# *Methods to accelerate the learning of bayesian network structures*

Shen, Qiang; Daly, Ronan

*Publication date:*
2007

*Citation for published version (APA):*
Shen, Q., & Daly, R. (2007). *Methods to accelerate the learning of bayesian network structures*.
http://hdl.handle.net/2160/421

# Methods to Accelerate the Learning of Bayesian Network Structures

**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*

## Abstract

Bayesian networks have become a standard technique in the representation of uncertain knowledge. This paper proposes methods that can accelerate the learning of a Bayesian network structure from a data set. These methods are applicable when learning an equivalence class of Bayesian network structures whilst using a score and search strategy. They work by constraining the number of validity tests that need to be done and by caching the results of validity tests. The results of experiments show that the methods improve the performance of algorithms that search through the space of equivalence classes multiple times and that operate on wide data sets. The experiments were performed by sampling data from six standard Bayesian networks and running an ant colony optimization algorithm designed to learn a Bayesian network equivalence class.

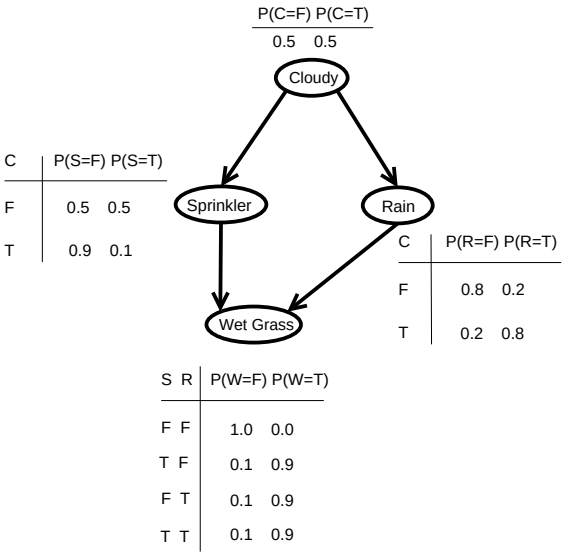



Figure 1: An example Bayesian network

## 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., 1995; 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 popularity 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.

In addition, 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) and other, more sophisticated strategies, such as ant colony optimization (de Campos et al., 2002).

However, in general, more complicated strategies imply more computationally complex algorithms and hence longer running times. This can be especially prevalent in algorithms that require multiple restarts and data with a high dimensionality. This paper will investigate strategies to speed up the learning

of Bayesian networks. Specifically, it will be used to speed up the learning of 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, new methods of speeding up this learning process 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 is to be dealt with here — defines a search on the space of Bayesian networks. This method uses a scoring function defined by the implementer, which 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. For illustration, the graph in figure 1 is a DAG.

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 (Chickering, 1996a). 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 operator 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.

One of these is given by computing the posterior probability of a structure $\mathcal{G}$ given a sample of data $D$, i.e.

$$\mathrm{S}(\mathcal{G}, D) = \mathrm{P}(\mathcal{G}|D) = \frac{\mathrm{P}(D|\mathcal{G})\mathrm{P}(G)}{\mathrm{P}(D)}. \qquad (1)$$

Here the value $\mathrm{P}(D)$ is a constant across all network structures and so can be ignored. This gives $\mathrm{S}(\mathcal{G}, D) = \mathrm{P}(\mathcal{G}, D) = \mathrm{P}(D|\mathcal{G})\mathrm{P}(G)$, i.e. the relative posterior probability.

