# *Using ant colony optimisation in learning Bayesian network equivalence classes*

Shen, Qiang; Daly, Ronan; Aitken, Stuart

*Publication date:*
2006

*Citation for published version (APA):*
Shen, Q., Daly, R., & Aitken, S. (2006). *Using ant colony optimisation in learning Bayesian network equivalence classes*. 111-118. http://hdl.handle.net/2160/437

# Using Ant Colony Optimization in Learning Bayesian Network Equivalence Classes

**Rónán Daly**
School of Informatics
University of Edinburgh
Edinburgh, EH8 9LE
*Ronan.Daly@ed.ac.uk*

**Qiang Shen**
Department of Computer Science
University of Wales, Aberystwyth
Aberystwyth, SY23 3DB
*qqs@aber.ac.uk*

**Stuart Aitken**
School of Informatics
University of Edinburgh
Edinburgh, EH8 9LE
*stuart@aiai.ed.ac.uk*

## Abstract

Bayesian networks are a useful tool in the representation of uncertain knowledge. This paper proposes a new algorithm to learn the structure of a Bayesian network. It does this by conducting a search through the space of equivalence classes of Bayesian networks using Ant Colony Optimization (ACO). To this end, two novel extensions of traditional ACO techniques are proposed and implemented. Firstly, multiple *types* of moves are allowed on the ACO construction graph. Secondly, moves can be given in terms of arbitrary identifiers. The algorithm is implemented and tested. The results show that ACO performs better than a greedy search whilst searching in the space of equivalence classes.

## 1 Introduction

The task of learning Bayesian networks from data has, in a relatively short amount of time, become a mainstream application in the process of knowledge discovery and model building (Heckerman et al., 1995b; Friedman, 2004). The reasons for this are many.

For one, the model built by the process has an intuitive feel — this is because a Bayesian network consists of a directed acyclic graph (DAG), with conditional probability tables annotating each node. Each node in the graph represents a variable of interest in the problem domain and the arcs can (with some caveats) be seen to represent causal relations between these variables — the nature of these causal relations is governed by conditional probability tables associated with each node/variable. An example Bayesian network is shown in Figure 1.

Another reason for the usefulness of Bayesian networks is that aside from the visual attractiveness of the model, the underlying theory is quite well understood and has a solid foundation. A Bayesian network can be seen as a factorisation of a joint probability distribution, with the conditional probability distributions at each node making up the factors and the graph structure making up their method of combination. Because of this equivalence, the network can answer any probabilistic question regarding the variables modelled, that is put to it.

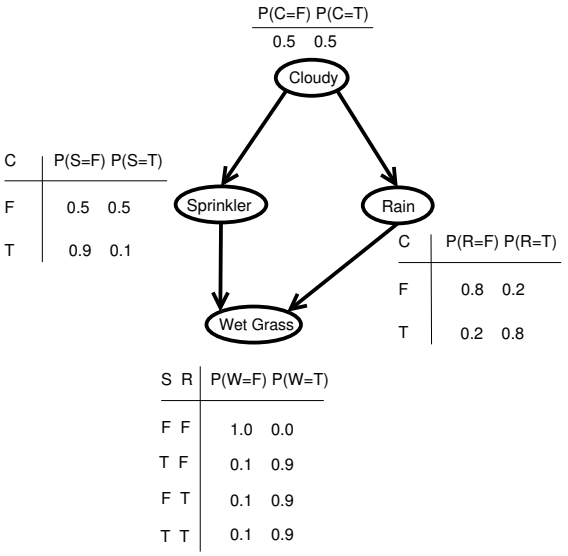


Figure 1: An example Bayesian network

Finally, the popularity of Bayesian networks has been increased by the accessibility of methods to query the model and learn both the structure and parameters of the network. It has been shown that inference in Bayesian networks is NP-Complete (Dagum & Luby, 1993; Shimony, 1994), but approximate methods have been found to perform this operation in an acceptable amount of time. Learning the structure of Bayesian networks is also NP-Complete (Chickering, 1996a), but here too, methods have been found to render this operation tractable. These include greedy search, iterated hill climbing and simulated annealing (Chickering et al., 1995). Recently however, other heuristics have become popular with the problem of combinatorial optimization in high dimensional spaces. These include approaches such as tabu search, genetic algorithms and — the approach that this paper will investigate — Ant Colony Optimization.

Ant Colony Optimization (ACO) is a fairly recent, so called metaheuristic, that is used in the solution of combinatorially hard problems (Dorigo & Stützle, 2004). It is an iterated, stochastic technique that is biased by the results of previous iterations (Birattari et al., 2002). The method is modelled on the behaviour

of real-life ants whilst foraging for food.

