Ant Colony Algorithms for Constructing Bayesian Multi-net Classifiers

Khalid M. Salama and Alex A. Freitas
School of Computing, University of Kent, Canterbury, UK.
{kms39,A.A.Freitas}@kent.ac.uk

December 5, 2013

Abstract

Bayesian Multi-nets (BMNs) are a special kind of Bayesian network (BN) classifiers that consist of several local Bayesian networks, one for each predictable class, to model an asymmetric set of variable dependencies given each class value. Deterministic methods using greedy local search are the most frequently used methods for learning the structure of BMNs based on optimizing a scoring function. Ant Colony Optimization (ACO) is a meta-heuristic global search method for solving combinatorial optimization problems, inspired by the behavior of real ant colonies. In this paper, we propose two novel ACO-based algorithms with two different approaches to build BMN classifiers: ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$. The former uses a local learning approach, in which the ACO algorithm completes the construction of one local BN at a time. The latter uses a global approach, which involves building a complete BMN classifier by each single ant in the colony. We experimentally evaluate the performance of our ant-based algorithms on 33 benchmark classification datasets, where our proposed algorithms are shown to be significantly better than other commonly used deterministic algorithms for learning various Bayesian classifiers in the literature, as well as competitive to other well-known classification algorithms.

Keywords. Ant Colony Optimization (ACO), Classification, Bayesian Network Classifiers, Bayesian Multi-nets.

1 Introduction

Bayesian network (BN) classifiers are well-known kind of graphical probabilistic models that aim to perform class prediction by modeling the dependency relationships between the predictor attributes (variables) in a dataset given the class variable. To classify a given case, the BN classifier computes the posterior probability of each available class value, given the values of the attributes of the case, and then labeling the case with the class having the highest posterior

probability [1, 2]. Several Bayesian network classifiers were introduced in the literature: Naïve-Bayes, Tree Augmented Naïve-Bayes (TANs), Bayesian networks Augmented Naïve-Bayes (BANs) and General Bayesian Networks (GBNs) [3, 2], where a single (probably complex) network structure is built to model the variable dependencies from the whole dataset. In contrast, a Bayesian Multi-net (BMN) classifier consists of several local networks, one for each available class value. Each local Bayesian network in the class-based BMN has a different set of (in)dependencies amongst the variables with respect to a specific class value, which promotes the model's predictive effectiveness and reduces its structural complexity [2, 4].

Ant Colony Optimization (ACO) [5] is a meta-heuristic typically used for solving combinatorial optimization problems, inspired by observations of the intelligent behavior (in particular, finding the shortest path between two points) of ant colonies in nature. ACO has been effectively used for learning general-purpose BNs (rather than BN classifiers) [6, 7, 8, 9], as well as different types of classification models [10, 11, 12, 13, 14, 15]. However, ABC-Miner (where "ABC" stands for Ant-based Bayesian Classifier), recently introduced by the authors in [16, 17], is the only algorithm to use Ant Colony Optimization for learning BN *classifiers* by constructing a BAN structure with at most k-dependencies from a dataset. The ABC-Miner algorithm has outperformed conventional deterministic algorithms in terms of predictive accuracy [16, 17].

The motivation behind this work is the following. Although learning the optimal Bayesian network structure from a dataset is $\mathcal{NP}$-hard [18], most of the algorithms used in the literature for building Bayesian network classifiers utilize greedy local search and deterministic techniques; while several stochastic heuristic global search algorithms can be effectively applied to the task of building high-quality networks in an acceptable computational time. ACO has been successful in solving several types of data mining problems, including classification rule induction [12, 19, 13, 10, 11, 14] and BN classifier structure learning [16, 17]. Therefore, we carry on exploiting the ACO paradigm and exploring different techniques for building Bayesian classifiers.

Unlike ABC-Miner [16, 17], which builds a single BAN model, in this paper we propose two novel ACO-based algorithms which build BMN classifiers, where a different Bayesian network is built for each class label. These two algorithms follow different approaches to build BMN classifiers, and are called: ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$. The former uses a local learning approach, in which the ACO algorithm completes the construction of one local BN at a time. The latter uses a global approach, which involves building a complete BMN classifier in a single ant trail of the ACO algorithm. We describe all the elements necessary to tackle our target learning problem using ACO, and experimentally compare the performance of our ACO-based algorithms for building BMN classifiers with the recently introduced ABC-Miner, with other conventional algorithms for learning BN classifiers used in the literature, and with 3 other types of classification algorithms: rule induction, decision tree construction and support vector machines. Note that this work is the first to utilize the ACO meta-heuristic for building class-based BMN classifiers.

The structure of the paper is as follows. Section 2 gives a brief overview of concepts and methods from the relevant areas of Ant colony Optimization, Bayesian networks and Bayesian multi-net classifiers. In Section 3, we define the essential elements for applying the ACO meta-heuristics to the problem of learning BMN classifiers. We propose ABC-Miner$_l^{mn}$ that uses a local approach, and ABC-Miner$_g^{mn}$ that uses a global approach for learning BMN classifiers using ACO algorithms in Sections 4 and 5, respectively. Section 6 discusses our experimental methodology, while the results and their analysis are presented in Section 7. Finally, we conclude with some general remarks and provide possible directions for future research in Section 8.