The likelihood term above can take many forms. One popular method is called the Bayesian Dirichlet (BD) metric. Here,

$$\mathrm{P}(D|\mathcal{G}) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})} \qquad (2)$$

In this formula, there are $n$ variables in the graph, so the first product is one over each variable. There are $q_i$ configurations of the parents of node $i$, so the second product is over all possible parent configurations, i.e. the cross product of the number of possible values each parent variable can take on. Each variable $i$ can take on one of $r_i$ possible values. The value $N_{ijk}$ is the number of times the configuration where $i = k$ and the parents of $i$ are in configuration $j$, comes up in the data sample $D$. $N_{ij}$ is given as $\sum_{k=1}^{r_i} N_{ijk}$, i.e. the sum of $N_{ijk}$ over all possible values that $i$ can take on. With $N'_{ij} = \sum_{i=1}^{r_i} N'_{ijk}$, the values $N'_{ijk}$ are given as parameters that give different variants of the BD metric. E.g. if $N'_{ijk}$ is set to 1 the K2 metric results, as given by Cooper & Herskovits (1992). With $N'_{ijk}$ set to $N'/(r_i \cdot q_i)$ (where $N'$, known as the equivalent sample size is a measure of the confidence in the prior value $\mathrm{P}(\mathcal{G})$), the BDeu metric results which was proposed by Buntine (1991).

The prior value $\mathrm{P}(\mathcal{G})$ is a measure of how probable a particular structure is before any data is seen. These values can often be hard to estimate and are often given as uniform over all possible network structures, possibly favouring structures with less arcs.

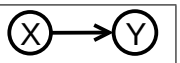
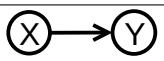
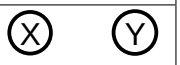
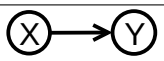
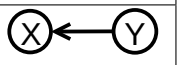
| Operator | Before | After |
|---|---|---|
| Insert_Arc(X,Y) |  |  |
| Delete_Arc(X,Y) |  |  |
| Reverse_Arc(X,Y) |  |  |

Table 1: Basic Modification Operators

Other forms used for the scoring function are $\mathrm{S}(\mathcal{G}, D) = \log \mathrm{P}(D|\mathcal{G}, \hat{\theta}) - \frac{d}{2} \log N$, known as the Bayesian information criterion (BIC) (Schwarz, 1978) and $\mathrm{S}(\mathcal{G}, D) = \log \mathrm{P}(D|\mathcal{G}, \hat{\theta}) - d$, known as the Akaike Information Criterion (AIC) (Akaike, 1974). In these models, the parameter $\hat{\theta}$ gives the maximum likelihood estimate of the likelihood, $d$ is the number of free parameters in the structure and $N$ is the number of samples in the data $D$.

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, as illustrated in table 1. This is possible because of the decomposition properties of most score functions,

$$\mathrm{S}(\mathcal{G}, D) = \sum_{i=1}^{n} \mathrm{s}\left(v_i, \mathrm{Pa}^{\mathcal{G}}\left(v_i\right), D\right), \tag{3}$$

where $s$ is a scoring function that takes a node $v_i$ and the parents of this node in graph $\mathcal{G}$, $\mathrm{Pa}^{\mathcal{G}}\left(v_i\right)$. Popular scoring functions such as the BD metric are decomposable in this manner. If

$$\mathrm{S}(\mathcal{G}, D) = \mathrm{P}(D|\mathcal{G})\mathrm{P}(\mathcal{G}), \tag{4}$$

and since the logarithm is a monotonically increasing function, the scoring function S can be *redefined* to

$$\begin{aligned} \mathrm{S}(\mathcal{G}, D) &= \log\left(\mathrm{P}\left(D|\mathcal{G}\right)\mathrm{P}\left(\mathcal{G}\right)\right) \\ &= \log \mathrm{P}(D|\mathcal{G}) + \log \mathrm{P}(\mathcal{G}). \end{aligned} \tag{5}$$

Now by the likelihood given in equation 2,

$$\log \mathrm{P}(D|\mathcal{G}) = \sum_{i=1}^{n} \log \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})} \tag{6}$$

Therefore, for the BD metric,

$$\mathrm{s}\left(v_i, \mathrm{Pa}^{\mathcal{G}}\left(v_i\right), D\right) = \log \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})} \tag{7}$$

The log function is often distributed into the right hand term of this equation in order to avoid the instability of the gamma function at high values, giving

$$\begin{aligned} \mathrm{s}\left(v_i, \mathrm{Pa}^{\mathcal{G}}\left(v_i\right), D\right) = & \\ \sum_{j=1}^{q_i} & \log \Gamma(N'_{ij}) - \log \Gamma(N'_{ij} + N_{ij}) \\ + \sum_{k=1}^{r_i} & \log \Gamma(N'_{ijk} + N_{ijk}) - \log \Gamma(N'_{ijk}) \end{aligned} \tag{8}$$

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.

