Learning Bayesian Networks by Ant Colony Optimisation: Searching in two Different Spaces

Luis M. de Campos\textsuperscript{1}, José A. Gámez\textsuperscript{2}, José M. Puerta\textsuperscript{2}

\textsuperscript{1} Departamento de Ciencias de la Computación e I.A.
Universidad de Granada
18071 - Granada, SPAIN.
\textit{lci@decsai.ugr.es}

\textsuperscript{2} Departamento de Informática
Universidad de Castilla-La Mancha
02071 - Albacete, SPAIN
\{jgamez,jpuerta\}@info-ab.uclm.es

Abstract

The most common way of automatically learning Bayesian networks from data is the combination of a scoring metric, the evaluation of the fitness of any given candidate network to the data base, and a search procedure to explore the search space. Usually, the search is carried out by greedy hill-climbing algorithms, although other techniques such as genetic algorithms, have also been used.

A recent metaheuristic, \textit{Ant Colony Optimisation (ACO)}, has been successfully applied to solve a great variety of problems, being remarkable the performance achieved in those problems related to path (permutation) searching in graphs, such as the Traveling Salesman Problem. In two previous works [13, 12], the authors have approached the problem of learning Bayesian networks by means of the search+score methodology using ACO as the search engine.

As in these articles the search was performed in different search spaces, in the space of orderings [13] and in the space of directed acyclic graphs [12]. In this paper we compare both approaches by analysing the results obtained and the differences in the design and implementation of both algorithms.

1 Introduction

Bayesian Networks (BNs), also known as Belief or Causal Networks, are knowledge representation tools able to efficiently manage the dependence and independence relationships among the random variables that compose the problem domain we
want to model. This representation has two components: a) a graphical structure, more precisely a directed acyclic graph (dag), and b) a set of parameters, which together specify a joint probability distribution over the random variables [29, 26]. In Bayesian networks, the graphical structure represents dependence and independence relationships. The parameters are a collection of conditional probabilities, which measure the strength of the relationships.

Once we have the Bayesian network specified, it constitutes an efficient tool to perform inference tasks. However, there still remains the problem of building such a network. So, an interesting task is to develop automatic methods capable of learning the network directly from data, as an alternative or a complement to the method of eliciting opinions from experts.

Algorithms for automatically learning (the structure of) BNs have been studied, basically from two points of view: methods based on conditional independence tests [11, 15, 32, 30] and methods based on a scoring metric optimisation [9, 24, 27]. This classification is not exhaustive and/or strict, there also exist algorithms that use a combination of these two methods [1, 2, 10, 31]. In this paper we focus on the score+search approach to learn Bayesian networks.

As learning Bayesian networks is, in general, a NP-hard problem [8] and exact methods become unfeasible, the problem is usually approached by using heuristic methods. Most existing scoring-based learning algorithms apply standard heuristic search techniques, such as greedy (stochastic or deterministic) hill-climbing, and metaheuristics (simulated annealing, genetic algorithms, tabu search, etc).

In [13, 12] a recent stochastic search technique, the Ant Colony Optimisation metaheuristic is used as the search engine for the BN learning problem.

The ACO metaheuristic is the result of attempt to provide a framework to several applications of ant algorithms, which are proposed earlier in the literature [21, 19, 33].

The search space in which the ACO algorithms operate can be defined in two different ways: dags and orderings (permutations of the variables in the BN). Searching in the space of dags was usually considered as the standard choice, due to the fact that a Bayesian network is a dag itself. However, in the last years, several authors [28, 14, 22, 17] have shown that the space of orderings is much ‘smoother’ than the space of dags. Moreover, it is known that, by providing a good ordering of the variables, the learning algorithms become more efficient and accurate. In fact, there is a number of algorithms that, although they perform the search in the space of dags, need to use such an ordering [2, 1, 15, 9].

Because of these reasons and to take advantage of the similarities with the application of ACO to the Traveling Salesman Problem (TSP) [20], the space of orderings was our first option when we decided to study the applicability of ACO (as the search engine) to the Bayesian network learning problem [13]. Following this research line, in [12] we studied the applicability of ACO algorithms to the search in the space of dags. To do so, we had to describe all the elements necessary to tackle our learning problem using ACO, because no direct adaptation from a well known problem was possible.