2 Background

2.1 Ant Colony Optimization

Social insect swarms are distributed systems that, in spite of the simple behavior of their individuals, produce a group behavior that is able to effectively accomplish complex tasks. Inspired by the behavior of natural ant colonies, Dorigo et al. [5, 20, 21] have defined an artificial ant colony meta-heuristic that can be applied to solve optimization problems, called ant colony optimization (ACO). The main idea is to utilize a swarm of simple individuals that use collective behavior to achieve a certain goal, such as finding the shortest path between a food source and the nest. ACO algorithms simulate the behavior of real ants using a colony of artificial ants, which cooperate in finding good solutions to optimization problems. The outline of the basic ACO procedures is presented 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
```

Each artificial ant creates candidate solutions to the problem at hand. Communication between artificial ants in the colony is performed by depositing

pheromone on the part of the search space that was visited while constructing the solution, indicating its quality, in order to guide the search. The iterative process of building candidate solutions, evaluating their quality and updating pheromone values allows an ACO algorithm to converge to near-optimal solutions. In order to apply the ACO meta-heuristic to a given optimization problem, the following elements should be defined in advance:
– **Construction Graph** - An appropriate representation of the problem's search space that contains the available decision components, with which an ant can incrementally construct a candidate solution.
– **Heuristic Information** - Problem-specific knowledge that is associated with each decision component in the construction graph and influences the choice of which components will be used for constructing candidate solutions.
– **State Transition Formula** - A probabilistic transition rule that uses the heuristic value η and the pheromone amount τ associated with the decision components for an ant to move in the search space from a state to another (i.e. selecting a decision component) during the solution construction process.
– **Quality Evaluation Function** - A problem-specific function by which the quality of a constructed candidate solution is evaluated for the purpose of pheromone update. The higher the quality of a solution constructed, the more pheromone will be deposited on the decision components used in that solution, which will encourage other ants to select those decision components in future iterations.
– **Pheromone Update Strategy** - A formula to be used for pheromone reinforcement and evaporation. Pheromone reinforcement is applied on decision components occurring in the constructed solution in proportion to its quality, while pheromone evaporation is applied on the whole construction graph to avoid stagnation and early convergence.
– **Local Optimizer** - An optional local search procedure to improve the quality of a constructed solution. This can be performed on each candidate solution, or just on the best solution among the colony to reduce computational time.

All the aforementioned elements for designing an ACO-based algorithm are concretely defined later in the context of learning Bayesian multi-net classifiers in Section 3.

2.2 Bayesian Networks

Bayesian networks are knowledge representation and reasoning tools that model dependency and independency relationships amongst variables in a specific domain [22]. A directed acyclic graph (DAG) is used to model the network structure G where the nodes represent the domain variables, and the edges between the nodes represent statistical dependencies between these variables. In addition, a conditional probability table (CPT) is obtained for each variable with respect to its parents in the network. The set of CPTs represent the parameters Θ of the network, which quantifies the dependency relationships between the variables. A BN(G,Θ) specifies a joint probability distribution over the set of

variables $\mathbf{X}$ that is formulated in the product form:

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

Learning a Bayesian network from a dataset (in which the attributes are referred to as variables) consists of two phases: learning the network structure, and then learning the parameters of the network. Parameter learning is considered a relatively straightforward process for any given BN structure with specified (in)dependencies between variables, since the values of a CPT can be estimated directly from the data by the relative frequencies of each variable with respect to its parents. The CPT of variable X_i encodes the likelihood of each value of this variable given each combination of values of its parents in the graph G. On the other hand, the aim of network structure learning is to find the graph G that best fits a given dataset $\mathbf{D}$ in terms of $P(\mathbf{D}|G)$ This is known as the scoring-based approach [23, 22]. The algorithm uses a scoring function that evaluates each G with respect to $\mathbf{D}$, searching for the network structure that maximizes the value of the scoring function. K2, MDL, KL, BDEu and several other scoring functions can be used for this task [24, 25, 26].

A recent, very comprehensive review on learning Bayesian networks approaches and issues is presented by Daly et al. in [22]. For further information about Bayesian networks, the reader is referred to [25, 23], which also provide a detailed discussion of the subject.

2.3 Bayesian Multi-net Classifiers

Bayesian Network Classifiers (BNCs) [27, 2, 4] are a special kind of the probabilistic networks, which focus on answering queries about the probability of a specific node: the class attribute. A Bayesian multi-net (BMN) is composed of the prior probability distribution of the class node and a set of local Bayesian networks. Each Local BN corresponds to a value that the class variable can have [3, 2, 28]. Unlike other types of BNs, a BMN allows different (in)dependencies relationships amongst variables – given a dataset – to be represented separately, with respect to each individual class value, as shown in Figure 1. For instance, for a given pair of variables X_i and X_j, the actual dependency between them might be best represented as $(X_i \rightarrow X_j)$, given one class value, and as $(X_j \rightarrow X_i)$ given another class value, and there might be no dependency between them given yet another class value. Even if the same kind of dependency (with the same direction) is represented in two local BNs for different class values, it is possible that the precise values of the CPTs associated with that dependency would be different in the two local BNs, since each one is learnt from a different data subset (as discussed next). This should lead to a better modeling for reasoning and class prediction [2].

A BMN classifier is learnt by partitioning the training set $\mathbf{D}$ into $|C|$ subsets, where $|C|$ is the number of values in the domain of the class attribute. Each subset D_l would contain only the instances labeled by class value l. Then, a

general (not a classifier by itself) Bayesian network BN_l is built for each class $C = l$ with $\mathbf{X} = X_1, X_2, ..., X_n$ variables using the D_l subset, to capture the variable-dependency relationship given the specific l class value. Thus, a BMN classifier is a set of local BNs $\{BN_1, BN_2, ..., BN_{|C|}\}$ that, together with the prior probability distribution of C, classifies an instance x by choosing the class $C(x)$ that maximizes the posterior probability, as shown in Equation 2.

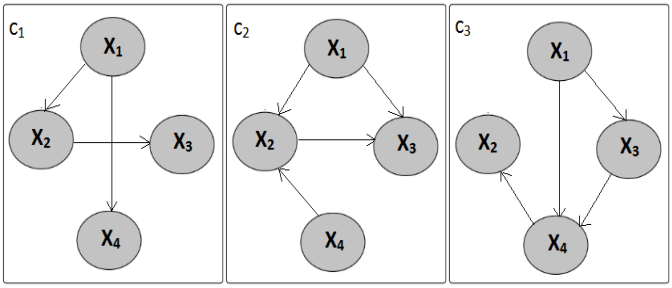
$$C(x) = \underset{\forall\ l \in C}{\arg\max}\, P(\mathbf{x} = x_1, x_2, ..., x_n | BN_l) \times P(C = l), \tag{2}$$

$$P(\mathbf{x} = x_1, x_2, ..., x_n | BN_l) = \prod_{i=1}^{n} P(x_i | \mathbf{Pa}(X_i), BN_l) \tag{3}$$

According to Equations 2 and 3, in order to classify an instance using a BMN classifier, we select the class of the local BN that maximizes the probability of this instance. In other words, we ask each local BN what is the probability of this instance, and assign the instance with the class of the local BN that answers with the highest probability, which means that this instance is more likely to belong to the domain of that class - in terms of the variable-dependency structure and parameters - than to any other class.

In BMNs, building the local models for each data subset can be performed as a general BN learning process. A well-known greedy algorithm for learning a BN structure is Algorithm-B, proposed in [29], which searches for the network structure that optimizes the value of a scoring metric, such as K2, cross-entropy or the Kullback distance [2, 30]. Chow-Liu tree Multi-net [2, 4] is a remarkable algorithm for learning local BNs due to its simplicity and effectiveness. The algorithm builds a tree-like network structure, based on the mutual information between the variables. An extension to the Chow-Liu tree Multi-net was proposed in [30], which involves maximizing the cross-class divergence. The Bayesian class-matched multi-net algorithm [31] is another extension that uses a scoring function based on detection-rejection behavior. The recursive Bayesian Classifier induction [32] can build multiple local BNs for the same class by further dividing its data subset recursively.

Figure 1: A Bayesian multi-net that consists of three different local BNs, one for each class value. Each local BN has a different network structure that asserts the variable dependencies in its corresponding class value data subset.



2.4 ACO Related Work

Ant Colony Optimization has contributed effectively in tackling the classification problem. Parpinelli et al. proposed Ant-Miner [10], the first ant-based classification algorithm, which learns a set of classification rules of the form: [IF conditions THEN class]. Several extensions, such as AntMiner+ by Martens et. al. [12], *c*Ant-Miner by Otero et al. [19, 13], and multi-pheromone Ant-Miner by Salama et al. [33, 11, 14, 34] have been introduced in the literature. A recent survey on Ant-Miner and its related work is presented in [35]. Besides, Ant-Tree-Miner by Otero et al.[36] and *c*ACDT by Boryczka et al. [37] are recently-introduced ACO algorithms for building decision trees for classification. Note that our work also utilizes ant-based algorithms to handle classification problems, yet with a very different approach: learning Bayesian networks to be used as classifiers.

Besides, ACO has been employed for learning *general*-purpose BNs, rather than classifiers, in several works, including ACO-B by Campos et al. [7], MMACO by Pinto et al. [38, 9], ACO-E by Daly et al. [39, 8], CHAINACO and K2ACO by Yanghui et al. [6] and recently HACO-B by Junzhong et al. [40]. These various works are briefly discussed in [17].

However, in the area of Bayesian classification, the authors have recently introduced ABC-Miner [16] and extended the work in [17], at present the only algorithm that tackles the classification problem via using ACO for learning a BN *classifier* in the structure of a BAN, with at most k-dependencies at each variable node. The ant-based ABC-Miner algorithm was shown to outperform other conventional BN classifier learners proposed in the literature in terms of predictive accuracy, overall, across 25 benchmarking classification datasets from the well-known UCI dataset repository. For detailed discussion of the algorithm and it results, the reader is referred to [16, 17]. Note that, unlike the current work, ABC-Miner learns one BN model on a given dataset to perform the classification task. Yet, in this work, we use ACO to learn a BMN for a given dataset, which consists of several local BN models, one for each class value, which is a different and more elaborated technique for Bayesian classification.

