.7	13.2 ± 2.3	12.8 ± 1.8
yeast	8.0 ± 0.0	8.6 ± 1.4	11.9 ± 1.6	20.7 ± 2.2	12.7 ± 1.7	12.2 ± 2.1
zoo	17.0 ± 0.0	12.5 ± 1.5	10.7 ± 1.1	20.3 ± 2.8	19.5 ± 1.7	16.2 ± 1.5
Rank	1.4	2.4	1.6	4.3	2.3	3.1

and 8G RAM machine, running Windows 7 operating system. We implemented our algorithms with C# programming language, using .Net framework v4.0.

As shown in Table 7, both of our proposed algorithms for learning MB classifiers, ABC-Miner+$_1$ and ABC-Miner+$_2$, took less computational time, overall, compared to the ABC-Miner algorithm, since they achieved on average about 79% and 87% of the running time of ABC-Miner, respectively.

Table 7 Running time (in seconds) results.

Dataset	CL-Tree		Algo-B		GHC-Acc		ABC+$_1$		ABC+$_2$		ABC
	time	ratio	time	ratio	time	ratio	time	ratio	time	ratio	time
abl	210	**0.12**	7400	**4.11**	1600	**0.89**	1300	**0.72**	1500	**0.83**	1800
bal	20	**0.01**	2105	**0.96**	2300	**1.05**	2700	**1.23**	3200	**1.45**	2200
bcw	34	**0.02**	1200	**0.6**	3500	**1.75**	980	**0.49**	1400	**0.7**	2000
car	27	**0.03**	700	**0.7**	5700	**5.7**	710	**0.71**	750	**0.75**	1000
chess	590	**0.16**	6800	**1.84**	6700	**1.81**	3100	**0.84**	2700	**0.73**	3700
cmc	78	**0.01**	2100	**0.39**	5800	**1.07**	2050	**0.38**	3200	**0.59**	5400
crd-a	21	**0.01**	2200	**1**	2100	**0.95**	2110	**0.96**	2100	**0.95**	2200
crd-g	36	**0.01**	1400	**0.31**	4300	**0.96**	4200	**0.93**	4100	**0.91**	4500
drm	26	**0.01**	500	**0.18**	4600	**1.64**	1900	**0.68**	2700	**0.96**	2800
ecoli	7	**0**	700	**0.47**	1600	**1.07**	1400	**0.93**	1600	**1.07**	1500
glass	8	**0**	400	**0.08**	700	**0.14**	3800	**0.76**	3800	**0.76**	5000
hay	6	**0.02**	15	**0.05**	200	**0.67**	150	**0.5**	180	**0.6**	300
hrt-c	21	**0.01**	400	**0.1**	2200	**0.52**	3600	**0.86**	3900	**0.93**	4200
hrt-s	32	**0.01**	600	**0.13**	4300	**0.96**	3600	**0.8**	4000	**0.89**	4500
hep	30	**0.01**	780	**0.31**	640	**0.26**	200	**0.08**	850	**0.34**	2500
iono	13	**0**	450	**0.17**	2200	**0.81**	1900	**0.7**	2100	**0.78**	2700
iris	1	**0.01**	10	**0.1**	100	**1**	100	**1**	110	**1.1**	100
lung	1	**0.04**	15	**0.6**	30	**1.2**	20	**0.8**	25	**1**	25
monk	70	**0.02**	470	**0.11**	3000	**0.7**	3700	**0.86**	5200	**1.21**	4300
mush	180	**0.12**	6000	**4**	1800	**1.2**	1200	**0.8**	1500	**1**	1500
nurs	90	**0.01**	6500	**0.76**	9000	**1.06**	6400	**0.75**	6000	**0.71**	8500
park	26	**0.01**	400	**0.19**	6000	**2.86**	2300	**1.1**	2100	**1**	2100
pbc	320	**0.09**	9200	**2.63**	3500	**1**	3200	**0.91**	3100	**0.89**	3500
pima	20	**0.01**	2105	**0.96**	2300	**1.05**	2700	**1.23**	3200	**1.45**	2200
pop	1	**0**	14	**0.02**	100	**0.14**	300	**0.43**	310	**0.44**	700
seg	160	**0.06**	1200	**0.43**	3900	**1.39**	2800	**1**	2700	**0.96**	2800
soy	24	**0.02**	320	**0.25**	4200	**3.23**	1160	**0.89**	890	**0.68**	1300
SPECT	45	**0.01**	690	**0.14**	4600	**0.96**	4700	**0.98**	4600	**0.96**	4800
ttt	24	**0.01**	3800	**1.46**	2300	**0.88**	1200	**0.46**	1200	**0.46**	2600
vot	35	**0.01**	500	**0.12**	1800	**0.44**	2800	**0.68**	3900	**0.95**	4100
wine	27	**0.11**	200	**0.8**	700	**2.8**	220	**0.88**	230	**0.92**	250
yeast	30	**0.01**	3500	**1.17**	3800	**1.27**	3200	**1.07**	3000	**1**	3000
zoo	29	**0.12**	180	**0.72**	250	**1**	190	**0.76**	210	**0.84**	250
Avg. ratio		**0.03**		**0.68**		**1.3**		**0.79**		**0.87**	