# 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 (Verma & Pearl, 1991), that two DAGs are equivalent iff they have the same skeletons and the same v-structures. 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 $Class(\mathcal{G})$.

## 3.1 Representation of equivalence classes

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 $\mathcal{P}$. 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 skele-

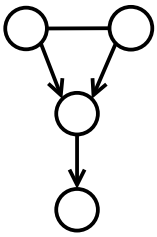


Figure 2: An example of a PDAG

ton 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, a completed PDAG (CPDAG) can be defined, where every undirected edge is reversible in the equivalence class and every directed edge is compelled in the equivalence class. Such a CPDAG will be denoted as 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.

### 3.2 Advantages of searching in E-space

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. However, what reasons are there for pursuing this type of search? For one, an equivalence 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. In particular, 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).

### 3.3 Techniques for searching through equivalence classes

Note that below, a *move* is referred to 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 that paper, validity checking of moves is performed 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 led to 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. An example of one set of these operators is given in table 2. The variables $x$ and $y$ refer to nodes

| Operator | Effect | Validity Tests | Change in Score |
|---|---|---|---|
| InsertU<br>$x - y$ | Add an undirected arc between $x$ and $y$ | 1. Every undirected path from $x$ to $y$ contains a node in $N_{x,y}$<br>2. $\Pi_x = \Pi_y$ | $s\left(y, N_{x,y}^{+x} \cup \Pi_y\right)$<br>$- s\left(y, N_{x,y} \cup \Pi_y\right)$ |
| DeleteU<br>$x - y$ | Delete an undirected arc between $x$ and $y$ | $N_{x,y}$ is a clique | $s\left(y, N_{x,y} \cup \Pi_y\right)$<br>$- s\left(y, N_{x,y}^{+x} \cup \Pi_y\right)$ |
| InsertD<br>$x \rightarrow y$ | Add a directed arc from $x$ to $y$ | 1. Every semi-directed path from $y$ to $x$ contains a node in $\Omega_{x,y}$<br>2. $\Omega_{x,y}$ is a clique<br>3. $\Pi_x \neq \Pi_y$ | $s\left(y, \Omega_{x,y} \cup \Pi_y^{+x}\right)$<br>$- s\left(y, \Omega_{x,y} \cup \Pi_y\right)$ |
| DeleteD<br>$x \rightarrow y$ | Delete a directed arc from $x$ to $y$ | $N_y$ is a clique | $s\left(y, N_y \cup \Pi_y^{-x}\right)$<br>$- s\left(y, N_y \cup \Pi_y\right)$ |
| ReverseD<br>$x \rightarrow y$ | Reverse a directed arc from $x$ to $y$ | 1. Every semi-directed path from $x$ to $y$ that does not include the edge $x \rightarrow y$ contains a node in $\Omega_{y,x} \cup N_y$<br>2. $\Omega_{y,x}$ is a clique | $s\left(y, \Pi_y^{-x}\right)$<br>$+ s\left(x, \Pi_x^{+y} \cup \Omega_{y,x}\right)$<br>$- s\left(y, \Pi_y\right)$<br>$- s\left(x, \Pi_x \cup \Omega_{y,x}\right)$ |
| MakeV<br>$x \rightarrow z \leftarrow y$ | Direct undirected arcs from $x$ and $y$ to $z$ | Every undirected path between $x$ and $y$ contains a node in $N_{x,y}$ | $s\left(z, \Pi_z^{+y} \cup N_{x,y}^{-z+x}\right)$<br>$+ s\left(y, \Pi_y \cup N_{x,y}^{-z}\right)$<br>$- s\left(z, \Pi_z \cup N_{x,y}^{-z+x}\right)$<br>$- s\left(y, \Pi_y \cup N_{x,y}\right)$ |