Besides, we have recently introduced an ACO-based algorithm that learns cluster-based BMNs [41], rather than learning class-based BMNs; the focus of this current work. Cluster-based BMN refers to partitioning the dataset into arbitrary data subsets, and learning a local BN classifier for each subset. While the procedure of ABC-Miner is comparable to the algorithms proposed in our current work, the ACO clustering-based BMN algorithm [41] is very different for the following reason. In the current proposed algorithms, ACO is employed for finding the optimal structure of the local BNs – similar to ABC-Miner, in which the ACO algorithm optimizes the structure of BAN classifiers. On the other hand, in [41], ACO is utilized to produce the data clusters, rather than learning the local BN classifiers, while a simple Naïve-Bayes classifier is constructed for each data cluster.

3 ACO Elements for learning BMN Classifiers

In this work, we are concerned with building the structure of a Bayesian multinet. Unlike ABC-Miner that builds a single BAN model to represent the variable (in)dependency relationships with respect to the class variable, our target is to build several local Bayesian networks BN_l, one for each class value $l \in \mathbf{C}$, to compose a BMN classifier. Each local BN_l should model the (in)dependencies between the input variables (attributes) with respect to its specific class value l. This approach explicitly encodes asymmetric class-based (in)dependencies assertions amongst the variables that cannot be represented in the structure of a single Bayesian network. This in turn aims to improve the predictive power of the Bayesian model, and simplify its interpretability as well.

The construction of a BMN classifier is carried out using two new ant-based algorithms, ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$, which utilize two different approaches. While each approach is described later in a separate section, the common ACO meta-heuristic elements that we used in our algorithms are presented in this section. It is important to highlight that one of the most important elements of any ACO-based algorithm is the function by which the quality of a constructed candidate solution is evaluated. However, the quality evaluation functions used for the two proposed algorithms are not the same, since the type of constructed solution differs from one approach to the other. ABC-Miner$_l^{mn}$ finishes the construction of a *general* local BN before it proceeds to the next local BN, while in ABC-Miner$_g^{mn}$, each ant builds a complete BMN classifier at once. Thus, each algorithm uses a different quality evaluation function, which will be discussed later in the context of each proposed approach.

3.1 Construction Graph

Similar to ABC-Miner, the decision components in the search space are all the edges $X \rightarrow Y$ where $X \neq Y$ and X, Y belongs to the input attributes of the data subset D_l. These components (edges) represent the variable (attribute) dependencies in the candidate local BN constructed by an ant. An ant should select a good and valid combination of these decision components (variable-dependency relationships) to construct a candidate solution (BN structure). A representation of the construction graph for a dataset with five input attributes (besides the class attribute) is shown in Figure 2.

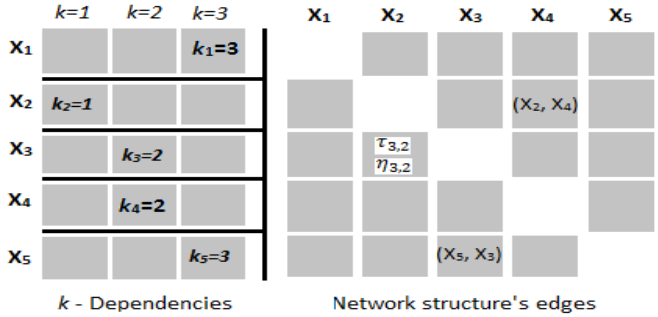
At the beginning, the pheromone amount is initialized for each decision component with the same value, given by: $1/TotalEdges$, where $TotalEdges$ is the number of edges associated with a complete graph (with an edge between every pair of nodes). In addition, the heuristic value for each edge $X \rightarrow Y$ is set using the mutual information, which is computed as follows:

$$I(X, Y|D_l) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \tag{4}$$

Mutual information is a measure of correlation between two variables. Such a function is used as the local heuristic information associated with the edges

in order to lead the ants during the search to the edges between correlated variables for building a local BN. Unlike conditional mutual information, used by ABC-Miner, the calculation of the mutual information in this context is more accurate with respect to the local BN_l. Since the mutual information between variables is calculated using only subset D_l, which contains only the instances labeled with class value l, the mutual information is conditioned on the class value l, which may change between the same two variables in different local BNs (i.e., different class values). However, the conditional mutual information between two variables is conditioned on the class variable as a whole and using the whole dataset, which makes the current approach for calculating the mutual information more class-based relevant.

Figure 2: Matrix representations of the construction graph for a five-input-attributes dataset. The elements on the left represent the number of parents that each variable can have in the network. The decision components on the right are the edges to build the network structure, each has a current pheromone amount τ, and heuristic information η.



3.2 State Transition Rule

The selection of the edges is performed according to the following probabilistic – the standard formula in the ACO literature [5]:

$$P_{ij} = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{a}^{I} \sum_{b}^{J} [\tau_{ab}(t)]^{\alpha} \cdot [\eta_{ab}]^{\beta}} \tag{5}$$

In Equation 5, P_{ij} is the probability of selecting the edge $i \rightarrow j$, $\tau_{ij}(t)$ is the current amount of pheromone associated with edge $i \rightarrow j$ at iteration t and η_{ij} is the heuristic information for edge $i \rightarrow j$. The edge $a \rightarrow b$ represents a valid selection in the set of available edges. An edge $a \rightarrow b$ is valid to be added in the local BN being constructed if the following two criteria are satisfied: 1) its inclusion does not create a directed cycle, 2) the upper limit of k_b parents for the child variable b (discussed in the next subsection) is not exceeded by the inclusion of the edge. After the ant adds a valid edge to the local BN, all the invalid edges are eliminated from the construction graph.

The exponents α and β are used to adjust the relative emphases of the pheromone (τ) and heuristic information (η), respectively. Our proposed algorithms adapt the "ants with personality" technique [11]. In other words, some ants will give more emphases to pheromone information, while others will give more emphases to heuristic information. The α_i and β_i parameters are each independently drawn from a Gaussian distribution centered at 2 with a standard deviation of 1, as used in [11]. This approach aims to advance exploration and promote diversity in the colony during the search process.

3.3 k-Parents Selection

In order to build the structure of a Bayesian network, the number of parents (dependencies) that a node (variable) can have in the network should be specified firstly. The maximum number of parents parameter is usually fixed for all the variables in the network, and typically is set to a small value both for the sake of computational efficiency and to avoid over-fitting.

Instead of having the user selecting the optimum number of dependencies that a variable in the BN can have (at most k parents for each node), this selection is carried out by the ants in the ACO algorithm in a self-adaptive manner. Moreover, we allow each variable to have its own number of parents selected independently prior to solution construction. As shown in the left side of Figure 2, the number of dependencies is treated as decision components as well. The selection of k_i value 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 BN classifier 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 BN classifier, which used value k_i for variable i in the process of the 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.

3.4 Local Optimization

The best solution *tbest* constructed amongst the ants in the colony at the current iteration t undergoes local search, which aims to improve its quality, with respect to the used quality measure function for each learning approach.

The local search procedure works as follows. It temporarily removes one edge at a time from the constructed BN, considering edges in the reverse order of their addition to the BN (i.e. removing first the last edged that was added to the BN). If this edge removal improves the quality of the BN, this edge is removed permanently from the BN; otherwise it is added once again. Then it proceeds to the next edge, and this process is iteratively repeated until all the edges are tested to be removed from the constructed network and the BN with

the highest quality – with respect to the used quality measure – is obtained, and used for pheromone update.

3.5 Pheromone Update

After the current iteration's best solution is optimized via local search, pheromone amounts are updated on the two types of decision components in the construction graph, i.e. on network edges as well as on the numbers of parents (variable dependencies) to be selected for each node for constructing a BN classifier structure. Pheromone update is carried out to affect the probability of selecting the decision components for building further candidate solutions according to the quality of the previously constructed ones, and consists of two phases; pheromone deposit and evaporation.

Concerning pheromone deposit, the pheromone amount is increased on each decision component (edge $i \rightarrow j$) according to the quality of two constructed solutions: the iteration best *tbest* and the global best *gbest* using the following weighting reinforcement strategy:

$$\tau_{ij}(t+1) = \begin{cases} \tau_{ij}(t) + \phi_1.Q_{tbest}(t) & \textbf{if } i \rightarrow j \in tbest \\ \tau_{ij}(t) + \phi_2.Q_{gbest}(t) & \textbf{if } i \rightarrow j \in gbest \end{cases} \tag{6}$$

where ϕ_1 and ϕ_2 represent the *intensity* of the pheromone to be deposited in iteration t according to the quality of the iteration best and global best solutions respectively, and are calculated as:

$$\phi_1 = \frac{max\_iterations - t}{max\_iterations}, \quad \phi_2 = \frac{t}{max\_iterations} \tag{7}$$

Hence, in early iterations more weight is given to the local best rather than the global best (as $max\_iterations - t$ is greater than t). This is applied in order to improve search diversity. However, as the iterations go on, the quality of the global best increases, which gains more weight (as t increases and $max\_iterations - t$ decreases), leading to convergence towards a good solution. Note that the two pheromone deposit conditions in Equation 6 are applied on the same edge if it belongs to both *tbest* and *gbest*, and $\phi_1 + \phi_2$ is always 1 at any given iteration t.

Pheromone normalization (to simulate evaporation as in [10]) is then applied to all the τ_{ij} in the construction graph by dividing each τ_{ij} over the total amount of pheromone on the edges in the construction graph. An analogous normalization is also carried out for the decision components representing the maximum number of dependencies.

4 A Local ACO Approach for Learning BMN Classifiers

4.1 The ABC-Miner$_l^{mn}$ Algorithm Outline

The BMN classifier learning procedure in ABC-Miner$_l^{mn}$ is carried out in a local fashion. The Ant Colony Optimization meta-heuristic is utilized to build, independently, a local Bayesian network BN_l for class value l, using subset D_l in the training set $\mathbf{D}$ which contains only the instances labeled with class value l. This is repeated for each class value belonging to the domain of the class variable C. We treat each local process as a separate problem, where a *general* Bayesian network is built; each local BN by itself is not a classifier. Nevertheless, we consider – in quality evaluation – that this local BN will play a part in a global classification problem solved by the finally produced BMN classifier. The overall process of ABC-Miner$_l^{mn}$ is shown in Algorithm 2.

Algorithm 2 Pseudo-code of ABC-Miner$_l^{mn}$.