Moreover, both of them outperformed the greedy hill-climbing GHC-Acc for learning GBNs. Obviously, Naïve-Bayes and TAN algorithms took much less computational time compared to the other Bayesian classification learning algorithms used in the experiments.

8 Discussion

According to the predictive accuracy results, our two proposed ACO algorithms for learning MB classifiers obtained better results than the original ACO algorithm that learns BAN classifiers: ABC-Miner+$_1$ and ABC-Miner+$_2$ achieved better overall rank and reached the highest predictive accuracy results in more datasets compared to ABC-Miner. However, according to the Friedman test with post-hoc Holm test, only ABC-Miner+$_2$ was shown to be statistically better than ABC-Miner at the significance level of 10%. Note that in [48], we used the matched-pair samples Wilcoxon Signed-Rank statistical test [26] to compare the predictive accuracy results of ABC-Miner+$_1$ and ABC-Miner where the samples are the datasets. In that work ABC-Miner+$_1$ was shown to be statistically better at the significance level of 5%. However, in this current paper we are using different and more sophisticated statistical tests, which are necessary due to the fact that in the current paper we are comparing six algorithms (which requires the p-value to be corrected accordingly), unlike the case in [48], where only two algorithms were compared. Hence, overall the results show that the proposed extension for learning class variable MB structures, which allows capturing different types of dependency relationships between the class and the input variables, as well as performing embedded variable selection, improved the performance of the original ABC-Miner algorithm in terms of the predictive accuracy of the produced models.

On the other hand, ABC-Miner+$_1$ and ABC-Miner+$_2$ significantly outperformed the conventional BN classification algorithms, including Algorithm-B and GHC-Acc, which also produce MB classifiers. As discussed in Section 4.2, our proposed algorithms are focused on directly learning MB classifier structures, rather than building a GBN structures and then extracting the class variable's MB. In that sense, in our algorithms, all the computational budget (maximum number of solution evaluations as discussed in Section 6.2) is utilized for adding and evaluating relevant edges to the target class variable. However, in case of the GBN learning algorithms, the algorithm is allowed to add and evaluate any edge in the BN, which wastes part of the computational budget with irrelevant structures to the target class variable. Moreover, our proposed algorithms use the ACO-meta heuristic, which performs a global search that is less likely to get stuck into a local optimum in the search space, compared to the greedy local search performed by Algorithm-B and GHC-Acc.

In addition, both our proposed ACO algorithms were shown to be very competitive to well-known classification algorithms: Ripper, C4.5 and SVM. SVM, in particular, is currently a very popular type of classification algorithm because it often achieves higher predictive accuracy than other types of algorithms – although of course this is not always the case, as shown by the results in Table 4. Note, however, that a classifier built by SVM has the disadvantage of being a "black-box" from the perspective of users - i.e., the output of an SVM algorithm can hardly be interpreted by users. By contrast, MB classifiers represent graphical models of the dependencies between variables that can be directly interpreted by users, which is an advantage in many application do-

mains. For a review of the importance of comprehensible classification models, see [17,36,16].

Figure 4 shows an example of the comprehensible graphical model of an MB classifier, produced by ABC-Miner+$_2$, for the "voting records" dataset used in our experiments, with comparison to a BAN model produced by the original ABC-Miner algorithm for the same dataset. Note that the MB model shown in the figure has better classification accuracy and fewer edges, compared to its correspondent BAN model, which makes it more useful and reliable.

According to the model size results, the ABC-Miner+$_1$ and ABC-Miner+$_2$ algorithms that learn MB classifiers, overall, produced smaller BN models, in terms of the number of edges, compared to the BAN classifiers produced by the original ABC-Miner algorithm. This is due to the variable selection process, as well as allowing the class to have input variables as parents. These options have limited the number of the available edges to be added between the input variables in the network during the execution of the two variations of the ABC-Miner+ algorithm, compared to the BAN construction in ABC-Miner. This, in turn, leads to producing models with a smaller number of edges. Figure 4 shows an example of an MB classifier that has fewer edges compared to a BAN model constructed for the same dataset.