In this paper we compare both algorithms by comparing the two approaches on two well-known domains: ALARM and INSURANCE. We begin in Section 2
with the preliminaries, where we briefly describe some of the concepts and methods about Bayesian networks. Background knowledge about ant colony algorithms is not provided in this paper, the reader is referred to the first paper of this special issue, and to the references therein. In Section 3 we give a detailed description of the ACO-based learning BNs in the space of orderings, summarizing the main points of [13], while in Section 4 we briefly describe the ACO-based algorithm learning of BNs in the space of dags, which is explained in details in [12]. In Section 5 we present the experimental evaluation of both approaches, and we also compare with two hill climbing algorithms: HCST (in the space of dags [24]) and HCSN (in the space of orderings [17]). Finally, Section 6 contains the concluding remarks.

2 Preliminaries

In this section we will briefly review some basic concepts about Bayesian networks and how to automatically learn them from data bases.

A BN is a directed acyclic graph (dag) \( G = (V, E) \), where the set of nodes \( V = \{x_1, x_2, \ldots, x_n\} \) represents the system variables and \( E \), a set of arcs, represents the direct dependence relationships among the variables. A set of parameters is also stored for each variable in \( V \), usually conditional probability distributions. For each variable \( x_i \in V \) we have a family of conditional distributions \( P(x_i|Pa(x_i)) \), where \( Pa(x_i) \) represents the parent set of the variable \( x_i \). From these conditional distributions we can recover the joint distribution over \( V \):

\[
P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^n P(x_i|Pa(x_i))
\]

This expression represents a decomposition of the joint distribution. The dependence/independence relationships which make possible this decomposition are graphically encoded (through the d-separation criterion [29]) by means of the presence or absence of direct connections between pairs of variables.

The problem of learning a BN can be stated as follows: given a training set \( D = \{v^1, \ldots, v^m\} \) of instances of \( V \), find the BN that best matches \( D \). The common approach to this problem is to introduce a scoring function, \( f \), that evaluates each network with respect to the training data, and then to search for the best network according to this score. Different Bayesian and non-Bayesian scoring metrics can be used [1, 7, 9, 24, 27].

A desirable and important property of a metric is its decomposability in the presence of full data, i.e., the scoring function can be decomposed in the following way:

\[
f(G : D) = \sum_{i=1}^n f(x_i, Pa(x_i) : N_{x_i, Pa(x_i)})
\]

where \( N_{x_i, Pa(x)} \) are the statistics of the variable \( x_i \) and \( Pa(x_i) \) in \( D \), i.e., the number of instances in \( D \) that match each possible instantiation of \( x_i \) and \( Pa(x_i) \).
The decomposition of the metric is very important for the learning task: a local search procedure that changes one arc at each move can efficiently evaluate the improvement obtained by this change, because it can reuse most of the computations made in previous stages.

In the rest of this section we review the K2 metric, because this will be the scoring function used in our experiments, although any other decomposable metric could be used.

2.1 The K2 Metric and the K2SN heuristic.

The K2 algorithm [9] is perhaps the best known of the algorithms for learning BNs, and it has been the basis of many research work. This algorithm uses a Bayesian scoring metric, which measures the joint probability of a BN \( G \) and a database \( D \). The metric adopted the name of the algorithm, so that it is referred to as the K2 metric, whose expression is

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

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

Assuming a uniform prior for \( P(G) \) and, due to the requirement of a given ordering \( \theta \) as an input to the K2 algorithm, it is possible to maximise the metric by working separately on each variable \( x_i \) and its parent set \( Pa(x_i) \). The algorithm K2 goes over the given ordering \( \theta \), and for each variable it starts with an empty parent set, and (step by step) adds to it that variable which is lower ranked than \( x_i \) and leads to a maximum increase in the resultant probability, measured as\(^1\):

\[
f_{K2}(x_i, Pa(x_i)) = \log \left( \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \right)
\]

The K2SN algorithm [16] is an extension of K2 in which the requirement of a given ordering is removed. The algorithm starts with an empty graph and iteratively determines the best node to be added. At each step, for each variable still non included in the graph, the best parent set is selected among the variables already included in the graph, and the variable producing the best K2 score is added to the graph, together with the arcs defining its parent set. Therefore, K2SN obtains an ordering and a graph compatible with it.