```
Begin
D ← training_set; BMC ← φ;
for l = 1 to |C| do
    D_l ← Subset(D, l);
    BN_l(gbest) ← φ;
    Q(gbest) ← 0;
    t ← 1;
    InitializePheromoneAmounts();
    InitializeHeuristicValues();
    repeat
        BN_l(tbest) ← φ; Q(tbest) ← 0;
        for i = 1 to colony_size do
            BN_l(i) ← ant(i).CreateSolution(D_l);
            Q(i) ← ComputeQuality(BN_l(i), D);
            if Q(i) > Q(tbest) then
                BN_l(tbest) ← BN_l(i);
                Q(tbest) ← Q(i);
            end if
        end for
        PerformLocalSearch(BN_l(tbest));
        UpdatePheromone(BN_l(tbest));
        if Q(tbest) > Q(gbest) then
            BN_l(gbest) ← BN_l(tbest);
            Q(gbest) ← Q(tbest);
        end if
        t ← t + 1;
    until t = max_iterations or Convergence()
    append BN_l(gbest) to BMC;
end for
return BMC;
End
```

In essence, each $ant(i)$ in the colony creates a candidate solution $BN_l(i)$, i.e., a local BN, for a given subset D_l. Then the quality of the constructed solution is evaluated. The best solution $BN_l(tbest)$ produced in the colony

at iteration t is selected to undergo local search before the ant updates the pheromone trail according to the quality of its solution $Q(tbest)$. After that, we compare the iteration best solution $BN_l(tbest)$ with the global best solution $BN_l(gbest)$ to keep track of the best solution found so far. This set of steps is considered an iteration of the $repeat - until$ loop and is repeated until the same solution is generated for a number of consecutive trials specified by the `conv_iterations` parameter (used by $Convergence()$ function in Algorithm 2 to test for convergence) or until `max_iterations` is reached. When this loop terminates, $BN_l(gbest)$ is considered the local Bayesian network BN_l for class l, and is appended to the Bayesian multi-net classifier BMC. This process is repeated to build a local BN for each class value $l \in C$ to produce a complete BMN classifier.

4.2 Local Bayesian Network Construction

In order to create a candidate local BN, each ant starts with an empty BN, i.e., an edge-less network structure. As shown in Algorithm 3, which shows the pseudo-code of the BN creation procedure, the ant starts to add edges (representing variable dependencies) to construct a *general* Bayesian network. Unlike ABC-Miner, which starts with a Naïve-Bayes structure to construct a BAN classifier, the purpose of this procedure is to construct a local BN to be a part of the overall produced BMN classifier. The selection of the edges is performed according to the probabilistic state transition formula presented in Equation 5. After the ant adds a valid edge to the local BN, all the invalid edges are eliminated from the construction graph. The ant keeps adding edges to the current solution until no valid edges are available. When the structure of a candidate local BN_l is completely constructed by an ant, the CPTs are learnt for the network using the data subset D_l. Hence, all the parameters of BN_l are conditioned on the class value l, which makes a local BN in the BMN class dependent.

Algorithm 3 Pseudo-code of local BN creation procedure.

Begin CreateBN
$BN_l(i) \leftarrow \phi$;
$D_l \leftarrow\ INPUT$;
$k\_list \leftarrow ant(i).SelectMaxParentsForEachVariable()$;
while $GetValidEdges() \neq \phi$ **do**
    $\{i \rightarrow j\} \leftarrow ant_i.SelectEdgeProbablistically()$;
    $BN_l(i) \leftarrow BN_l(i) \cup \{i \rightarrow j\}$;
    $RemoveInvalidEdges(BN_l(i), k_j)$;
end while
$BN_l(i).LearnParameters(D_l)$;
return $BN_l(i)$;
End

4.3 Evaluating the Quality of a Candidate Local BN

Since the target of the algorithm is to build a classifier, the scoring function, which the algorithm tries to maximize, should be related to the classification effectiveness of the produced model. This reasoning was behind selecting classification accuracy as the scoring function by which a candidate solution is evaluated in ABC-Miner.

However, we should recall that each ant in the ABC-Miner algorithm creates a complete BN classifier, which can be evaluated directly using that scoring function. On the other hand, since ABC-Miner$_l^{mn}$ learns each local BN in the multi-net separately, classification accuracy cannot be used as a scoring function, because each ant builds a *general* Bayesian network BN_l (not a classifier) for just a subset D_l of the dataset $\mathbf{D}$ for class value l. Consequently, the constructed local BN_l cannot be evaluated as a classifier until all the local BNs are learnt and the complete multi-net is composed. Hence, a candidate solution (local BN) built by an ant must be evaluated as a *general* Bayesian network.

One of the most used scoring metrics for building and evaluating Bayesian networks is K2, a metric based on uniform prior scoring [26]. It is used by Algorithm-B [29] and ACO-B [7], which are a greedy and an ant-based algorithm, respectively, aiming to build conventional Bayesian networks. However, ABC-Miner$_l^{mn}$ does not aim to build a BN as a final product; it builds a local BN to serve as a part of a classifier. On the other hand, some discriminative scoring functions, such as Kullback distance for cross-class divergence, have been used to learn BMNs [30]. However, in such cases, the algorithm starts with a *complete* initial structure for all the local BN in the BMN (unlike the current local approach), and incrementally modifies this structure using a greedy search. This complete structure can be evaluated as a classifier using a discriminative objective function.

Accordingly, the target is to learn a local BN_l that not only maximizes the marginal likelihood $P(D_l|BN_l)$ of the data subset D_l, but also minimizes the marginal likelihood $P(\mathbf{D} - D_l|BN_l)$ of the other part of the training set. This aims to increase the discriminative power of the local BNs with respect to the class values, so that an instance x_l labeled by class value l would have a high probability in Bayesian network BN_l and a low probability in other local BNs when the multi-net is used as a classifier. Therefore, we propose a scoring function – by which a candidate local BN is evaluated – based on maximizing the difference between average marginal log-likelihood of positive and negative instances of the whole training set, given the local BN currently constructed. The function is formulated as follows:

$$Q(BN_l) = \frac{\sum_{i=1}^{n} LL(x_i^+|BN_l)}{|x^+|} - \frac{\sum_{j=1}^{m} LL(x_j^-|BN_l)}{|x^-|} \tag{8}$$

where $LL(x|BN)$ is the log-likelihood of the instance x given BN, x^+ are the instances in the training set labeled by class value l – whose local BN is currently being constructed by the algorithm, x^- are the instances in the training set not labeled by class value l, $|x^+|$ is the number of instances labeled by l, $|x^-|$ is the

number of instances not labeled by l. Although only the subset D_l is used to learn the CPT of a local BN_l (as shown in Algorithm 3), the whole training set $\mathbf{D}$ is used to evaluated the quality of the constructed local BN (as shown in Algorithm 2) to perform the discrimination between instances belonging and not belonging to class l. This is performed by calculating the difference between the average of the (log)likelihood of the instances belonging to the current class and the instances belonging to the other classes (as shown in Equation 8), and the optimization objective is to maximize this difference.

5 A Global ACO Approach for Learning BMN Classifiers

5.1 The ABC-Miner$_g^{mn}$ Algorithm Outline

The global approach algorithm works in a different way from its local counterpart. A single run of the Ant Colony Optimization meta-heuristic is utilized to build the whole solution; each ant builds a *complete* Bayesian multi-net (BMN) classifier as a candidate solution at once. This is accomplished by building a candidate local BN_l for each class value $l \in \mathbf{C}$ in each single ant trial, appending them to the BMN classifier, then evaluating the resulting final BMN classifier as a whole. Unlike ABC-Miner$_l^{mn}$, which has to completely finish building the local Bayesian network BN_l before starting to build BN_{l+1}, in ABC-Miner$_g^{mn}$ each ant builds the whole BMN classifier BMC (including all the local BNs) before performing evaluation and pheromone update.

As shown in Algorithm 4, each $ant(i)$ in the colony creates a candidate solution $BMC(i)$, i.e. a complete BMN classifier. Then the quality of the constructed BMN is evaluated. The best solution $BMC(tbest)$ produced in the colony at iteration t is selected to undergo local search before the ant updates the pheromone trail according to the quality of its solution $Q(tbest)$. After that, we compare the iteration's best solution $BMC(tbest)$ with the global best solution $BMC(gbest)$ to keep track of the best solution found so far. This is iteratively repeated until the termination conditions are met. At the end, $BMC(gbest)$ is considered the final solution.

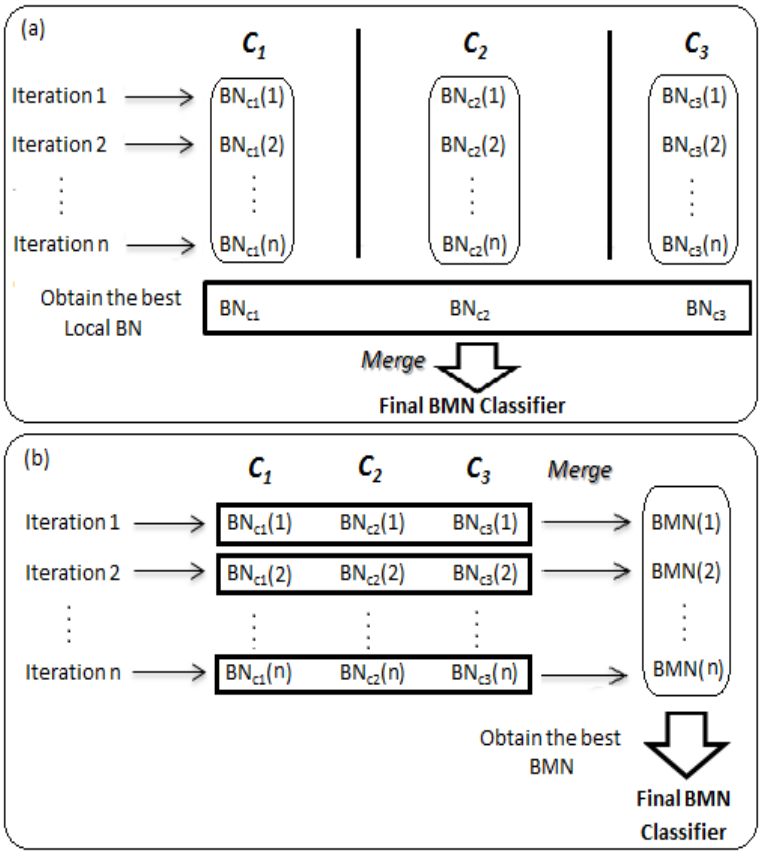
In order to apply such a global approach using ACO, we use several construction graphs for a given dataset. More precisely, we use $|C|$ construction graphs, one for each class value $l \in C$. Pheromone amounts and heuristic information (using mutual information) initialization are performed on all the used construction graphs. As for solution creation, an ant uses each construction graph to build each local BN, one for each class value. Once an ant creates the structure of the local BN_l and learns its parameters, it proceeds to the construction of the local BN_{l+1}, using the $(l+1)th$ construction graph and the D_{l+1} subset of the data for parameter learning. The ant proceeds until it constructs a local BN for all the class values, to compose a complete BMN classifier. At this point, the quality of the complete solution can be evaluated, and the pheromone is updated on all construction graphs according to the quality of the

Algorithm 4 Pseudo-code of ABC-Miner$_g^{mn}$.