Besides, compared to GBN learning algorithms, our proposed algorithms for learning MB classifiers produced, overall, larger model sizes (however, ABC-Miner+$_1$ obtained better ranking than Algorithm-B). The reason behind this, as explained in the previous section, is that our algorithms use the computational budget to add relevant edges to the MB of the target class. On the other hand, for the Algorithm-B and GHC-Acc, during the search process, several irrelevant edges are tested and added to the BN, which wastes a lot of the allowed iterations without adding relevant edges to the class variables' MB. Therefore, the extracted MB would have a relatively low number of edges, with low predictive accuracy (see Section 7.1).

According to the computational time results, our extensions seemed to improve the execution performance of the ABC-Miner algorithm, as discussed in Section 5.4. In general, both of the proposed algorithms execute two steps (in two sequential phases as in ABC-Miner+$_1$, and in one integrated phase as in ABC-Miner+$_2$). The first step finds a primary structure that defines the edges between the class and the input variables, and the second step finds the edges among the input variables. If the primary structure – which defines the edges between the class and the input variables – has an input variable as a parent to the class variable, no edge pointing to this input variable becomes available in the process of finding the edges among the input variables. The first step reduces the search space for the second step, while the first step does not take much time. This makes the overall execution faster.

In addition, it is noticeable that GHC-Acc took more computational time compared to our proposed algorithms. As discussed earlier, the algorithm evaluates many irrelevant edges in the search space, which consumes a lot of execution time before finding an edge that is relevant to the target class prediction and can improve the classification accuracy of the model. Note that GHC-Acc