\(^1\)By using the logarithmic version of the K2 metric, it becomes decomposable, in the sense defined in eq. (2).
3 Learning Bayesian Networks Using Ant Colony Optimisation: Searching in the space of orderings

In this section we develop the scoring-based BNs learning algorithm introduced in [13]. In this algorithm the search is guided by a colony of ants and is carried out in the space of orderings (permutations of the network variables).

To do so, we give a brief introduction to the problem of learning in the space of orderings, and then we describe the algorithm by detailing each one of its components (inspired by the application of ant colony optimisation to the traveling salesman problem [20]).

3.1 Searching in the Space of the Orderings

For a dag $G$, given a causal ordering $\theta$ (i.e., an ordering compatible with the topology of the dag\(^2\)), the following independence relationships hold: $x_i$ is conditionally independent of all the variables that precede it in the ordering, given its parent set $Pa_G(x_i)$, for all $x_i$. This fact provides a systematic method to build Bayesian networks: For each node $x_i$, the parents of $x_i$ in the dag are the minimal subset of predecessors of $x_i$ (in the ordering $\theta$) which makes $x_i$ conditionally independent of the rest of its predecessors.

However, different orderings may produce different networks. We would prefer those networks that are able to represent as much true independence relationships as possible (i.e., having as few arcs as possible). For that reason it makes sense to search for the best ordering.

In the literature [3] we can find works in which the learning task is divided into two stages: first, the greatest effort is devoted to search a good ordering $\theta^*$, and then a secondary search process is carried out in the space of dags compatible with $\theta^*$. In this work, the search space is the set of $n!$ orderings, $\theta$, of the variables in $V$ (i.e., the set of permutations of $n$ elements). However, our method will simultaneously search for the best network consistent with the ordering being obtained, thus avoiding the second search stage.

3.2 Describing the ACO algorithm

Standard ACO notation will be followed in this paper, thus $\eta_{ij}$ stands for the heuristic value associated to edge $(i, j)$, $\tau_{ij}$ stands for the amount of pheromone stored in edge $(i, j)$, $\tau_0$ is the initial amount of pheromone, and $\rho$ is the parameter that controls the pheromone evaporation (decay).

First of all we describe what kind of heuristic will be used in order to add problem specific knowledge to the search process. Our idea is to use the K2SN heuristic [16] as the heuristic component in the ACO approach. Therefore, the heuristic information is dynamic, because using K2SN, the score $f(x_j, Pa(x_j))$ of going from node $i$ to node $j$ depends on the nodes visited previously, because

\(^2\)If there is an arc $x_i \rightarrow x_j$ in $G$, then $\theta(x_i) < \theta(x_j)$. 
$Pa(x_j)$ is a subset of them. In this way, each ant will carry out a K2SN search but weighted by the amount of pheromone deposited in each arc.

Below, we describe the rest of the necessary components to model our ACO-based algorithm, which is based on Ant Colony System (ACS)[20].

- The representation of the problem on a graph could look trivial, because we are dealing with a graph (the network). However, our search space is the set of permutations of the variables, so that we consider the complete graph defined over the variables (in this way, it is always possible to reach node $j$ from node $i$ for every pair of nodes $(i, j)$). We thus have a graph representation equal to the one used for the TSP problem, although in its asymmetric\(^3\) version, because in our case going from $i$ to $j$, with score $f(x_j, Pa(x_j))$, is not the same as going from $j$ to $i$ (with score $f(x_i, Pa(x_i))$).

- In the TSP the heuristic component was defined as the inverse of the distance cities $i$ and $j$. In our case, we will look for the log-likelihood $\log P(G, D)$ maximisation, so the result will be negative. Because of this, we will use $\eta_{ij} = \frac{1}{f(x_j, Pa(x_j))}$ as our heuristic component, with $Pa(x_j)$ being calculated by the K2 metric.

- Pheromone is initialised by depositing a small amount of pheromone, $\tau_0$, in every link of the graph. This amount is usually calculated as a function of the goodness of a solution to the problem obtained by using a quick method (a greedy algorithm). In our case, the initial amount of pheromone is calculated as: $\tau_0 = \frac{1}{n! |S_{K2SN}|}$, where $n$ is the number of variables, $S_{K2SN}$ is the solution obtained by the algorithm K2SN, and $f(S_{K2SN})$ is the score of that solution.

  Observe that we are also using the absolute value, $|\cdot|$, in the expression for $\tau_0$. The reason is again that the value of $f(S_{K2SN})$ is negative, since we always use the logarithmic version of the metric.