Many ants secrete a pheromone trail that is recognisable by other ants and which positively biases them to follow that trail, with a stronger trail meaning it is more likely to be biased. Over time this pheromone trail evaporates. When hunting for food, an ant's behaviour is to randomly walk about, perhaps by following a pheromone trail, until it finds some food. It then returns in the direction from whence it came. Because the strength of the trail is a factor in choosing to follow it, if an ant is faced with two or more pheromone trails to choose from, it will tend to choose the trails with the highest concentration of pheromone.

With these characteristics, in a situation where there are multiple paths to a food source, ants generally follow the shortest path. This can be explained as follows. Assuming ants start from a nest and no pheromone trails are present, they will randomly wander until they reach a food source and then return home, laying pheromone on the way back. The ant that chooses the shortest path to the food source will return home the quickest, which means their pheromone trail will have the highest concentration, as more pheromone is laid per unit of time. This stronger trail will cause other ants to prefer it over longer trails, who will leave their own pheromone on this short trail, thereby providing a reinforcing behaviour to choose this trail over others.

Ant Colony Optimization as a computing technique is roughly modelled on this behaviour. Artificial ants walk around a graph where the nodes represent pieces of a solution. They continue this until a complete solution is found. At each node, a choice of the next edge to traverse is made, depending on a pheromone value associated with the edge and a problem specific heuristic. After a number of ants have performed a traversal of the graph, one of the best solutions is chosen and the pheromone on the edges that it took is increased, relative to the other edges. This biases the ants towards choosing these edges in future iterations. The search stops when a problem specific criterion is reached. This could be stagnation in the quality of solutions or the passage of a fixed amount of time.

This paper will seek to use the ACO technique in learning Bayesian networks. Specifically, it will be used to learn an equivalence class of Bayesian network structures. To this end, the rest of this paper will be structured in the following fashion.

Firstly, there will be a more in-depth study of the problem of searching for an optimum Bayesian network, in both the space of Bayesian networks themselves and of equivalence classes of Bayesian networks. Then, a new method of formulating a search for a Bayesian network structure in terms of the ACO metaheuristic will be introduced. Next, results of tests against previous techniques will be discussed and finally, any conclusions and possible future directions will be stated.

## 2 Searching for a Bayesian Network Structure

There are, in general, three different methods used in learning the structure of a Bayesian network from data. The first finds conditional independencies in the data and then uses these conditional independencies to produce the structure (Spirtes et al., 2000). The second uses dynamic programming and optionally, clustering, to construct a DAG (Ott et al., 2004; Ott & Miyano, 2003). The third method — which we will be dealing with — defines a search on the space of Bayesian networks. This method uses a scoring function defined by the implementer, that says relatively how good a network is compared to others. Before discussing how this method works, some definitions and notation will be introduced.

A graph $\mathcal{G}$ is given as a pair $(V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices or nodes in the graph and $E$ is the set of edges or arcs between the nodes in $V$. A directed graph is a graph where all the edges have an associated direction from one node to another. A directed acyclic graph or DAG, is a directed graph, without any cycles, i.e. it is not possible to return to a node in the graph by following the direction of the arcs.

A Bayesian network on a set of variables $V = \{v_1, \ldots, v_n\}$ is a pair $(\mathcal{G}, \Theta)$, where $\mathcal{G} = (V, E)$ is a DAG and $\Theta = \{\theta_1, \ldots, \theta_n\}$ is a set of conditional probability distributions, where each $\theta_i$ is associated with each $v_i$.

In learning a Bayesian network from data, both the structure $\mathcal{G}$ and parameters $\Theta$ must be learned, normally separately. In the case of complete multinomial data, the problem of learning the parameters is easy, with a simple closed form formula for $\Theta$ (Heckerman, 1995). However, in the case of learning the structure, no such formula exists and other methods are needed. In fact, learning the structure is an NP-Hard problem and consequently enumeration and test of all network structures is not likely to succeed. With just ten variables there are roughly $10^{18}$ possible DAGs, which leaves non-exact methods as possibly the only tractable solution.

In order to create a space in which to search through, three components are needed. Firstly all the possible solutions must be identified as the set of states in the space. Secondly a representation mechanism for each state is needed. Finally a set of operators must be given, in order to move from state to state in the space.