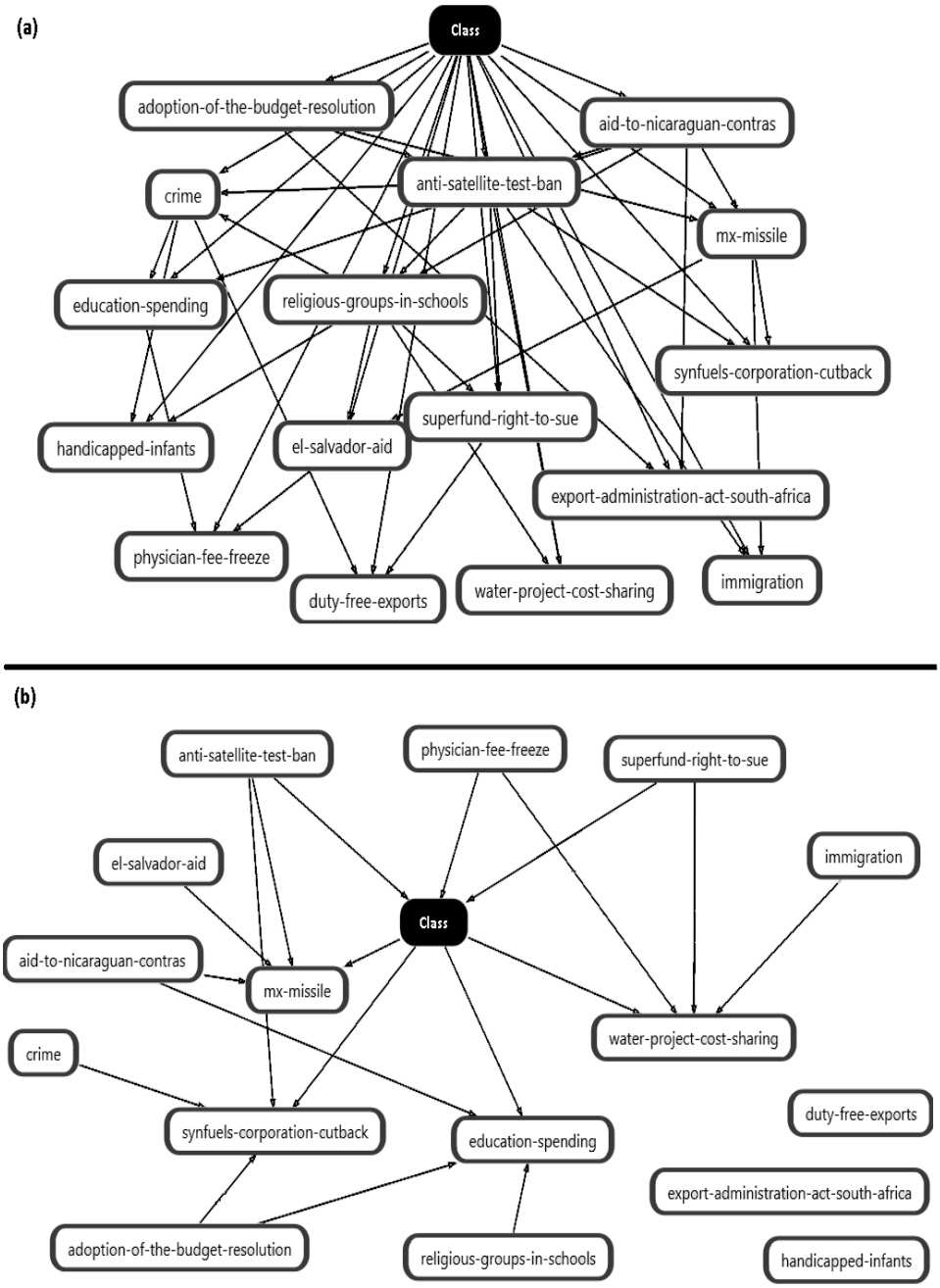

Fig. 4 BN classifiers output from voting records dataset. (a) A BAN that is produced by ABC-Miner, where the class variable is parent to all the input variables. (b) MBC that is produced by ABC-Miner+$_2$, where the class variable has three parent input variables, and three other input varibales are not included in the model as an effect of the feature selection.

is not designed to learn MB classifiers; rather, it learns GBNs using accuracy as a scoring function. Note also that, although Algorithm-B suffers from the same search space problem, it uses the decomposable K2 scoring function, which takes less time for BN re-evaluation.

9 Concluding Remarks

In this paper, we have proposed two new ACO algorithms for Markov blanket classifiers: ABC-Miner+$_1$, which utilizes two sequential phases, and ABC-Miner+$_2$, which uses one integrated phase in the learning process. Our proposed algorithms build BN structures in which the class variable does not have to be a parent to all the input variables, which performs an embedded variable selection with respect to the target class. Moreover, the algorithms allow having various dependency relationships between the class and an input variable, where the input variable can be a parent or a child (or have no relationship) to the class variables. These flexible structures can capture new conditional (in)dependency relationships that cannot be modelled by the BAN structure constructed by the original ABC-Miner algorithm, and probably compute more accurate class posterior probability.

In addition, we proposed a new probabilistic accuracy measure for evaluating the candidate BN structures constructed during the learning process, besides the structure complexity penalty that mitigates overfitting.

Empirical results in terms of predictive accuracy showed that our new algorithms outperformed our previously introduced ACO algorithm for learning BAN classifiers, as well as outperforming other conventional BN learning algorithms. In addition, our proposed ACO algorithms have been shown to be very competitive to other well-known classification algorithms: Ripper, C4.5 and SVM. Moreover, both ABC-Miner+$_1$ and ABC-Miner+$_2$ produced smaller-sized models, with less computational time, compared to ABC-Miner.

As a future work, we would like to evaluate the quality of the produced model using probability-based measures, such as Quadratic Loss Funciton, Bayesian Information Reward (BIR) and Kullback-Leibler (KL) divergence.

Moreover, we would like to investigate the effect of each over-fitting mitigation mechanisms separately to evaluate the contribution of each in improving the predictive performance of the algorithms. This can be accomplished comparing the classification accuracy of the produced model on the training set to the predictive accuracy of the model on the test set, with and without each mechanism. Plus, we would like try to remove the k-parents restriction in the construction of the Bayesian network classifiers, and introduce a generalized formula to penalize the complexity of the model with respect to the number of parents that each node has in the network, including the class node.

Besides, we would like to build MB classifiers for hierarchical classification problems [29]. A preliminary work in this direction has been done in [46], but that paper used ABC-Miner in a local-classifier-per-node hierarchical approach. We would like to extend the ABC-Miner+ algorithms to tackle the

hierarchical classification in a global-classifier approach. It would be also interesting to try our proposed ACO-based algorithm – in the context of flat classification – with new real-world classification datasets from domains such as fraud detection, churn prediction, targeted marketing, and social media analysis, rather than the typical UCI datasets.