- **Probabilistic transition rule.** The next node to be visited in the sequence is selected in the same way as in the application of ACS to the TSP [20].

- Another interesting point worth commenting upon is the selection of the first node. In problems like the TSP, the choice of the first city is of minor importance, because the goal is to obtain a cycle. In our problem, however, the choice of the first node in the ordering is very important, because this choice will influence on the subsequent steps (note that the first node will be necessarily a root node). Our proposal is to select the initial node in the ordering in the same way we will select the rest of nodes in the ordering $2, \ldots, n$. To do this, we handle the situation as if we had a dummy node $x_0$ fixed as our initial node, and use K2SN to look for our true initial node in the ordering (in this case $\tau_{0i} = \tau_0$ and $\eta_{0i} = \frac{1}{f(x_0, i)}$). This way of proceeding does not imply to enlarge our pheromone array $\tau$, because we can use the diagonal of $\tau$ to store the values $\tau_{0ij}$.

\(^3\) In the asymmetric TSP problem there are at least two cities $i$ and $j$, such that, $d_{ij} \neq d_{ji}$.
- The classical local and global pheromone updating rules are used in the algorithm, that is, those used in the TSP [20]. Therefore, in the global pheromone updating, only the edges belonging to the best solution found so far, \( S^+ \), are updated, and \( \Delta \tau_{ij} \) is calculated as \( \frac{1}{f(S^+)} \), with \( f(\cdot) \) being the scoring metric used in our problem.

- As a local optimizer we will use the algorithm HCSN [17]. This is an algorithm specifically designed for carrying out a hill climbing in the space of orderings. The local neighbourhood is defined by the interchange of two nodes in the ordering. In [17] an efficient method for evaluating neighbouring configurations is described. The local optimizer has been used just in the last iteration, where the result obtained by each ant is the starting point of the corresponding local search. This decision is due to the high computational cost exhibited by the local optimisation procedure. Besides, an alternative scheme was tried (optimising every 10 iterations), but the results achieved were worse than those obtained by optimising just in the last iteration.

Figure 1 shows the process followed by a K2SN ant while constructing its solution. In step 3.2.b.ii \( K2(x_i, \text{VISITED}) \) returns the parent set for \( x_i \), selected among the nodes previously visited, and computed according to the heuristic K2SN. In step 3.2.b.iii \( f(x_i, Pa(x_i)) \) returns the value of the scoring metric used in this work (eq. 3), and \( \frac{1}{f(x_i, Pa(x_i))} \) is the heuristic component \( \eta_{ji} \).

4 Learning Bayesian Networks Using Ant Colony Optimisation: Searching in the space of dags

In this section we develop the scoring-based BNs learning algorithm introduced in [12]. In this algorithm the search is guided by a colony of ants and carried out in the space of dags.

In this case the algorithm is not an adaptation of a well known ACO algorithm, as the one for solving the TSP in the previous section. Therefore, more design decisions are needed in order to define/specify the ACO algorithm for learning BN in the space of dags. The first one is that in this case the behaviour of the ants will be based on a different Bayesian network learning algorithm, as is the Algorithm B [6]. Because of this we talk about ant-B.

Below we describe briefly the main components of the algorithm; for a complete description the reader is referred to the original paper [12].

- **Representation of the problem.** In this case, the representation of the problem is a graph, where the states of the problem are dags with \( n \) nodes. Thus, a state \( G_h \) will be a graph with the nodes \( x_i \in V \) and exactly \( h \) arcs and no directed cycle. The ant incremental construction of the solution starts from the empty graph \( G_0 \) (arc-less dag) and proceeds by adding an arc \( x_j \rightarrow x_i \) to the current state \( G_h \), i.e., \( G_{h+1} = G_h \cup \{x_j \rightarrow x_i\} \). The final solution will be the state \( G_k \) in which the ant decides to stop the construction phase.
• **Heuristic information.** The selected heuristic is to include in the graph the arc producing the greatest increase in the selected decomposable metric $f$. Therefore, we define

$$\eta_{ij} = f(x_i, Pa(x_i) \cup \{x_j\}) - f(x_i, Pa(x_i))$$ (4)

In fact this behaviour reproduces the way in which the algorithm B proceeds. As in the space of orderings, the heuristic information is not static. In this case the heuristic also depends on the ant’s internal state, i.e., the current graph representing the partial solution of the problem, which determines the identity of the sets $Pa(x_i)$.