```
Begin
D ← training_set; t ← 1;
BMC(gbest) ← φ; Q(gbest) ← 0;
InitializePheromoneAmounts();
InitializeHeuristicValues();
repeat
    BMC(tbest) ← φ; Q(tbest) ← 0;
    for i = 1 to colony_size do
        BMC(i) ← φ;
        for l = 1 to |C| do
            D_l ← subset(D, l);
            BN_l(i) ← ant(i).CreateSolution(D_l);
            append BN_l(i) to BMC(i);
        end for
        Q(i) ← ComputeQuality(BMC(i), D);
        if Q(i) > Q(tbest) then
            BMC(tbest) ← BMC(i);
            Q(tbest) ← Q(i);
        end if
    end for
    PerformLocalSearch(BMC(tbest));
    UpdatePheromone(BMC(tbest));
    if Q(tbest) > Q(gbest) then
        BMC(gbest) ← BMC(tbest);
        Q(gbest) ← Q(tbest);
    end if t ← t + 1;
until t = max_iterations or Convergence()
return BMC(gbest);
End
```

constructed BMN classifier. Then a subsequent ant can perform a new solution construction trial. Figure 3 shows a diagram that compares the local and the global approaches for building a BMN classifier.

Figure 3: The two proposed ACO approaches for building a BMN classifier for a 3-class dataset: (a) the local approach, where each local BN is constructed at a time and the best local BN constructed over n iterations for each class are merged at the end to compose the final BMN, and (b) the global approach, where a complete candidate BMN is constructed at each iteration and the best BMN constructed over n iterations is the final solution. The diagram is simplified by assuming that the size of the colony is 1; only one solution is constructed at each iteration.



5.2 Evaluating the Quality of a Candidate BMN

The quality of a candidate BMN classifier is evaluated directly (as a classifier) using the accuracy measure, as a conventional measure of classification performance, which is computed as follows:

$$Accuracy = \frac{|Correctly\_Classified\_Cases|}{|Training\_Set|} \tag{9}$$

Note that the use of classification accuracy as a solution quality evaluation function is valid in ABC-Miner$_g^{mn}$, similar to ABC-Miner, since in both algorithms the constructed candidate solution represents a classifier, which can be evaluated using any measure of *classification* performance. This is in contrast to the case in ABC-Miner$_l^{mn}$, which builds a general Bayesian network as a candidate solution for a local BN in the BMN classifier.

6 Experimental Methodology

6.1 Comparative Evaluations

We compare the predictive accuracy of our proposed ant-based algorithms for learning BMN classifiers with our previously introduced ABC-Miner that learns BAN classifiers, as well as three other widely used Bayesian classifiers, namely Naïve-Bayes, Tree Augmented Naïve-Bayes (TAN) and General Bayesian Network (GBN). TAN allows a node in a BN to have one parent, besides the class variable. This produces a tree-like structure BN. Unlike the other BN classifier learners, a GBN treats the class variable as an ordinary node; it builds a general BN, finds the Markov blanket of the class node and uses it as a Bayesian classifier [2].

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 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 [29] 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 classifier.

As for BMN algorithms, we compared our algorithms to Chow-Liu tree Multi-net (CL-Tree MN) [4]. This algorithm builds a tree-like network structure, one for each class value, using the mutual information between the variables, by finding the maximum weighted spanning tree as in CL-tree algorithm [4]. In addition, we implemented BMN-K2, which uses a greedy hill-climbing (GHC)

approach to learn local Bayesian networks for the BMN classifiers. That is, for each local BN in the BMN, the algorithm starts with an empty (edge-less) network. Then the algorithm searches for the edge that leads to the maximum increase in the quality of the BN classifier being constructed according to the K2 scoring function. Table 1 presents the main properties of the used Bayesian 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
CL-Tree MN	Determ.	Max. Spanning Tree.	BMN	Mut. Info.
BMN-K2	Determ.	Greedy Hill Clim.	BMN	K2 Function
ABC-Miner	Stoch.	Ant Colony Optim.	BAN	Predictive Acc.
ABC-Miner$_l^{mn}$	Stoch.	Ant Colony Optim.	BMN	Likelihood Diff.
ABC-Miner$_g^{mn}$	Stoch.	Ant Colony Optim.	BMN	Predictive Acc.

Besides the Bayesian algorithms, we compare our proposed ACO algorithms with three well-known classification algorithms: JRip, J48 (the WEKA tool's implementation of Ripper and C4.5, respectively), and SVM [42]. JRip is a classification rule induction algorithm, which learns a classification model that consists of a list of classification rules. J48 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 cases into a higher-dimensional feature space and then finds the best hyperplane for separating cases of different classes – where the best hyperplane is the one with the greatest possible margin (a gap separating cases of different classes in the data space). SVMs has the disadvantage of producing "black-box" models that can hardly be interpreted by users [42, 43].

6.2 Experiment Setup

The experiments were carried out using stratified 10-fold cross validation [42]. 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 3. Note that the `max_iterations` parameter refers to the maximum number of iterations used to build a single local BN in the ABC-Miner$_l^{mn}$ algorithm. The

Table 2: Parameter settings of ABC-Miner^{mn} used in experiments.

Parameter	Value
`max_iterations`	100
`colony_size`	10
`conv_iterations`	15
`max_parents`	3

value of this parameter is multiplied by the number of class values when used with ABC-Miner$_g^{mn}$. On the other hand, for the BMN-K2 greedy 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 single local BN. 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 BN classifier. 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. Note that, the results for ABC-Miner in the current experiments are slightly different than its results in [17] because here we used different parameter settings and different training/testing set folds.

Unlike our ant-based algorithms, the number of parents (k-dependencies) must be specified for Algorithm-B and BMN-K2. 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 4.3). In our experiments, we used WEKA [42] implementations for Rippper (JRip), J48 (J48), and SVM (SMO), with its default parameter settings.

6.3 Datasets

The performance of ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$ was evaluated using 33 public-domain datasets from the University of California at Irvine UCI dataset repository [44]. The main characteristics of the datasets, such as number of cases, number of predictor attributes and number of classes are shown in Table 3. For the BN classification algorithms, datasets having continuous attributes were discretized in a pre-processing step, using the well-known J48-Disc algorithm [42], applied to the training sets. For the other algorithms, the original datasets with real-valued attributes were used.

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

7 Computational Results and Analysis

7.1 Predictive Accuracy Results

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.

According to the results in Table 4, our proposed ACO-based algorithm for learning BMNs using a global approach, ABC-Miner$_g^{mn}$, obtained the best overall rank in terms of predictive accuracy of 2.9, and achieved the best predictive results in 13 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. Our proposed ACO-based local approach for learning BMNs, ABC-Miner$_l^{mn}$, came in the third place with overall rank of 3.6, and achieved the best predictive results in 6 datasets. J48 and JRip came in fourth and the fifth place, respectively, by obtaining 5.2 and 6.5 overall rank, respectively, achieved the best result in 5 and 7 datasets, respectively.

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

Dataset	Naïve-B	Cl-tree	Algo-B	Cl-tree MN	BMN-K2	ABC-Miner	ABC_l^{mn}	ABC_g^{mn}	JRip	J48	SVM
abl	86.1 ± 0.6	87.2 ± 0.6	88.3 ± 0.2	81.2 ± 0.2	88.3 ± 0.2	85.6 ± 0.2	89.0 ± 0.2	89.1 ± 0.2	**92.6 ± 0.9**	**92.6 ± 0.6**	89.1 ± 0.6
bal	**91.2 ± 0.8**	84.4 ± 0.9	85.6 ± 0.6	84.7 ± 0.7	82.1 ± 0.9	84.7 ± 0.7	84.7 ± 0.7	85.6 ± 0.6	77.6 ± 1.8	80.2 ± 1.2	88.4 ± 0.9
bcw	92.1 ± 0.9	95.4 ± 0.9	93.8 ± 0.9	95.7 ± 0.9	95.7 ± 0.9	95.4 ± 0.9	97.2 ± 0.6	97.5 ± 0.9	94.7 ± 2.6	95.1 ± 1.2	**97.7 ± 0.9**
car	85.3 ± 0.9	93.6 ± 0.6	86.2 ± 0.9	90.6 ± 0.6	92.8 ± 0.6	97.2 ± 0.3	**98.6 ± 0.3**	**98.6 ± 0.3**	90.6 ± 2.6	94.3 ± 3.1	94.1 ± 0.9
chess	88.2 ± 1.2	92.5 ± 1.2	89.8 ± 0.9	93.1 ± 0.9	92.5 ± 0.9	97.6 ± 0.9	95.5 ± 0.6	**98.4 ± 0.6**	98.1 ± 0.8	**98.4 ± 3.1**	95.1 ± 2.8
cmc	52.2 ± 1.2	56.4 ± 1.2	54.6 ± 1.2	57.9 ± 0.6	58.2 ± 0.6	66.4 ± 0.6	66.4 ± 0.9	**66.7 ± 0.9**	60.1 ± 0.3	64.2 ± 1.2	60.2 ± 2.2
crd-a	77.5 ± 1.2	85.1 ± 0.9	85.7 ± 0.9	86.8 ± 0.6	86.1 ± 0.6	86.8 ± 0.9	88.4 ± 0.9	90.6 ± 1.2	85.7 ± 0.3	**91.9 ± 0.3**	85.5 ± 1.6
crd-g	75.6 ± 0.9	73.7 ± 1.2	75.6 ± 1.2	74.2 ± 0.9	73.2 ± 0.9	73.7 ± 0.9	77.6 ± 0.9	**79.5 ± 0.6**	73.2 ± 2.4	75.7 ± 0.6	75.2 ± 0.6
drm	96.2 ± 0.6	97.8 ± 0.9	97.2 ± 0.9	97.2 ± 0.6	97.2 ± 0.6	**98.6 ± 0.4**	**98.6 ± 0.4**	**98.6 ± 0.4**	94.8 ± 0.6	96.1 ± 2.2	97.2 ± 0.8