Once the search space has been defined, two other pieces are needed to complete the search algorithm, a scoring function which evaluates the "goodness of fit" of a structure with a set of data and a search procedure that decides which operator to apply, normally using the scoring function to see how good a particular oper-

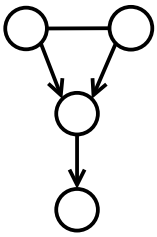

Figure 2: An example of a PDAG

ator application might be. An example of a search procedure is greedy search, that at every stage applies the operator that produces the best change in the structure, according to the scoring function. As for the scoring function, various formulæ have been found to see how well a DAG fits a data sample, e.g. by giving the relative posterior probability (Heckerman et al., 1995a),

$$\mathrm{S}\left(\mathcal{G}, D\right) = \mathrm{P}\left(\mathcal{G}, D\right) = \mathrm{P}\left(D|\mathcal{G}\right)\mathrm{P}\left(\mathcal{G}\right)$$

or using a large-sample approximation to the likelihood such as the Bayesian information criterion (Heckerman, 1995), given as

$$\log \mathrm{P}\left(D|\mathcal{G}\right) \approx \log \mathrm{P}\left(D|\mathcal{G}, \hat{\theta}\right) - \frac{d}{2}\log N,$$

where $\hat{\theta}$ is the maximum likelihood estimate of the parameters, $d$ is the dimension or number of variables in the network and $N$ is the number of data.

Traditionally, in searching for a Bayesian network structure, the set of states was the set of possible Bayesian network structures, the representation was a DAG and the set of operators were various small local changes to a DAG, e.g. adding, removing or reversing an arc. This is possible because of the decomposition properties of most score functions,

$$\mathrm{S}\left(\mathcal{G}, D\right) = \sum_{i=1}^{n} \mathrm{s}\left(v_i, \mathrm{Pa}^{\mathcal{G}}(v_i)\right),$$

which means the score of a particular DAG is the sum of the scores of each node given it's parents. Successful application of the operators was also dependent on the changed graph being a DAG, i.e. that no cycle was formed in applying the operator. In keeping with the terminology used by Chickering this space shall be called B-space (Chickering, 1996b).

## 3 Searching in the Space of Equivalence Classes

According to many scoring criteria, there are DAGs that are equivalent to one another, in the sense that they will produce the same score as each other. Looking at this in more depth, it is found that these equivalent DAGs produce the same set of independence constraints as each other, even though the structures are different. Independence constraints show how a set of variables are influenced or dependent on another set of variables, given a certain third set of variables. These constraints can be checked by analysing the DAG for certain structures. It turns out, according to a theorem by Verma and Pearl, that two DAGs are equivalent iff they have the same skeletons and the same v-structures (Verma & Pearl, 1991). By skeleton, it is meant the undirected graph that results in undirecting all edges in a DAG and by v-structure, (sometimes referred to as a morality) it is meant a head-to-head meeting of two arcs, where the tails of the arcs are not joined. From this notion of equivalence, a class of DAGs that are equivalent to each other can be defined, notated here as $Class(\mathcal{G})$.

Because of this apparent redundancy in the space of DAGs, attempts have been made to conduct the search for Bayesian network structures in the space of equivalence classes of DAGs (Chickering, 1996b, 2002a; Munteanu & Bendou, 2001). The search set of this space is the set of equivalence classes of DAGs and will be referred to as E-space. To represent the states of this set, a different type of structure is used, known as a partially directed acyclic graph (PDAG). A PDAG (an example of which is shown in Figure 2) is a graph that contains both undirected and directed edges and that contains no directed cycles and will be notated herein as $\mathcal{P}$. Again, the equivalence class of DAGs corresponding to a PDAG is denoted as $Class(\mathcal{P})$, with a DAG $\mathcal{G} \in Class(\mathcal{P})$ iff $\mathcal{G}$ and $\mathcal{P}$ have the same skeleton and same set of v-structures. Related to this is the idea of a *consistent extension*. If a DAG $\mathcal{G}$ has the same skeleton and the same set of directed edges as a PDAG $\mathcal{P}$ then it is said that $\mathcal{G}$ is a consistent extension of $\mathcal{P}$. Not all PDAGs have a DAG that is a consistent extension of itself. If a consistent extension exists, then it is said that the PDAG *admits* a consistent extension. Only PDAGs that admit a consistent extension can be used to represent an equivalence class of DAGs and hence a Bayesian network.

Directed edges in a PDAG can be either compelled, or made to be directed that way, whilst others are reversible, in that they could be undirected and the PDAG would still represent the same equivalence class. From this idea, we can define a completed PDAG (CPDAG), where every undirected edge is reversible in the equivalence class and every directed edge is compelled in the equivalence class. We shall denote a CPDAG as $\mathcal{P}^{\mathcal{C}}$. It can be shown that there is a one-to-one mapping between a CPDAG $\mathcal{P}^{\mathcal{C}}$ and $Class(\mathcal{P}^{\mathcal{C}})$. With this isomorphism, a state in the space of equivalence classes can be represented by a CPDAG.