• **Pheromone updating rules.** As in the case of searching in the space of orderings, the classical ACS global and local pheromone updating rules are applied. Now, if $G^+$ is the best graph found so far, then $\Delta \tau_{ij}$ is computed as $\frac{1}{|\{i \rightarrow j \in G^+\}|}$ if $x_j \rightarrow x_i \in G^+$. The initial level of pheromone is $\tau_0 = \frac{1}{n |\{i \rightarrow j \in G_{K2SN} \}|}$, where $n$ is the number of variables and $G_{K2SN}$ is the network obtained by the K2SN heuristic [16]. For the same reasons as in the case of orderings the absolute value is considered.

• **Probabilistic transition rule.** The next arc to be included in the current graph, $G$ is selected by an ant in a way similar to that used by algorithm B, but using a stochastic decision rule (instead of a deterministic rule) that also takes into account the pheromone deposited at each arc (thus obtaining expressions similar to those considered in the ACS).

• **Local optimizer.** A hill climbing algorithm with the standard operators of arc addition, arc deletion and arc reversal (HCST) [24] is considered. This algorithm is a greedy best-improvement local search where, at each step, the best move according to the metric and operators used is selected.

We should note that the transition operators chosen for HCST contain the one chosen for Ant B. Therefore, once an ant has obtained a solution, then by deleting or reversing an arc, the HCST algorithm can escape from an eventual local optimum reached by the ant.

In order to perform a fair comparison between the two ACO algorithms, the local optimizer has been used just in the last iteration.

Figure 2 shows the steps followed by an ant in our system to build a solution.

5 Experimental Evaluation

In order to compare the behavior of the two ACO algorithms proposed in the paper, two domain problems have been selected: ALARM [4] and INSURANCE [5]. The ALARM network (Figure 3) has 37 nodes and 46 arcs and is used for diagnosis in a medical domain. It was considered as a benchmark for evaluating Bayesian network learning algorithms. All the experiments with ALARM are carried out on
the first 3000 cases of the ALARM database (which contains 20000 cases, generated by probabilistic logic sampling [25]). The INSURANCE network (Figure 4), which contains 27 variables and 52 arcs, is a network for evaluating car insurance risks. The experiments with INSURANCE use three databases containing 10000 cases each, generated also by probabilistic logic sampling.

The aim of each experiment is to recover the network from which the data bases were generated. On the other hand, we are also interested in showing the efficiency of the different algorithms. In order to carry out the empirical comparison we have collected the following parameters:

- Parameters to measure the structural difference between the original and the learned networks: the number of arcs added (A), deleted (D) and inverted\(^5\) (I), compared with the original network.

- The value of the K2 metric (log version), eq. (3). As a reference for the goodness of the results we can consider the K2 values for the true graphical structures (Figures 3 and 4), which are -14412.69 for ALARM and -58120.95 for INSURANCE.

- The KL value, defined as follows:

\[
KL(G : D) = \sum_{i=1, Pa(x_i) \neq \emptyset}^{n} Dep(x_i, Pa(x_i))
\]  

where \(Dep(X, Y)\) is the measure of mutual information between two sets of variables \(X\) and \(Y\). Note that \(KL(G : D)\) is a decreasing monotonic transformation of the Kullback distance between the probability distribution associated to the database and the probability distribution associated to the network \(G\) [11, 27]. We use this transformation because it can be calculated very efficiently, whereas the computation of the Kullback distance has an exponential complexity. The interpretation of \(KL(G : D)\) is: the higher this parameter the better the network.

- Measures to evaluate the complexity of the algorithms. TEst \(n\) represents the total number of statistics \(N_{ijk}\) evaluated during the learning process. This value is not necessarily equal to the number of statistics truly computed from the data\(^6\), since we can use hashing techniques to avoid the necessity of recomputing previously calculated values, thus reducing substantially this number. So, we also show the number of different statistics used, EstEv. Finally, as the complexity of the computation of the statistics grows exponentially with the number of variables involved, we also show the average number of variables appearing in the different statistics evaluated, NVars. For comparative

---

\(^4\) This is a method for randomly generating cases from the joint probability distribution captured by a BN.