References

1. A. Asuncion and D.J. Newman. UCI Machine Learning Repository. URL:http://www.ics.uci.edu/ mlearn/MLRepository.html. 2007.
2. Wray Buntine. Theory Refinement on Bayesian Networks. In *17th Conference on Uncertainty in Artificial Intelligence*, pages 52–60, San Francisco, CA, USA, 1991. Morgan Kaufmann.
3. Jie Cheng and Russell Greiner. Comparing Bayesian Network Classifiers. In *15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 101–108, San Francisco, CA, USA, 1999. Morgan Kaufmann.
4. Jie Cheng and Russell Greiner. Learning Bayesian Belief Network Classifiers: Algorithms and System. In *14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, pages 141–151, London, UK, 2001. Springer.
5. D. Chickering, M. Geiger, and D. Heckerman. Learning Bayesian Networks is NP-Hard. *Advanced Technologies Division, Microsoft Corporation, Technical Report*, 1994.
6. D. Chickering, D. Heckerman, and C Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.
7. G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9(4):309–347, 1992.
8. Rónán Daly and Qiang Shen. Learning Bayesian Network Equivalence Classes with Ant Colony Optimization. *Journal of Artificial Intelligence Research (JAIR)*, 35:391–447, 2009.
9. Rónán Daly, Qiang Shen, and Stuart Aitken. Review: Learning Bayesian Networks: Approaches and Issues. *Knowledge Engineering Reviews*, 26(2):99–157, 2011.
10. L. M. de Campos, J. M. Fernandez-Luna, J. A. Gamez, and J. M. Puerta. Ant Colony Optimization for Learning Bayesian Networks. *International Journal of Approximate Reasoning*, 31(3):291–311, 2002.
11. Janez Demsar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 1(7):1–30, 2006.
12. Marco Dorigo, Gianni M. Caro, and Luca M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, 1999.
13. Marco Dorigo and Thomas Stützle. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. volume 57 of *OPRMS*, pages 250–28, New York, NY, USA, 2003. Springer.
14. Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA, 2004.
15. Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1973.
16. A.A. Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD Explorations*, 15(1):1–10, 2013.
17. A.A. Freitas, D.C. Wieser, and R. Apweiler. On the importance of comprehensible classification models for protein function prediction. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 7(1):172–182, 2010.
18. Nir Friedman, Dan Geiger, Moises Goldszmidt, G. Provan, P. Langley, and P. Smyth. Bayesian Network Classifiers. *Machine Learning*, 29:131–163, 1997.
19. Nir Friedman and Moises Goldszmidt. Learning Bayesian networks with local structure. *Learning in Graphical Models, Norwell, MA: Kluwer*, pages 252–262, 1998.

20. Salvador Garca and Francisco Herrera. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.
21. Yaniv Gurwicz and Boaz Lerner. Bayesian Class-Matched Multinet Classifier. In *International Conference on Structural, Syntactic, and Statistical Pattern Recognition (IAPR'6)*, pages 145–153, Berlin, Heidelberg, 2006. Springer.
22. Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2nd edition, 2000.
23. David Heckerman. A Tutorial on Learning with Bayesian Networks. *Studies in Computational Intelligence: Innovations in Bayesian Networks*, 156:33–82, 2008.
24. David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20(3):197–243, 1995.
25. Kaizhu Huang, I. King, and M.R. Lyu. Discriminative Training of Bayesian Chow-Liu Multinet Classifiers. In *International Joint Conference on Networks*, volume 1, pages 484–488, New York, NY, USA, 2003. IEEE Press.
26. Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.
27. L. Jiang, H. Zhang, and Z. Cai J. Su. Evolutional Naive Bayes. In *1st International Symposium on Intelligent Computation and its Applications*, pages 344–350. China University of Geosciences Press, 2005.
28. Liangxiao Jiang, Dianhong Wang, Zhihua Cai, and Xuesong Yan. Survey of Improving Naive Bayes for Classification. In *3rd International Conference on Advanced Data Mining and Applications (ADMA'07)*, number 4632 in LNCS, pages 134–145, Berlin, Heidelberg, 2007. Springer.
29. C.N. Silla Jr. and A.A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.
30. Eugene Santos Jr. and Ahmed Hussein. Case-Based Bayesian Network Classifiers. In *17th International FLAIRS Conference, AAAI*, volume 5, pages 598–605, Stanford, USA, 2004. AAAI Press.
31. Josef Kittler. *Handbook of Pattern Recognition and Image Processing*. Academic Press, New York, 1986.
32. Kevin B. Korb and Ann E. Nicholson. *Bayesian Artificial Intelligence*. CRC Press, San Francisco, CA, USA, 2nd edition, 2011.
33. Pat Langley. Induction of Recursive Bayesian Classifiers. In *European Conference on Machine Learning (ECML)*, pages 153–164, Berlin, Heidelberg, 1993. Springer.
34. Pat Langley and Stephanie Sage. Induction of Selective Bayesian Classifiers. In *10th Conference on Uncertainty in Artificial Intelligence*, pages 399–406, San Francisco, CA, USA, 1994. Morgan Kaufmann.
35. Huan Liu and Hiroshi Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Springer, Berlin, Heidelberg, 1st edition, 1998.
36. D. Marteens, J. Vanthienen, W. Verbeke, , and B. Baesens. Performance of Classification Models from a User Perspective. *Decision Support Systems*, 51(4):782–793, 2011.
37. D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens. Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11:651–665, 2007.
38. T.M. Mitchell. The need for biases in learning generalizations. *Readings in Machine Learning*, 10:184–191, 1980.
39. F.E. Otero, A.A. Freitas, and C.G Johnson. Handling continuous attributes in ant colony classification algorithms. In *IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*, pages 225–231, New York, NY, USA, 2009. IEEE Press.
40. R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.
41. J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2000.
42. Pedro C. Pinto, Andreas Nägele, Mathäus Dejori, Thomas A. Runkler, and Ao. Using a Local Discovery Ant Algorithm for Bayesian Network Structure Learning. *IEEE Transactions on Evolutionary Computation*, 13(4):767–779, 2009.