With this representation of equivalence classes of Bayesian network structures and a set of operators that modify the CPDAGs which represent them (e.g. insert an undirected arc, insert a directed arc etc.), a search procedure can proceed. But one might ask why go to the bother of this type of search. For one, an equiv-

alence class can represent many different DAGs in a single structure. Search in the space of DAGs often moves between states with the same equivalence class and so, in a sense, is wasted effort. This also affects the connectivity of the search space, in that the ability to move to a particular neighbouring equivalence class can be constrained by the particular representation given by a DAG.

There is also the problem given by the prior probability used in the scoring function. Whilst searching through the space of DAGs, certain equivalence classes can be over represented by this prior, because there are many more DAGs contained in the class.

These concerns have motivated researchers, with the results that recent implementations of algorithms that search through the space of equivalence classes have produced results that show a marked improvement in execution time and a small improvement in learning accuracy, depending on the type of data set (Chickering, 2002a,b).

### 3.1 Techniques for Searching through Equivalence Classes

Note that here we refer to a *move* as an application of an operator to a particular state in the search space.

To be able to conduct a search through the space of equivalence classes, a method must be able to find out whether a particular move is valid and if valid, how good that move is. These tasks are relatively easy whilst searching through the space of DAGs — a check whether a move is valid is equivalent to a check whether a move keeps a DAG acyclic. The goodness of such a move is found out by using the scoring function, but rather then scoring each neighbouring DAG in the search space, the decomposability of most scoring criterion can be taken advantage of, with the result that only nodes whose parent sets have changed need to be scored.

However, this task of checking move validity and move score is not as easy in the space of equivalence classes. For one, instead of just checking for cycles, checks also have to be made so that unintended v-structures are not created in a consistent extension of a PDAG. Scoring a move also creates difficulties, as it is hard to know what extension and hence what changes in parent sets of nodes will occur, without actually performing this extension. Also, a local change in a PDAG *might* make a non-local change in a corresponding extension and so force unnecessary applications of the score function.

These problems were voiced as concerns by Chickering (1996b). In this paper he performs validity checking of moves by trying to obtain a consistent extension of the resulting PDAG — if none exists then the move is not valid. Scoring the move was achieved by scoring the changed nodes in the consistent extension given. These methods were very generic, but resulted in a significant slowdown in algorithm execution, compared to search in the space of DAGs.

To alleviate this problem, authors proposed improvements that would allow move validity and move score to be computed without needing to obtain a consistent extension of the PDAG (Munteanu & Bendou, 2001; Chickering, 2002a). This was done by defining an explicit set of operators, with each operator having a validity test and corresponding score change function, that could be calculated on the PDAG. These changes resulted in a speedup of the execution time of the algorithm, with the result that search in the space of equivalence classes of Bayesian networks became competitive with search in the space of Bayesian networks.

## 4 Ant Colony Optimization

Since the inception of it's present form in 1992 (Dorigo, 1992), ACO has been a successfully applied to many combinatorially hard problems (Maniezzo & Colorni, 1999; Bullnheimer et al., 1999; Gambardella & Dorgio, 2000; Stützle, 1998; Costa & Hertz, 1997). The combination of reinforcement of good solutions, multiple iterations and the ability to introduce problem dependent heuristics and local search has meant world class performance on a number of problems. These include the sequential ordering problem, the vehicle routing problem, the bin-packing problem and many more.

Perhaps because of it's physical resemblance to ants walking around, the traveling salesman problem was one of the first to be studied using an algorithm called Ant System. This introduced most of the fundamental ideas used in ACO today. One of these is the random proportional rule,

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}$$

This says that the probabilities of taking a particular path is proportional to the pheromone on that path and heuristic desirability of that path. After all the ants have constructed their tour, pheromone is then evaporated from the paths,

$$\tau_{ij} \leftarrow (1-\rho)\,\tau_{ij}, \quad \forall\,(i,j) \in L,$$

which says that all paths evaporate $\frac{\rho}{1}\%$ of their pheromone. Finally all ants deposit pheromone on the path they traversed as below

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \quad \forall\,(i,j) \in L$$

Here, $\Delta\tau_{ij}^k$ is the amount of pheromone ant $k$ deposits on the path and is given by

$$\Delta\tau_{ij}^k = \begin{cases} 1/C^k, & \text{if arc } (i,j) \text{ belongs to } T^k \\ 0, & \text{otherwise} \end{cases},$$

where $C^k$ is the cost or length of the tour $T^k$ made by ant $k$.