\(^5\) Only if they give rise to non-equivalent structures [30].

\(^6\) Note that this is usually the most costly process for scoring-based learning algorithms.
purposes, a raw estimation of the running time employed by an algorithm is $\text{EstEv} \times 2 \text{ NVars}$.

In addition to the previously described measures, in the ACO algorithms, we have also considered the value of the $K_2$ metric attained \textit{before} using the local optimizer, $\text{K2noHC}$.

As a reference we will consider the results obtained by the algorithms HCSN and HCST in two different cases:

- **EMP**: Starting from the empty network and taking as initial ordering the one in which the variables are listed in the data base.
- **K2SN**: Starting from the network (and the ordering) found by running the algorithm K2SN.

Notice, that the two hill climbing algorithms are those used as local optimizers in the ACO approach.

In order to perform a fair comparison, the same parameter setting has been used in the two ACO algorithms: $\rho = 0.4$, $\beta = 2.0$, $q_0 = 0.8$, 10 ants and 100 iterations. No systematic tuning has been carried out. On the contrary, the values have been selected to be similar to those used in other optimisation problems [23, 20]. Besides, the same parameters setting has been used in both domains, ALARM and INSURANCE.

The results of the experiments are displayed in Tables 1, 2, 3 and 4. For ACO-K2SN and ACO-B, ten runs have been carried out, so we display the average value $\mu$ and the standard deviation $\sigma$ of each item. With respect to HCSN and HCST, only one run has been carried out due to its deterministic behaviour.

<table>
<thead>
<tr>
<th>Table 1: Results obtained by HCSN for ALARM and INSURANCE.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>K2</td>
</tr>
<tr>
<td>KL</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>EstEv</td>
</tr>
<tr>
<td>TEst</td>
</tr>
<tr>
<td>NVars</td>
</tr>
</tbody>
</table>

From the analysis of the results we are in a position to draw the following conclusions:

- In both domains the proposed algorithms obtain high-quality networks. In fact, in the two cases of study, the networks learnt have even a better $K_2$ score than the reference values.
Table 2: Results obtained by HCST for ALARM and INSURANCE.

<table>
<thead>
<tr>
<th></th>
<th>ALARM</th>
<th>INSURANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EMP</td>
<td>K2SN</td>
</tr>
<tr>
<td>K2</td>
<td>-14425.62</td>
<td>-14520.21</td>
</tr>
<tr>
<td>KL</td>
<td>9.220</td>
<td>9.229</td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>EstEv</td>
<td>3375</td>
<td>4970</td>
</tr>
<tr>
<td>TEst</td>
<td>$1.54 \times 10^4$</td>
<td>$4.70 \times 10^3$</td>
</tr>
<tr>
<td>NVars</td>
<td>2.99</td>
<td>3.57</td>
</tr>
</tbody>
</table>

Table 3: Results of ACO-K2SN for ALARM and INSURANCE.

<table>
<thead>
<tr>
<th></th>
<th>ALARM</th>
<th>INSURANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>K2</td>
<td>-14404.07</td>
<td>3.28</td>
</tr>
<tr>
<td>KL</td>
<td>9.228</td>
<td>0.003</td>
</tr>
<tr>
<td>A</td>
<td>2.20</td>
<td>0.60</td>
</tr>
<tr>
<td>D</td>
<td>1.40</td>
<td>0.49</td>
</tr>
<tr>
<td>I</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>K2noHC</td>
<td>-14462.46</td>
<td>14.09</td>
</tr>
<tr>
<td>EstEv</td>
<td>89154.80</td>
<td>8099.94</td>
</tr>
<tr>
<td>TEst</td>
<td>$58.88 \times 10^6$</td>
<td>$51.65 \times 10^3$</td>
</tr>
<tr>
<td>NVars</td>
<td>4.97</td>
<td>0.08</td>
</tr>
</tbody>
</table>

- With respect to the accuracy of the algorithms, it is clear that ACO improves the results obtained by the local search algorithms. This conclusion is valid for the K2 and KL values, and (in most of the cases) also for structural differences (A+D+I). At this point we can wonder why HCSN-EMP works better than HCSN-K2SN for INSURANCE. The answer is that in this case the order in which the variables are listed in the data base is in fact a causal ordering.