ecoli	86.5 ± 1.2	84.2 ± 1.2	82.6 ± 1.4	85.0 ± 1.2	85.2 ± 0.9	87.4 ± 0.6	87.4 ± 0.6	87.5 ± 0.4	84.3 ± 0.4	85.7 ± 0.9	**87.6 ± 2.2**
glass	74.2 ± 1.2	78.4 ± 0.9	80.7 ± 0.9	80.7 ± 0.9	79.3 ± 0.9	82.6 ± 1.2	**83.1 ± 0.9**	82.4 ± 1.2	75.7 ± 0.9	82.2 ± 0.6	82.6 ± 0.9
hay	80.0 ± 2.8	77.9 ± 3.1	83.1 ± 3.5	83.3 ± 1.4	82.0 ± 2.8	80.2 ± 3.1	83.6 ± 2.6	82.6 ± 2.6	**84.2 ± 2.6**	83.3 ± 0.9	81.6 ± 0.2
hrt-c	62.7 ± 2.2	68.8 ± 2.5	66.1 ± 2.2	70.5 ± 0.9	68.8 ± 2.2	73.8 ± 3.1	74.7 ± 2.6	**77.5 ± 2.2**	57.1 ± 2.6	70.9 ± 0.6	72.6 ± 0.3
hrt-s	83.4 ± 2.2	83.3 ± 1.8	81.5 ± 1.6	84.2 ± 3.5	83.6 ± 1.6	85.6 ± 1.2	86.8 ± 0.8	**88.1 ± 1.6**	76.6 ± 0.8	75.5 ± 0.9	84.1 ± 2.4
hep	71.9 ± 1.6	**78.0 ± 1.6**	67.3 ± 2.1	77.8 ± 1.2	72.7 ± 1.4	77.8 ± 1.2	**78.0 ± 1.6**	77.8 ± 1.2	72.7 ± 1.4	68.9 ± 1.2	74.2 ± 1.6
iono	82.6 ± 0.6	90.7 ± 0.9	90.7 ± 0.6	92.1 ± 2.2	92.1 ± 0.6	94.5 ± 0.3	94.3 ± 0.3	**95.6 ± 0.6**	92.8 ± 0.3	93.1 ± 0.6	94.3 ± 0.6
iris	96.2 ± 1.5	94.2 ± 1.8	92.9 ± 0.8	94.8 ± 2.2	94.2 ± 0.8	96.2 ± 0.6	96.2 ± 0.3	96.2 ± 0.3	95.6 ± 0.3	95.0 ± 1.2	**96.6 ± 0.4**
lung	72.5 ± 2.8	75.4 ± 2.2	63.2 ± 2.8	70.1 ± 1.6	66.6 ± 2.2	74.8 ± 2.2	75.4 ± 2.4	**75.7 ± 2.4**	58.7 ± 2.4	**75.7 ± 0.6**	69.3 ± 0.8
monk	61.6 ± 0.6	58.8 ± 0.6	61.6 ± 0.9	60.9 ± 0.6	62.7 ± 0.9	61.8 ± 0.9	62.9 ± 0.6	63.1 ± 0.6	61.4 ± 0.6	62.2 ± 0.6	**63.3 ± 0.9**
mush	95.8 ± 0.4	98.2 ± 0.6	96.1 ± 0.9	98.4 ± 0.8	98.2 ± 0.2	98.8 ± 0.6	99.8 ± 0.4	99.8 ± 0.9	**100. ± 0.4**	**100. ± 0.8**	**100. ± 0.6**
nurs	90.1 ± 0.9	94.3 ± 0.9	92.7 ± 1.2	95.1 ± 2.2	94.2 ± 0.6	98.0 ± 0.9	**98.9 ± 0.8**	**98.9 ± 0.8**	96.5 ± 0.8	97.1 ± 0.6	93.2 ± 0.9
park	84.5 ± 2.5	91.7 ± 1.8	84.5 ± 2.1	92.1 ± 1.9	88.9 ± 1.8	95.8 ± 1.6	96.1 ± 1.6	96.4 ± 2.1	**98.5 ± 1.4**	96.2 ± 1.6	98.2 ± 1.5
pbc	93.5 ± 0.9	96.1 ± 0.6	95.6 ± 0.6	96.1 ± 0.9	95.9 ± 0.8	96.5 ± 0.6	96.8 ± 0.6	97.6 ± 0.6	97.4 ± 0.4	98.2 ± 0.6	**98.7 ± 0.6**
pima	75.4 ± 1.2	77.8 ± 1.5	76.2 ± 1.5	76.5 ± 1.2	78.9 ± 1.9	**78.9 ± 1.9**	77.6 ± 1.4	77.6 ± 1.4	**79.9 ± 1.9**	79.5 ± 1.6	75.7 ± 1.2
pop	68.2 ± 0.6	64.1 ± 0.6	66.6 ± 2.5	65.5 ± 0.2	69.8 ± 0.9	74.3 ± 0.9	73.1 ± 0.6	**76.9 ± 0.4**	71.1 ± 0.9	71.1 ± 0.6	72.2 ± 0.9
seg	91.6 ± 0.8	94.8 ± 0.6	94.1 ± 0.8	95.0 ± 0.6	95.5 ± 0.4	95.5 ± 0.6	95.5 ± 0.6	96.3 ± 0.8	94.7 ± 0.9	96.3 ± 0.9	**97.8 ± 0.6**
soy	91.4 ± 1.2	95.6 ± 1.2	93.2 ± 0.6	95.6 ± 0.6	95.2 ± 1.2	95.6 ± 1.2	95.6 ± 0.9	96.4 ± 1.2	91.5 ± 0.6	95.1 ± 0.8	**97.6 ± 1.2**
SPECT	73.7 ± 0.8	72.1 ± 0.9	74.0 ± 1.2	75.5 ± 1.2	75.5 ± 1.2	79.5 ± 0.8	78.1 ± 0.8	**80.9 ± 0.6**	76.7 ± 0.9	74.4 ± 0.6	78.5 ± 0.8
ttt	70.3 ± 0.3	76.6 ± 0.3	74.3 ± 0.9	78.1 ± 0.9	78.4 ± 0.6	86.4 ± 0.3	90.4 ± 0.6	91.2 ± 0.3	**97.8 ± 0.6**	85.7 ± 1.2	96.3 ± 0.3
vot	90.3 ± 0.9	92.1 ± 0.9	90.3 ± 0.6	92.8 ± 0.6	94.1 ± 1.2	94.6 ± 1.2	94.7 ± 1.2	95.8 ± 1.4	**96.5 ± 0.9**	96.3 ± 0.9	95.8 ± 1.2
wine	95.6 ± 1.2	97.3 ± 0.9	97.3 ± 0.6	97.3 ± 0.6	97.3 ± 0.9	98.4 ± 1.5	98.5 ± 0.9	98.5 ± 0.9	97.6 ± 0.6	95.8 ± 1.2	**98.9 ± 1.5**
yeast	59.7 ± 1.5	61.2 ± 1.2	60.2 ± 0.9	62.6 ± 0.9	62.1 ± 1.2	62.6 ± 0.8	63.2 ± 1.2	63.8 ± 0.9	63.8 ± 1.2	64.2 ± 1.2	**64.6 ± 0.8**
zoo	94.2 ± 0.6	97.4 ± 0.3	95.1 ± 0.6	97.4 ± 0.6	97.4 ± 0.3	**98.0 ± 0.1**	**98.0 ± 0.1**	**98.0 ± 0.1**	94.2 ± 0.8	96.1 ± 0.8	97.5 ± 0.1
Rank	9.2	7.6	8.7	6.8	7.1	4.6	3.6	2.9	6.5	5.2	3.2

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 [45, 46], which is used to evaluate multiple algorithms on multiple datasets [47], with respect to our two proposed algorithms: ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$. We performed the Friedman test using the freely available Java program suggested by Garcia et al. in [46], 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 a result is significant at 5% level, and a single underlined value indicates that a result is significant at 10% level.

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

Algorithm	ABC-Miner$_l^{mn}$	ABC-Miner$_g^{mn}$
Naïve-Bayes	5.4E-11	1.3E-14
CL-Tree	2.13E-6	3.3E-9
Algorithm-B	4.8E-9	2.4E-12
CL-Tree MN	0.0011	7.06E-6
BMN-K2	3.3E-4	1.59E-6
ABC-Miner	1.926	0.0958
JRip	0.0016	1.1E-5
J48	0.0795	0.0493
SVM	2.999	1.0044

7.2 Model Simplicity Results

Table 6 reports the model size results, in terms of the number of edges, of the models produced by the Bayesian classification algorithms, as the average of the 10-fold cross validation experiments. Note that the results reported for the BMN learning algorithms represent the average number of edges for a single local BN in the constructed BMN for a given dataset. The number of local BNs produced equals the number of the classes, which is shown in the last column of Table 6 for each dataset. Note also that for GBN, we report the class Markov blanket size, whilst for TAN and ABC-Miner we report the number of edges only between the input variables in the network, without counting the edges between the class and the input variables. The last row in the table shows the average result for the model sizes over all the datasets.

As shown in Table 6, TAN, as expected, produce the smallest BN models in general, since the number of parents in a TAN is restricted to one (besides the class parent whose edges are not counted), thus a TAN structure contains n-1 edges, where n is the number of the attributes in the dataset. This is

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

Dataset	TAN	GBN	TAN-MN	BMN-K2	ABC-Miner	ABC_l^{mn}	ABC_g^{mn}	classes
abl	7.0 ± 2.3	18.9 ± 3.5	7.0 ± 2.2	15.4 ± 1.5	19.5 ± 3.6	16.4 ± 2.6	16.8 ± 3.5	29
bal	4.0 ± 2.3	3.7 ± 2.5	3.2 ± 1.1	2.4 ± 3.2	2.8 ± 1.3	2.4 ± 3.6	2.1 ± 3.9	3
bcw	8.0 ± 2.1	18.7 ± 2.7	8.0 ± 3.2	24.5 ± 2.3	31.0 ± 1.8	22.6 ± 3.4	16.1 ± 2.5	2
car	5.0 ± 1.5	13.6 ± 3.5	5.0 ± 3.6	6.7 ± 3.2	11.5 ± 1.1	9.8 ± 1.1	5.5 ± 3.2	4
chess	35.0 ± 3.2	63.4 ± 2.6	35.0 ± 2.5	51.8 ± 2.3	69.7 ± 3.4	55.2 ± 2.7	47.5 ± 2.5	2
cmc	8.0 ± 3.5	20.6 ± 2.4	8.0 ± 3.5	19.8 ± 3.7	21.9 ± 1.3	17.8 ± 3.8	19.3 ± 1.2	3
crd-a	13.0 ± 3.8	17.4 ± 2.3	13.0 ± 3.2	24.4 ± 2.4	18.6 ± 1.5	22.5 ± 2.4	16.8 ± 1.1	2
crd-g	19.0 ± 3.8	34.6 ± 1.4	19.0 ± 2.5	38.5 ± 1.3	39.8 ± 1.7	36.7 ± 2.2	32.2 ± 3.4	2
drm	32.0 ± 1.4	22.5 ± 3.5	32.0 ± 1.8	22.1 ± 1.7	25.4 ± 3.1	17.6 ± 1.1	21.8 ± 3.4	6
ecoli	7.0 ± 2.3	5.6 ± 1.1	7.0 ± 1.8	8.5 ± 2.4	7.6 ± 3.3	8.1 ± 3.2	6.1 ± 3.3	8
glass	9.0 ± 2.8	8.2 ± 1.7	9.0 ± 1.2	7.8 ± 2.7	8.5 ± 3.4	6.9 ± 2.4	6.4 ± 1.1	7
hay	3.0 ± 2.3	2.8 ± 2.5	3.0 ± 1.1	1.9 ± 3.2	2.5 ± 1.3	2.2 ± 3.6	1.8 ± 3.8	3
hrt-c	11.0 ± 2.3	21.8 ± 3.8	11.0 ± 1.6	20.9 ± 3.3	22.5 ± 3.3	21.9 ± 1.5	17.8 ± 2.7	3
hrt-s	12.0 ± 3.2	22.6 ± 3.3	12.0 ± 3.1	18.9 ± 2.3	24.4 ± 1.2	21.8 ± 2.1	15.5 ± 2.7	2
hep	15.0 ± 2.3	24.8 ± 1.2	15.0 ± 3.6	26.8 ± 2.7	26.9 ± 1.6	23.4 ± 1.4	21.2 ± 3.2	2
iono	33.0 ± 2.7	31.2 ± 3.8	33.0 ± 1.4	32.7 ± 2.5	41.8 ± 2.3	29.6 ± 3.7	31.3 ± 3.1	2
iris	3.0 ± 1.8	3.2 ± 2.7	3.0 ± 1.6	1.9 ± 3.4	1.9 ± 3.4	1.7 ± 3.1	0.8 ± 1.7	3
lung	55.0 ± 2.3	38.6 ± 1.5	55.0 ± 3.2	20.8 ± 2.5	41.5 ± 2.2	22.6 ± 3.3	18.8 ± 1.7	3
monk	5.0 ± 2.4	15.8 ± 3.5	5.0 ± 1.2	12.8 ± 2.6	11.2 ± 3.4	6.8 ± 1.7	10.4 ± 3.5	2
mush	21.0 ± 2.7	29.7 ± 2.5	21.0 ± 3.1	22.4 ± 2.1	36.4 ± 2.1	28.7 ± 1.1	19.6 ± 3.5	2
nurs	7.0 ± 1.4	10.5 ± 2.6	7.0 ± 3.2	11.9 ± 3.3	10.2 ± 3.5	8.6 ± 3.4	6.2 ± 3.5	5
park	22.0 ± 2.1	9.4 ± 3.4	22.0 ± 2.7	13.2 ± 2.3	8.8 ± 1.7	12.5 ± 1.2	9.1 ± 3.5	2
pbc	9.0 ± 1.4	25.6 ± 3.2	9.0 ± 3.7	20.5 ± 3.3	24.9 ± 1.1	18.2 ± 3.3	22.5 ± 2.8	5
pima	8.0 ± 1.4	14.2 ± 2.2	8.0 ± 1.8	18.6 ± 1.3	10.5 ± 1.8	15.7 ± 2.4	8.9 ± 2.5	2
pop	7.0 ± 1.2	13.4 ± 2.7	7.0 ± 3.7	15.9 ± 2.3	11.7 ± 1.3	14.7 ± 3.2	7.8 ± 1.5	3
seg	18.0 ± 3.3	28.8 ± 1.4	18.0 ± 3.6	30.8 ± 3.7	32.9 ± 1.4	28.4 ± 1.4	26.2 ± 3.2	7
soy	34.0 ± 3.1	21.7 ± 1.6	34.0 ± 1.8	21.9 ± 2.4	18.4 ± 1.4	20.8 ± 1.6	14.6 ± 1.6	19
SPECT	21.0 ± 2.6	19.6 ± 2.1	21.0 ± 3.6	22.8 ± 1.4	21.1 ± 1.5	20.4 ± 2.3	18.8 ± 2.8	2
ttt	8.0 ± 2.5	20.2 ± 2.1	8.0 ± 2.6	19.8 ± 3.6	18.7 ± 3.7	16.5 ± 2.3	19.2 ± 2.3	2
vot	15.0 ± 1.7	25.5 ± 3.2	15.0 ± 2.7	24.1 ± 1.1	27.3 ± 3.5	24.6 ± 2.6	22.7 ± 2.3	2
wine	12.0 ± 1.2	7.3 ± 1.4	12.0 ± 1.1	5.8 ± 3.8	6.8 ± 3.7	6.1 ± 2.1	4.5 ± 1.7	3
yeast	7.0 ± 3.1	8.6 ± 3.6	7.0 ± 1.1	10.5 ± 1.5	10.7 ± 1.4	8.4 ± 1.1	8.2 ± 1.3	10
zoo	16.0 ± 3.3	2.5 ± 1.4	16.0 ± 1.3	2.1 ± 1.2	2.4 ± 3.7	2.1 ± 2.7	1.7 ± 2.3	7
Avg.	15.4	19.41	15.4	18.3	20.9	17.6	15.5	

also shown in CL-Tree MN, which produces a local tree structure for each class value. The reason why in some cases the ACO-based algorithm produces smaller models is due to the local search procedure that might remove edges, so that the number of the edges left is less than n-1. We can also see that, overall, each local BN produced by ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$ is smaller than the BAN models produced by ABC-Miner and GBN. However, note that the former two algorithms produce several local BNs, as many as the class values in the dataset, whilst the latter two algorithms produce only one model. On the other hand, comparing our proposed ACO algorithms to the BMN-K2 that also learns BMNs, we see that, overall, the ACO algorithms construct smaller local BNs.

7.3 Computational Time Results

Running time results are shown in Table 7. The (10-folds average) 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 corresponding column with "ratio" header. The last row in the table reports the average ratio of the running time of each algorithm to ABC-Miner across all the datasets.