Advances in ACO generally use the framework described above. Some of the newer systems which generally have improved on Ant System are the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) (Stützle & Hoos, 2000) and Ant Colony System (ACS) (Dorigo & Gambardella, 1997), which have mechanisms to favour exploitation of good solutions over exploration of other possibilities.

## 5 Using Ant Colony Optimization in Learning an Equivalence Class

To date, many state-based search algorithms that create a Bayesian network structure have relied on simple techniques such as greedy-based searches. These can produce good results, but have the ever prevalent problem of getting caught at local minima. More sophisticated heuristics have been applied whilst learning in B-space, such as iterated hill climbing and simulated annealing (Chickering et al., 1995), but so far, none of these have been applied to E-space.

This paper seeks to apply the ACO metaheuristic to E-space. To this end, two apparently novel extensions are made to the basic metaheuristic. The first is to allow multiple *types* of moves. This is to allow more than one operator to be used in traversing the state space. This is needed, because in general, more than one type of operator is used whilst searching in E-space. The second is to allow the pheromone to be accessed by arbitrary values — normally it is accessed by a single index if it is associated with solutions components $(\tau_i)$, or two indices if it is associated with connections on the construction graph $(\tau_{ij})$. Again this is needed because of the operators used in E-space — the MakeV operator takes three nodes as arguments.

The algorithm given is based, in large part, on the work of de Campos et al. (2002). In the work they performed, ACO was applied to learning Bayesian networks. This current work differs in that it searches in E-space, uses more than one operator (add an arc) and doesn't constrain itself to using matrices to store pheromone. The algorithm, ACO-E, is shown in Algorithm 1. A description of the algorithm working is given as follows. Firstly, the various equations implementing the parts of the ACO framework will be given and then the algorithm itself will be illustrated.

The random proportional choice rule, used to select which state to proceed to next, is given as

$$p_m = \frac{\tau_m \left[\eta_m\right]^\beta}{\sum_{\mu \in M} \tau_\mu \left[\eta_\mu\right]^\beta}, \quad \forall m \in M \qquad (1)$$

Here, $p_m$ is the probability that a particular move $m$ will be taken and $M$ is the set of possible moves from the current state, given a set of operators $O$. The algorithm also makes use of a technique used in ACS, which means that the random choice rule is only used some of the time; the rest of the time the best move is made. This other choice rule is given as

$$m \leftarrow \begin{cases} \arg\max_{m \in M} \tau_m \left[\eta_m\right]^\beta, & \text{if } q \leq q_0 \\ \text{random proportional,} & \text{otherwise} \end{cases}$$

Similarly, there is a local evaporation rule, as in ACS, whereby pheromone is removed from a path as an ant traverses it

$$\tau_m \leftarrow (1-\rho)\,\tau_m + \rho\tau_0$$

and a global pheromone update rule that deposits new pheromone on the best-so-far path

$$\tau_m \leftarrow (1-\rho)\,\tau_m + \rho\left(\text{score}\left(\mathcal{P}^+\right)/n\right)$$

ACO-E takes as input a number of parameters and returns the best PDAG found, according to a scoring criteria, "score". The meaning of the parameters is as follows

$O$ This is a set of operators that can modify the current PDAG state in the search. Examples of these are the ones given by Chickering (2002a), e.g. InsertU, DeleteU, etc.

$t_{max}$ This is the number of iterations of the algorithm to run

$t_{step}$ This is the gap, in iterations, between which local search procedures are run

**Algorithm 1** ACO-E

**Input:** Operators $O$, $t_{max}$, $t_{step}$, $m$, $\rho$, $q_0$, $\beta$, $n$
**Output:** PDAG $\mathcal{P}^+$

```
(P+, Path+) ← greedy(P^empty, Path^empty)
τ0 ← score(P+)/n
for each operator o in O do
  for each possible move m in o on P^empty do
    τm ← τ0
  end for
end for
for t ← 1 to t_max do
  for k ← 1 to m do
    (P^k, Path^k) ← ANT-E(O, q0, ρ, β, τ0)
    if (t mod t_step = 0) then
      (P^k, Path^k) ← greedy(P^k, Path^k)
    end if
  end for
  b ← arg max_{k=1}^{m} score(P^k)
  P^b ← P^k
  if score(P^b) > score(P+) then
    P+ ← P^b
    Path+ ← Path^b
  end if
  for each move m in Path+ do
    τm ← (1 − ρ) τm + ρ (score(P+)/n)
  end for
end for
return P+
```