- The effect of the local optimisation depends of the ACO algorithm. In the case of searching in the space of orderings, the solutions obtained by the ants are considerably improved by the local search. In the case of searching in the space of dags, the solutions obtained by the ants are already very good, and so the local search has a minor effect.

- The comparison between the accuracy of the two ACO algorithms is a bit complex. For the two domains, it seems that the ant-B achieves better re-
Table 4: Results of ACO-B for ALARM and INSURANCE.

<table>
<thead>
<tr>
<th></th>
<th>ALARM</th>
<th></th>
<th>INSURANCE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
<td>Best</td>
<td>$\mu$</td>
</tr>
<tr>
<td>K2</td>
<td>-14.406.06</td>
<td>4.54</td>
<td>-14.401.29</td>
<td>36.68</td>
</tr>
<tr>
<td>KL</td>
<td>9.231</td>
<td>0.003</td>
<td>9.231</td>
<td>8.450</td>
</tr>
<tr>
<td>A</td>
<td>3.30</td>
<td>1.79</td>
<td>2</td>
<td>3.60</td>
</tr>
<tr>
<td>D</td>
<td>1.10</td>
<td>0.30</td>
<td>1</td>
<td>8.50</td>
</tr>
<tr>
<td>I</td>
<td>1.70</td>
<td>1.90</td>
<td>0</td>
<td>2.43</td>
</tr>
<tr>
<td>K2noHC</td>
<td>-14.407.32</td>
<td>5.01</td>
<td></td>
<td>-57811.04</td>
</tr>
<tr>
<td>EstEv</td>
<td>43349.60</td>
<td>953.87</td>
<td></td>
<td>28738.77</td>
</tr>
<tr>
<td>TEst</td>
<td>$6.147 \times 10^6$</td>
<td>$1.167 \times 10^4$</td>
<td></td>
<td>$3.189 \times 10^5$</td>
</tr>
<tr>
<td>NVars</td>
<td>4.43</td>
<td>0.02</td>
<td></td>
<td>4.35</td>
</tr>
</tbody>
</table>

Results than ant-K2SN, before firing the local optimizer; in fact the solutions obtained are better than those obtained by the hill climbing algorithms. However, although the solutions obtained by ant-K2SN are (by far) worse than those obtained by ant-B, from the experiments we can see that these solutions are really very good starting points for the local optimizer HCSN, being the solutions obtained after the whole process are similar for both ACO approaches.

- With respect to the efficiency of the algorithms, it is clear that from the values obtained for EstEv, TEst and NVars, the ACO algorithms are much more complex than local search algorithms.

However, if we consider the expression previously introduced ($EstEv \cdot 2^{NVars}$) for measuring complexity of a BN learning algorithm, then we can see that the ratio between ACO-K2SN and HCSN is 9.41 and 4.47 for ALARM and INSURANCE respectively. These are very positive values, because we should not forget that in ACO-K2SN the algorithm HCSN is launched ten times. As an explanation to this fact, we think that the solutions obtained by the ants are really good starting points for the local search (HCSN) and also they allow to reuse a great deal of the computations previously done. In the case of ACO-B and HCSST the values for these ratios are higher, because HCSST is considerably faster than HCSN. A more exhaustive study (accuracy, number of statistics and CPU time) for ACO-B and HCSST can be found in [12].

With respect to the efficiency between ACO algorithms, it is clear that the fastest one is ACO-B (between 2.89 and 3.75 times faster than ACO-K2SN). This fact is not surprising, due to the double search space explored by ACO-K2SN and HCSN.
6 Conclusions

In this work the applicability of ACO algorithms to the Bayesian network learning problem have been studied. The contribution of this study lies in the use of the Ant Colony Optimisation metaheuristic to guide the search process and in the comparison between the two classical search spaces: dags and orderings. The use of ACO instead of other metaheuristics allows the algorithm to exploit heuristic knowledge about the problem, together with a simple but efficient form of cooperation between independent agents.

With respect to the experimental results, they are encouraging when comparing with the reference K2 values and with other heuristics specifically designed for the same task [24, 17], and similar for the two ACO approaches studied here.

With respect to the design of the ACO algorithms, it was not a straightforward task

- When searching in the space of orderings, the skeleton of the algorithm is based on the one designed by Dorigo et al. [20] to solve the TSP, but there remain many aspects to be considered. In fact, the search for an appropriate heuristic knowledge to be used, gave rise to a new heuristic algorithm for learning BNs (K2SN). As a consequence of using this (specifically build) new heuristic, the ant-K2SN construction heuristic is able to build the ordering and the network at the same time.