As shown in Table 7, the ACO ABC-Miner$_l^{mn}$ algorithm took less computational time, overall, compared to ABC-Miner, since it achieved on average about 79% of the running time of ABC-Miner. The global approach ABC-Miner$_g^{mn}$ algorithm, however, took about the double of the computational time of the ABC-Miner algorithm. BMN-K2, which uses a greedy hill-climbing (GHC) approach to build a BMN, took less computational time compared to Algorithm-B, which also uses GHC but builds a single BN. CL-tree MN, which learns local tree structures in the BMN, obtained the best running time results.

7.4 Discussion

According to the predictive accuracy results (shown in Table 4), algorithms that learn BMN classifiers outperform the corresponding algorithms that learn BN classifiers. This can be observed when comparing three pairs of algorithms. First, our proposed ACO algorithms, ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$, which learn BMN classifiers, obtained overall better results than their corresponding version, ABC-Miner, which learns BAN classifiers. Second, CL-tree MN that learns BMNs with local tree structures obtained overall better results than CL-tree that learns a single TAN classifier. Third, BMN-K2, which uses a greedy search to maximize the K2 scoring function in building BMNs, obtained overall better results than Algorithm-B, which also uses a greedy search to maximize the same scoring function, yet to build a single GBN.

As for the results of the statistical significance tests with respect to the predictive accuracy (shown in Table 5), both of our proposed algorithms are statistically better than the other conventional Bayesian classification algorithm with a significance level of 5%. ABC-Miner$_g^{mn}$, which utilizes the global approach in

Table 7: Running time (in seconds) results.

Dataset	CL-Tree MN		Algo-B		BMN-K2		ABC_l^{mn}		ABC_g^{mn}		ABC
	time	ratio	time	ratio	time	ratio	time	ratio	time	ratio	time
abl	110	**0.01**	2200	**0.12**	1700	**0.09**	21000	**1.17**	39000	**2.17**	18000
bal	20	**0.01**	2105	**0.96**	2300	**1.05**	2700	**1.23**	3200	**1.45**	2200
bcw	35	**0.02**	2360	**1.18**	1300	**0.65**	1700	**0.85**	4800	**2.4**	2000
car	30	**0.03**	700	**0.7**	850	**0.85**	800	**0.8**	4300	**4.3**	1000
chess	590	**0.02**	8410	**0.23**	6900	**0.19**	24000	**0.65**	30000	**0.81**	37000
cmc	75	**0.01**	2100	**0.39**	2200	**0.41**	3100	**0.57**	16400	**3.04**	5400
crd-a	20	**0.01**	2105	**0.96**	2300	**1.05**	2700	**1.23**	3200	**1.45**	2200
crd-g	35	**0.01**	2365	**0.53**	1400	**0.31**	3700	**0.82**	9100	**2.02**	4500
drm	25	**0.01**	1041	**0.37**	1200	**0.43**	2600	**0.93**	7500	**2.68**	2800
ecoli	10	**0.01**	100	**0.07**	130	**0.09**	1000	**0.67**	2100	**1.4**	1500
glass	10	$<$ **0.01**	789	**0.16**	500	**0.1**	4500	**0.9**	8200	**1.64**	5000
hay	5	**0.02**	20	**0.07**	15	**0.05**	200	**0.67**	500	**1.67**	300
hrt-c	20	$<$ **0.01**	260	**0.06**	240	**0.06**	4300	**1.02**	9500	**2.26**	4200
hrt-s	20	$<$ **0.01**	385	**0.09**	330	**0.07**	3600	**0.8**	8000	**1.78**	4500
hep	30	**0.12**	780	**3.12**	640	**2.56**	200	**0.8**	850	**3.4**	250
iono	15	**0.01**	1000	**0.37**	850	**0.31**	2200	**0.81**	7300	**2.7**	2700
iris	1	**0.01**	10	**0.1**	15	**0.15**	70	**0.7**	220	**2.2**	100
lung	1	**0.04**	20	**0.8**	5	**0.2**	15	**0.6**	35	**1.4**	25
monk	70	**0.02**	995	**0.23**	570	**0.13**	1200	**0.28**	14600	**3.4**	4300
mush	180	**0.01**	4040	**0.27**	9000	**0.6**	12000	**0.8**	28000	**1.87**	15000
nurs	90	**0.01**	5520	**0.65**	6500	**0.76**	7000	**0.82**	18200	**2.14**	8500
park	25	**0.01**	789	**0.38**	600	**0.29**	2500	**1.19**	3900	**1.86**	2100
pbc	850	**0.02**	3800	**0.11**	3700	**0.11**	30500	**0.87**	54000	**1.54**	35000
pima	20	**0.01**	2105	**0.96**	2300	**1.05**	2700	**1.23**	3200	**1.45**	2200
pop	1	$<$ **0.01**	25	**0.04**	65	**0.09**	500	**0.71**	600	**0.86**	700
seg	260	**0.01**	4100	**0.15**	5400	**0.19**	21000	**0.75**	55000	**1.96**	28000
soy	25	**0.02**	500	**0.38**	620	**0.48**	700	**0.54**	4100	**3.15**	1300
SPECT	20	$<$ **0.01**	4600	**0.96**	2360	**0.49**	4100	**0.85**	6500	**1.35**	4800
ttt	25	**0.01**	4290	**1.65**	4800	**1.85**	2900	**1.12**	5600	**2.15**	2600
vot	35	**0.01**	1841	**0.45**	1200	**0.29**	3200	**0.78**	6700	**1.63**	4100
wine	30	**0.12**	780	**3.12**	640	**2.56**	200	**0.8**	850	**3.4**	250
yeast	35	**0.01**	720	**0.24**	950	**0.32**	2200	**0.73**	6300	**2.1**	3000
zoo	10	**0.04**	105	**0.42**	120	**0.48**	100	**0.4**	350	**1.4**	250
Avg. Ratio		**0.02**		**0.51**		**0.46**		**0.79**		**2.09**	

learning BMNs, is statistically better than ABC-Miner with significance level of 10%. Both ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$ are statistically better than JRip with significance level of 5%. ABC-Miner$_l^{mn}$ is statistically better than J48 with significance level of 10% and ABC-Miner$_g^{mn}$ is statistically better than J48 with significance level of 5%.

Concerning the comparison of the results with Support Vector Machines (SVMs), broadly speaking, SVMs often obtain higher predictive accuracies than other types of classification algorithms, although this is not true for all types of datasets (as shown in Table 4). In our experiments, ABC-Miner$_g^{mn}$ obtained a better predictive accuracy rank than SVM, although the difference was not statistically significant (see Table 5). In addition, the classifier built by an SVM has the disadvantage of being in general a "black box" from the perspective of users - i.e., the output of an SVM algorithm can hardly be interpreted by users. By contrast, BN classifiers produce a graphical model of the dependencies between variables that can be directly interpreted by users, which is an advantage in many application domains. For a review of the importance of comprehensible classification models, see [43, 48].

Regarding the comprehensibility and the user interpretability of the constructed classification model, comparing the BMNs constructed by CL-Tree MN, BMN-K2, ABC-Miner$_l^{mn}$, and ABC-Miner$_g^{mn}$ to the BANs constructed by ABC-Miner is controversial. On one hand, a BAN is one single model that describes the whole domain, while a BMN consists of multiple local models, one for each class value, which can be argued to add overhead in the overall classifier interpretation. On the other hand, each local BN in a BMN is simpler and easier to interpret, compared to a single BAN. Moreover, having different class-based local BNs can enrich the analysis of the application domain by finding which attribute dependencies are common to all classes and which dependencies are specific to each class. Hence, more insight into the domain can be potentially gained by the user of the model.

In terms of the algorithm running time, ABC-Miner$_l^{mn}$ takes somewhat less computational time compared to ABC-Miner, as shown in Table 7, where, on average over the used 33 datasets, it took 0.79 of the time used by ABC-Miner. The reason is that ABC-Miner$_l^{mn}$ learns simpler BNs compared to ABC-Miner. This is attributed to several facts, as follows. First, the data used for learning a local BN in the BMN classifier is a subset of the training set. In addition, the number of probabilities (CPT size for each variable) to be calculated for each variable in the local BN is smaller in the BMN classifier. Moreover, the convergence to a good solution is faster while learning local BNs. This makes the overall time for building a BMN less than a BAN. The reason that ABC-Miner$_g^{mn}$ has a longer running time than ABC-Miner (about the double as shown in Table 7) is that ABC-Miner$_g^{mn}$ takes more iterations to converge than ABC-Miner$_l^{mn}$ and ABC-Miner. This is because the solution created in one iteration of ABC-Miner$_g^{mn}$ is more complex as it contains decision components for all the local BNs that compose a BMN, compared to ABC-Miner$_l^{mn}$ in which the solution has only decision components for one local BN, and compared to ABC-Miner in which the solution has only decision components for one BN classifier.

Therefore, it is less likely that the colony will converge fast to solutions with a large number of decision components such as in the global approach, since the number of combinations of decisions components is larger compared to the local approach and ABC-Miner. However, ABC-Miner$_g^{mn}$ compensates for this by achieving the best predictive accuracy results.

One important remark to be mentioned regarding the computational time is that both ABC-Miner$_l^{mn}$ and ABC-Miner$_g^{mn}$ increase the opportunities for algorithm parallelization, by comparison with ABC-Miner. In other words, since in ABC-Miner$_l^{mn}$ each local BN is learnt independently, the creation of the local BNs can be performed simultaneously without any inter-dependency. As for ABC-Miner$_g^{mn}$, in a given iteration ants in each class-based construction graph can construct their local BNs simultaneously; however, the algorithm must wait until all the ants in the current iteration finish constructing all the local BNs to build a BMN and proceed to the next iteration. In either way, this parallelization can have a significant positive impact on the computational efficiency of the algorithms.

8 Concluding Remarks