43. K.M. Salama, A.M. Abdelbar, and A.A. Freitas. Multiple Pheromone Types and Other Extensions to the Ant-Miner Classification Rule Discovery Algorithm. *Swarm Intelligence*, 5(3-4):149–182, December 2011.
44. K.M. Salama, A.M. Abdelbar, F.E. Otero, and A.A. Freitas. Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery. *Applied Soft Computing*, 13(1):667–675, 2013.
45. K.M. Salama and A.A. Freitas. ABC-Miner: an Ant-based Bayesian Classification Algorithm. In *8th International Conference on Swarm Intelligence (ANTS'12)*, number 7461 in LNCS, pages 13–24, Berlin, 2012. Springer.
46. K.M. Salama and A.A. Freitas. ACO-based Bayesian Network Ensembles for the Hierarchical Classification of Ageing-Related Proteins. In *The European Conference on Evolutionary Computation, Machine Learning and Data Mining in Computational Biology (EvoBio'13)*, number 7833 in LNCS, pages 80–91, Berlin, Heidelberg, 2013. Springer.
47. K.M. Salama and A.A. Freitas. Clustering-based Bayesian Multi-net Classifier Construction with Ant Colony Optimization. In *IEEE Congress on Evolutionary Computation (IEEE CEC) (2013)*, pages 3079–3086, New York, NY, USA, 2013. IEEE Press.
48. K.M. Salama and A.A. Freitas. Extending the ABC-Miner Bayesian Classification Algorithm. In *6th International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO'13)*, volume 512 of *Studies in Computational Intelligence*, pages 1–12, Berlin, 2013. Springer.
49. K.M. Salama and A.A. Freitas. Learning Bayesian Network Classifiers Using Ant Colony Optimization. *Swarm Intelligence*, 7(2-3):229–254, 2013.
50. Eugene Santos and Ahmed Hussein. Comparing case-based bayesian network and recursive bayesian multi-net classifiers. In *International Conference on Artificial Intelligence (ICAI)*, pages 627–633, 2004.
51. J. Smaldon and A.A. Freitas. A new version of the ant-miner algorithm discovering unordered rule sets. In *Genetic and Evolutionary Computation Conference (GECCO'06)*, pages 43–50. ACM Press, 2006.
52. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison Wesley, 2nd edition, 2005.
53. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco, CA, USA, 3rd edition, 2010.
54. Yanghui Wu, John McCall, and David Corne. Two Novel Ant Colony Optimization Approaches for Bayesian Network Structure Learning. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7, New York, NY, USA, 2010. IEEE Press.
55. Shulin Yang and Kuo-Chu Chang. Comparison of Score Metrics for Bayesian Network Learning. *IEEE Transactions on Systems, Man, and Cybernetics - Part A*, 32(3):419–428, 2002.
56. Fei Zheng and Geoffrey I. Webb. Semi-naive Bayesian Classification. *Journal of Machine Learning Research*, 87(1):93–125, 2008.