**Algorithm 2** ANT-E

**Input:** Operators $O$, $\rho$, $q_0$, $\beta$
**Output:** PDAG $\mathcal{P}$, Path $Path$
  Empty PDAG $\mathcal{P}$, Empty path $Path$
  **while** true **do**
    $M \leftarrow$ All possible moves from $\mathcal{P}$ using $O$
    **if** $\max_{m \in M}$ total-score$(m, \beta) \leq 0$ **then**
      return $\mathcal{P}$
    **end if**
    $q \leftarrow$ random number in $[0, 1)$
    **if** $q \leq q_0$ **then**
      $m \leftarrow \arg\max_{m \in M}$total-score$(m)$
    **else**
      $m \leftarrow$ random according to equation 1
    **end if**
    $\tau_m \leftarrow (1 - \rho)\, \tau_m + \rho \tau_0$
    $\mathcal{P} \leftarrow$ apply $m$ to $\mathcal{P}$
    $Path \leftarrow$ append $m$ to $Path$
  **end while**
  return $\mathcal{P}$

$m$ This is the number of ants that run at each iteration

$\rho$ This, a value in $[0, 1]$, is the rate at which pheromone evaporates

$q_0$ This, a value in $[0, 1)$, is a value that gives the preference of exploitation over exploration

$\beta$ This exponent gives the relative importance of the heuristic over the pheromone levels in deciding the chance that a particular trail will be followed

$n$ This is the number of nodes in the PDAG

In starting the algorithm, a greedy search is performed. This gives a starting optimum graph and path from which the search can proceed. Pheromone levels are then initialised to $\tau_0$. The main loop of the algorithm then begins for $t_{max}$ iterations. At each iteration, $m$ ants perform a search, given by algorithm ANT-E, shown in Algorithm 2. Also, every $t_{step}$ iterations, a local search is performed on the PDAGs returned from ANT-E, to try and improve results. After the $m$ ants have traversed the graph, the best graph and path are selected from the current best graph and path and the ones found by each of the ants in the current iteration. Finally, the global pheromone update lays and evaporates pheromone on the best path.

**Algorithm 3** total-score

**Input:** Move $m$, $\beta$
**Output:** Score $s$
**return** $s$ such that $s = \begin{cases} \tau_m \left(\eta_m\right)^\beta & \text{if } \eta_m > 0 \\ 0 & \text{otherwise} \end{cases}$

The ANT-E algorithm creates a PDAG by examining the various states that may be proceeded to from the current state, given a set of operators that may act on the current PDAG. It then selects a new state based on a random-proportional choice rule. The parameters to the function have the same description as the ones to the ACO-E function.

Starting out, the algorithm constructs an empty PDAG. Then at each stage a move is made to a new PDAG, that can be reached by applying one of the operators in $O$. Firstly, a number is given to each move by total-score, shown in Algorithm 3. This number represents a weight given to each move $m$ depending on the current pheromone associated with making that move $\tau_m$, and the heuristic associated with making the move $\eta_m$. This heuristic is given by the decrease in score given by taking that move, lower overall scores meaning better solutions. If there can be no decrease in the score, the ant stops and returns the solution $\mathcal{P}$ and the path followed. Otherwise there is a possible move and the ant decides how to make it. Firstly a random number $q$ is obtained. If it is less than a specified value $q_0$, then the best move is taken. If it is greater than $q_0$, then a random proportional choice is made, with the probability of better moves being higher. After this, a local pheromone update is applied to the path just taken, the path is updated with the new location at the end and the current state is updated to become the new state given by $m$. Note that applying a move to a CPDAG to change state implies the resulting PDAG will be extended to a DAG by a suitable method (e.g. that by Dor & Tarsi (1992)) and the resulting DAG be changed back to a CPDAG. Details can be found in Chickering (2002a).

## 6 Implementation and Experimental Results

In implementing the algorithms given in this paper, care must be taken to avoid long run times. Firstly, caching the score of a node given it's parents is a simple technique that can greatly improve performance. Secondly, caching the results of the validity tests needed to check which moves are applicable at a certain state, can again increase performance dramatically. However this technique is not as easy to implement (Daly et al., 2006).

Care must also be taken in implementing the pheromone for the moves. Traditionally, matrices of values are used, which allow fast access and updating. However in the case of the MakeV operator, which takes three indices, a three dimensional matrix would be needed. This would quickly become infeasible as the problem size grew. Instead a structure such as a map can store this information. If the map is implemented as a tree, entries can be accessed in logarithmic time and if a hash table is used, access can be in constant time.