- When searching in the space of dags the approach is different. In this case no adaptation is possible from the TSP ACO-algorithm, and more design decisions were required. In fact, the problem representation is completely new with respect to the previous case, being each internal state of the ants graph a graph itself.

For future research, we want to go deeply into the use of Ant Colony Optimisation for learning Bayesian networks, by refining the proposed algorithms (parameter fitting, other forms of local optimisation, ...). Also alternative ACO approaches could be used, as the $M.A./M.A.$ ant system [33] or the best-worse ant system [18]. More empirical work is also necessary to definitely establish the validity of our approach.

Another interesting aspect to be taken into account is the possibility of working with 'a priori' expert knowledge. We think ants can cope with at least two different cases: constraints on the orderings and presence/absence of some relations (arcs) in the network.

Acknowledgments

This work has been partially supported by the Spanish Ministerio de Ciencia y Tecnología (under projects TIC2001-2973-C05-01 and TIC2001-2973-C05-05) and by the Consejería de Ciencia y Tecnología de la JCCM (under project PBC-02-002). We want to thank to Thomas Stuetzle for his useful comments and suggestions.
References


Ant K2SN

1.- VISITED = \emptyset; TOVISIT = VARIABLES.
2.- last = 0
3.- While \(|TOVISIT| > 0\) do
   1. \(j = \text{last}\)
   2. If \((q \leq q_0)\) \((q \in U[0,1])\)
      /\* exploitation /\*
      (a) \(\text{max} = -\infty\)
      (b) For \(i = 1\) to \(|TOVISIT|\) do
          i. Let \(x_i \in TOVISIT\)
          ii. \(Pa(x_i) = K2(x_i, VISITED)\)
          iii. If \((\tau_{ji} * 1/|f(x_i|Pa(x_i))|^{\beta}) > \text{max} \)
                then
                   \(\text{max} = (\tau_{ji} * 1/|f(x_i|Pa(x_i))|^{\beta}), x = x_i\).
   3. If \((q > q_0)\)
      then /\* exploration /\*
      (a) For \(i = 1\) to \(|TOVISIT|\) do
          i. Let \(x_i \in TOVISIT\)
          ii. \(Pa(x_i) = K2(x_i, VISITED)\)
          iii. \(\text{prob}[i] = (\tau_{ji} * 1/|f(x_i|Pa(x_i))|^{\beta})\)
      (b) Normalise prob, \(x_\ast = \text{draw(prob)}\), and \(x = x_\ast\)
   4. Perform local pheromone update to \((\text{index} \ast f(x), \text{last})\).
   5. \(\text{last} \leftarrow \text{index} \ast f(x)\)
   6. \(\text{VISITED} = \text{VISITED} \cup \{x\}\)
   7. \(\text{TOVISIT} = \text{TOVISIT}\ \{x\}\)

Figure 1: Algorithm used by an ant for the construction of an ordering and a network structure
1. Initialization:

(a) for $i = 1$ to $n$ do: $Pa(x_i) = \emptyset$

(b) for $i = 1$ and $j = 1$ to $n$ do:

if ($i \neq j$) then $\eta_{ij} = f(x_i, x_j) - f(x_i, \emptyset)$

2. Loop:

(a) repeat

i. Select two indices $i$ and $j$ by using the probabilistic transition rule.

ii. if ($\eta_{ij} > 0$) then $Pa(x_i) = Pa(x_i) \cup \{x_j\}$

iii. $\eta_{ij} = -\infty$

iv. for all $x_a \in Ancestors(x_j) \cup \{x_j\}$ and $x_b \in Descendants(x_i) \cup \{x_i\}$

do: $\eta_{ab} = -\infty$.

v. for $k = 1$ to $n$ do:

if ($\eta_{ik} > \eta_{ij}$) then $\eta_{ik} = f(x_i, Pa(x_i) \cup \{x_k\}) - f(x_i, Pa(x_i))$

vi. $\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0$

until $\forall i, j$ ($\eta_{ij} \leq 0$ or $\eta_{ij} = -\infty$).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{AntB.png}
\caption{Structure of Ant B}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{ALARM.png}
\caption{ALARM network.}
\end{figure}
Figure 4: INSURANCE network.