In this paper, we have proposed two new ACO algorithms for learning BMN classifiers: ABC-Miner$_l^{mn}$, which utilizes a local approach in the learning process, and ABC-Miner$_g^{mn}$, which uses a global approach in the learning process. A novel BN quality measure was also introduced for ABC-Miner$_l^{mn}$, so that each local BN is evaluated according to the difference in the Log-Likelihoods of cases with positive and negative classes. Empirical results in terms of predictive accuracy showed that our new algorithms significantly outperformed our previously introduced ACO algorithm for learning BAN classifiers, as well as significantly outperforming both BMN-K2 (a greedy search-based algorithm for learning BMNs) and the well-known Naïve-Bayes, CL-tree for learning TANs, CL-tree for learning BMNs, and Algorithm-B for learning GBNs. In addition, our proposed algorithms have been shown to be very competitive to other well-known classification algorithms: JRip, J48 and SVM. The running time results showed that ABC-Miner$_l^{mn}$ reduces the computational time compared to ABC-Miner. Moreover, the algorithm's search strategy makes it suitable for parallelization, which can have a significant impact on the reduction of the running time.

As a future work, we would like to explore the effect of using different measures to evaluate the quality of the constructed local BNs in ABC-Miner$_l^{mn}$ and to evaluate the constructed BMN classifiers in ABC-Miner$_g^{mn}$ by each ant in the colony before pheromone update. Probability-based measures, such as Bayesian Information Reward (BIR) and Kullback-Leibler (KL) divergence can be employed in the training phase, while the area under the Receiver Operating Characteristic (ROC) and the Precision Recall (PR) curves can be used for the classifier evaluation in the testing phase.

References

[1] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, 2nd ed. Addison Wesley, 2005.

[2] N. Friedman, D. Geiger, M. Goldszmidt, G. Provan, P. Langley, and P. Smyth, "Bayesian Network Classifiers," *Machine Learning*, vol. 29, pp. 131–163, 1997.

[3] J. Cheng and R. Greiner, "Comparing Bayesian Network Classifiers," in *15th Annual Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann, 1999, pp. 101–108.

[4] ——, "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*. London, UK: Springer, 2001, pp. 141–151.

[5] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004.

[6] Y. Wu, J. McCall, and D. Corne, "Two Novel Ant Colony Optimization Approaches for Bayesian Network Structure Learning," in *IEEE Congress on Evolutionary Computation (CEC)*. New York, NY, USA: IEEE Press, 2010, pp. 1–7.

[7] 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*, vol. 31, no. 3, pp. 291–311, 2002.

[8] R. Daly and Q. Shen, "Learning Bayesian Network Equivalence Classes with Ant Colony Optimization," *Journal of Artificial Intelligence Research (JAIR)*, vol. 35, pp. 391–447, 2009.

[9] P. C. Pinto, A. Nägele, M. Dejori, T. A. Runkler, and Ao, "Using a Local Discovery Ant Algorithm for Bayesian Network Structure Learning," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 767–779, 2009.

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

[11] K. Salama, A. Abdelbar, and A. A. Freitas, "Multiple Pheromone Types and Other Extensions to the Ant-Miner Classification Rule Discovery Algorithm." *Swarm Intelligence*, vol. 5, no. 3-4, pp. 149–182, December 2011.

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

[13] F. Otero, A. A. Freitas, and C. Johnson, "Handling continuous attributes in ant colony classification algorithms," in *IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*. New York, NY, USA: IEEE Press, 2009, pp. 225–231.

[14] K. Salama, A. Abdelbar, F. Otero, and A. A. Freitas, "Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery." *Applied Soft Computing*, vol. 13, no. 1, pp. 667–675, 2013.

[15] 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)*. ACM Press, 2006, pp. 43–50.

[16] K. Salama and A. A. Freitas, "ABC-Miner: an Ant-based Bayesian Classification Algorithm," in *8th International Conference on Swarm Intelligence (ANTS'12)*, ser. LNCS 7461. Berlin: Springer, 2012, pp. 13–24.

[17] ——, "Learning Bayesian Network Classifiers Using Ant Colony Optimization," *Swarm Intelligence*, vol. 7, no. 2-3, pp. 229–254, 2013.

[18] D. Chickering, M. Geiger, and D. Heckerman, "Learning Bayesian Networks is NP-Hard," *Advanced Technologies Division, Microsoft Corporation, Technical Report*, 1994.

[19] F. Otero, A. A. Freitas, and C. Johnson, "cAnt-Miner: an Ant Colony Classification Algorithm to Cope with Continuous Attributes," in *Ant Colony Optimization and Swarm Intelligence (ANTS'08)*, ser. LNCS, no. 5217. Berlin: Springer, 2008, pp. 48–59.

[20] M. Dorigo and T. Stützle, *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*. New York, NY, USA: Springer, 2003, vol. 57.

[21] M. Dorigo, G. M. Caro, and L. M. Gambardella, "Ant Algorithms for Discrete Optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[22] R. Daly, Q. Shen, and S. Aitken, "Review: Learning Bayesian Networks: Approaches and Issues," *Knowledge Engineering Reviews*, vol. 26, no. 2, pp. 99–157, 2011.

[23] D. Heckerman, *Studies in Computational Intelligence: Innovations in Bayesian Networks.* Berlin: Springer, 2008, vol. 156, ch. 3: A Tutorial on Learning with Bayesian Networks., pp. 33–82.

[24] G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data," *Machine Learning*, vol. 9, no. 4, pp. 309–347, 1992.

[25] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data," *Machine Learning*, vol. 20, no. 3, pp. 197–243, 1995.

[26] S. Yang and K.-C. Chang, "Comparison of Score Metrics for Bayesian Network Learning," *IEEE Transactions on Systems, Man, and Cybernetics - Part A*, vol. 32, no. 3, pp. 419–428, 2002.

[27] R. Kelner and B. Lerner, "Learning Bayesian network classifiers by risk minimization," *International Journal of Approximate Reasoning*, vol. 53, no. 2, pp. 248–272, 2012.

[28] D. Geiger and D. Heckerman, "Knowledge Representation and Inference in Similarity Networks and Bayesian Multinets," *Artificial Intelligence*, vol. 82, no. 2, pp. 45–74, 1996.

[29] W. Buntine, "Theory Refinement on Bayesian Networks," in *17th Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann, 1991, pp. 52–60.

[30] K. Huang, I. King, and M. Lyu, "Discriminative Training of Bayesian Chow-Liu Multinet Classifiers," in *International Joint Conference on Networks*, vol. 1. New York, NY, USA: IEEE Press, 2003, pp. 484–488.

[31] Y. Gurwicz and B. Lerner, "Bayesian Class-Matched Multinet Classifier," in *International Conference on Structural, Syntactic, and Statistical Pattern Recognition (IAPR'6)*. Berlin: Springer, 2006, pp. 145–153.

[32] P. Langley, "Induction of Recursive Bayesian Classifiers," in *European Conference on Machine Learning (ECML)*. Berlin: Springer, 1993, pp. 153–164.

[33] K. Salama and A. Abdelbar, "Extensions to the Ant-Miner Classification Rule Discovery Algorithm," in *7th International Conference on Swarm Intelligence (ANTS'10)*, ser. LNCS, no. 6234. Berlin: Springer, 2010, pp. 167–178.

[34] ——, "Exploring Different Rule Quality Evaluation Functions in ACO-based Classification Algorithms," in *IEEE Swarm Intelligence Symposium*. Piscataway, NJ, USA: IEEE Press, 2011, pp. 1–8.

[35] D. Martens, B. Baesens, and T. Fawcett, "Editorial survey: swarm intelligence for data mining," *Machine Learning*, vol. 82, no. 1, pp. 1–42, 2011.

[36] F. Otero, A. A. Freitas, and C. Johnson, "Inducing Decision Trees with an Ant Colony Optimization Algorithm," *Applied Soft Computing*, vol. 12, no. 11, pp. 3615–3626, 2012.

[37] U. Boryczka and J. Kozak, "Ant Colony Decision Trees," in *4th International Conference on Computational Collective Intelligence: Technologies and Applications (ICCCI'11)*. Berlin: Springer, 2010, pp. 4373–382.

[38] P. C. Pinto, A. Nägele, M. Dejori, T. A. Runkler, and J. M. C. Sousa, "Learning of Bayesian Networks by a Local Discovery Ant Colony Algorithm," in *IEEE World Congress on Evolutionary Computation (CEC 2008)*. New York, NY, USA: IEEE Press, 2008, pp. 2741–2748.

[39] R. Daly and Q. Shen, "Using Ant Colony Optimization in Learning Bayesian Network Equivalence Classes," in *UK Workshop on Computational Intelligence (UKCI)*. Palo Alto, CA, USA: AAAI Pres, 2006, pp. 111–118.

[40] J. Ji, R. Hu, H. Zhang, and C. Liu, "A Hybrid Method for Learning Bayesian Networks based on Ant Colony Optimization," *Applied Soft Computing*, vol. 11, pp. 3373–3384, 2010.

[41] K. 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)*. New York, NY, USA: IEEE Press, 2013, pp. 3079–3086.

[42] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2010.

[43] A. A. Freitas, D. Wieser, and R. Apweiler, "On the importance of comprehensible classification models for protein function prediction," *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, vol. 7, no. 1, pp. 172–182, 2010.

[44] A. Asuncion and D. Newman, "UCI Machine Learning Repository. URL:http://www.ics.uci.edu/ mlearn/MLRepository.html," 2007.

[45] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 1, no. 7, pp. 1–30, 2006.

[46] S. Garca and F. Herrera, "An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons," *Journal of Machine Learning Research*, vol. 9, pp. 2677–2694, 2008.

[47] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

[48] D. Marteens, J. Vanthienen, W. Verbeke, , and B. Baesens, "Performance of Classification Models from a User Perspective." *Decision Support Systems*, vol. 51, no. 4, pp. 782–793, 2011.