The results of some basic experiments are shown in Figure 3. 5000 data were generated by sampling the

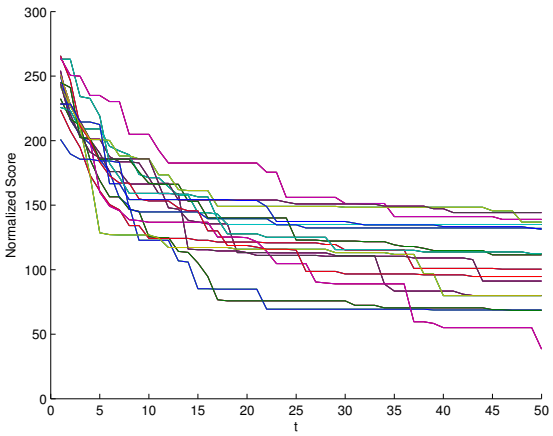


Figure 3: ALARM Network

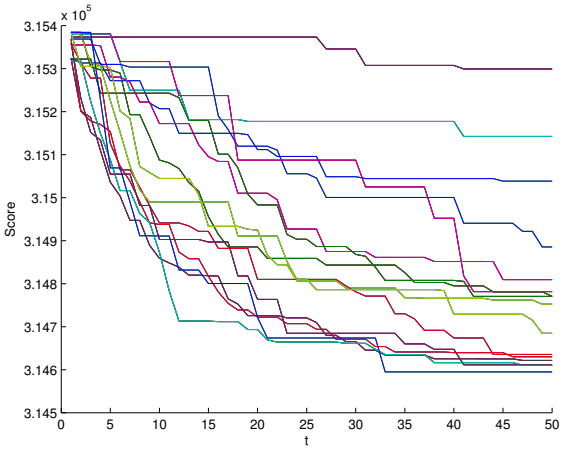


Figure 4: The Insurance Company Benchmark

ALARM network (Beinlich et al., 1989). Here the the algorithm was run with different combinations of $\rho$, $q_0$ and $\beta$. $\rho$ took four values in the range $[0.05, 0.2]$, $q_0$ four values in the range $[0.8, 0.95]$ and $\beta$ four values in the range $[1, 4]$. This gave a total of 64 different combinations of values. This was done so as to see algorithm performance over a wide range of possibilities. For all the experiments, the set of operators $O$ were those given by Chickering (2002a). $t_{max}$ was set to 50, $t_{step}$ to 6 and $m$ to 5. Each combination was run 10 times and the results averaged. The iteration $t$ is shown on the horizontal axis and the normalized score shown on the vertical axis. The score was normalized by subtracting the score of the network which generated the data. This network score can be seen as a baseline against which other network scores can be measured. Each of the lines on the graph represents a different combination of the values $\rho$, $q_0$ and $\beta$. As can be seen, with the progression of time, the algorithm finds progressively better networks. This is in comparison to the greedy search heuristic, whose average score was 273.

Similar results can be seen in Figure 4. This data came from the Insurance Company Benchmark (van der Putten & van Someren, 2004). However, as this data was not generated from a Bayesian network, a baseline score is not available to normalize the scores. That being said, an improvement is seen over a greedy search, as time progresses.

## 7 Conclusions and Future Directions

This paper has introduced a new method for learning the structure of a Bayesian network. It has done this by introducing a method for performing a search in the space of equivalence classes of Bayesian networks using Ant Colony Optimization. In order to implement this search, two novel extensions to ACO were proposed. Firstly different *types* of moves on the ACO construction graph are allowed. Secondly, moves were generalised, to allow moves that needed more than two integers to represent them. Experiments have been run using the proposed algorithm and results show an improvement against a traditional greedy search.

In the future, more tests need to be run on different datasets. Also, comparisons need to be drawn between the algorithm and other heuristic algorithms. Finally, the performance characteristics need to be analysed.

## References

Beinlich, I., Suermondt, H., Chavez, R. & Cooper, G. (1989). The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, 247–256.

Birattari, M., Caro, G.D. & Dorigo, M. (2002). Toward the Formal Foundation of Ant Programming. In *Proceedings of the Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, 188–201. London, UK: Springer-Verlag.

Bullnheimer, B., Hartl, R.F. & Strauss, C. (1999). An Improved Ant System Algorithm for the Vehicle Routing Problem. *Annals of Operations Research* **89**, 319–328.

Chickering, D.M. (1996a). Learning Bayesian Networks is NP-Complete. In D. Fisher & H. Lenz (Eds.), *Learning from Data: Artificial Intelligence and Statistics V*, chapter 12, 121–130. Springer-Verlag.

Chickering, D.M. (1996b). Learning Equivalence Classes of Bayesian Network Structures. In F. Jensen & E. Horvitz (Eds.), *Proceedings of the Twelfth Conference on Uncertainty in Artificial*

*Intelligence*, 150 –157. San Francisco, California: Morgan Kaufmann.

Chickering, D.M. (2002a). Learning Equivalence Classes of Bayesian-Network Structures. *Journal of Machine Learning Research* **2**, 445– 498.

Chickering, D.M. (2002b). Optimal Structure Identification with Greedy Search. *Journal of Machine Learning Research* **3**, 507–554.

Chickering, D.M., Geiger, D. & Heckerman, D. (1995). Learning Bayesian Networks: Search Methods and Experimental Results. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, 112–128.

Costa, D. & Hertz, A. (1997). Ants Can Colour Graphs. *Journal of the Operational Research Society* **48**(3), 295–305.

Dagum, P. & Luby, M. (1993). Approximating Probabilistic Inference in Bayesian Belief Networks is NP-hard. *Artificial Intelligence* **60**(1), 141–154.

Daly, R., Shen, Q. & Aitken, S. (2006). Speeding up the Learning of Equivalence Classes of Bayesian Network Structures. In *Proceedings of the 10th IASTED International Conference on Artificial Intelligence and Soft Computing*. To appear.

de Campos, L.M., Fernández-Luna, J.M., Gámez, J.A. & Puerta, J.M. (2002). Ant Colony Optimization for Learning Bayesian Networks. *International Journal of Approximate Reasoning* **31**(3), 291–311.

Dor, D. & Tarsi, M. (1992). A Simple Algorithm to construct a consistent extension of a partially oriented graph. Technical Report R-185, Cognitive Systems Laboratory, Department of Computer Science, UCLA.

Dorigo, M. (1992). *Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale*. Ph.D. thesis, Politecnico di Milano, Italy.

Dorigo, M. & Gambardella, L.M. (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation* **1**(1), 53–66.

Dorigo, M. & Stützle, T. (2004). *Ant Colony Optimization*. Cambridge, Massachusetts: The MIT Press.

Friedman, N. (2004). Inferring Cellular Networks Using Probabilistic Graphical Models. *Science* **303**(5679), 799–805.

Gambardella, L.M. & Dorgio, M. (2000). An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. *INFORMS Journal on Computing* **12**(3), 237–255.

Heckerman, D. (1995). A Tutorial on Learning with Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research.

Heckerman, D., Geiger, D. & Chickering, D.M. (1995a). Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning* **20**(3), 197–243.

Heckerman, D., Mamdani, A. & Wellman, M.P. (1995b). Real-world Applications of Bayesian Networks. *Communications of the ACM* **38**(3), 24–26.

Maniezzo, V. & Colorni, A. (1999). The Ant System Applied to the Quadratic Assignment Problem. *Knowledge and Data Engineering* **11**(5), 769–778.

Munteanu, P. & Bendou, M. (2001). The EQ Framework for Learning Equivalence Classes of Bayesian Networks. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, 417–424. Washington, DC, USA: IEEE Computer Society.

Ott, S., Imoto, S. & Miyano, S. (2004). Finding Optimal Models for Small Gene Networks. In *Proceedings of the Ninth Pacific Symposium on Biocomputing*, 557–567. World Scientific.

Ott, S. & Miyano, S. (2003). Finding Optimal Gene Networks Using Biological Constraints. *Genome Informatics* **14**, 124–133.

Shimony, S.E. (1994). Finding MAPs for Belief Networks is NP-hard. *Artificial Intelligence* **68**(2), 399–410.

Spirtes, P., Glymour, C. & Scheines, R. (2000). *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning. The MIT Press, 2nd edition.

Stützle, T. (1998). An Ant Approach to the Flow Shop Problem. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing: EUFIT '98*, volume 3, 1560–1564. Aachen, Germany.

Stützle, T. & Hoos, H.H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems* **16**(8), 889–914.

van der Putten, P. & van Someren, M. (2004). A Bias-Variance Analysis of a Real World Learning Problem: The CoIL Challenge 2000. *Machine Learning* **57**(1-2), 177–195.

Verma, T. & Pearl, J. (1991). Equivalence and Synthesis of Causal Models. In P. Bonissone, M. Henriona, L. Kanal & J. Lemmer (Eds.), *Proceedings of the 6th Annual Conference on Uncertainty in Artificial Intelligence*, 255–268. New York: Elsevier.