Table 2: Validity conditions and change in score for each operator

in a graph, so e.g. the InsertU operator takes two nodes as arguments, $x$ and $y$. It can be seen that all the operators take two arguments, except MakeV, which takes three arguments. Each operator also has a set of validity tests that must be passed in order for the application of the operator with its particular arguments to be valid. Finally, the score difference between the old and new PDAGs is given in the last column. In this table, $\Pi_x$ is the parent set of node $x$, $N_x$ is the neighbour set of node $x$, $N_{x,y}$ is the set of shared neighbours of nodes $x$ and $y$ and $\Omega_{x,y}$ is the set of parents of $x$ that are neighbours of $y$. Also, as a convenience, $M^{+x}$ is notation for $M \cup \{x\}$ and $M^{-x}$ is notation for $M \setminus \{x\}$.

## 4 Accelerating the learning process

Whilst the execution time of searching for equivalence classes of Bayesian networks has decreased, it still remains quite high for problem instances with many variables. This is especially so if the search algorithm needs multiple traversals through the search space. Therefore, a typical method of speeding up performance is to cache the values of the score function — this normally gives a large cut in execution time. With this situation, multiple identical runs over a typical greedy search using the operators given by Chickering (in Table 2) were analysed. From this analysis, it was found that much of the time was spent computing two main quantities.

In the first run, the dominant factor was the time needed to compute the values given by the score function. However, in the succeeding runs, it was found that practically all the execution time was used in calculating the validity tests for the various operators. This was because values given by the score function had been cached. In particular, checking the validity conditions for the operators InsertU, InsertD and MakeV was taking the most time. Upon further analysis, it was seen that checking the "path" condition in each of these operators was the main culprit. In order to reduce this time taken, two new methods were examined.

### 4.1 Reducing the number of checked nodes

As shown in Table 2, five of the operators take two nodes as parameters and one takes three nodes. In the naïve case, where $n = |V|$, the number of nodes in the graph, this would mean $O(n^2)$ and $O(n^3)$ checks. If the validity test includes a "path" condition, this will take time in $O(n + e)$, where $e$ is the number of edges on the graph. This could mean time in $O(n^3 + n^2 e)$ and $O(n^4 + n^3 e)$ to check the operators.

However, looking at the validity tests more closely, it can be seen that not all combinations of nodes need be checked. In particular, given a node, we can find a

| Operator | Validity Tests | VALID-NODES |
|---|---|---|
| InsertU<br>$x - y$ | 1. Every undirected path from $x$ to $y$ contains a node in $N_{x,y}$<br>2. $\Pi_x = \Pi_y$ | 1. $V \setminus X_N \cup \text{CHECK}\left(N_{N_x} \setminus (N_x \cup \{x\})\right)$<br>2. $\begin{cases} \{\xi \mid \xi \in V, \xi \neq x, \lvert\Pi_\xi\rvert = 0\} & \text{if } \lvert\Pi_x\rvert = 0 \\ \text{CHECK}\left(\Xi_{\Pi_x}\right) & \text{otherwise} \end{cases}$ |
| DeleteU<br>$x - y$ | $N_{x,y}$ is a clique | $\text{CHECK}\left(N_x\right)$ |
| InsertD<br>$x \rightarrow y$ | 1. Every semi-directed path from $y$ to $x$ contains a node in $\Omega_{x,y}$<br>2. $\Omega_{x,y}$ is a clique<br>3. $\Pi_x \neq \Pi_y$ | 1. $V \setminus X_{N\Pi} \cup \text{CHECK}\left(N_{\Pi_x} \setminus \Pi_x\right)$<br>2. $V \setminus X_{N\Pi} \cup \text{CHECK}\left(N_{\Pi_x} \setminus \Pi_x\right)$<br>3. $\begin{cases} \{\xi \mid \xi \in V, \lvert\Pi_\xi\rvert \neq 0\} & \text{if } \lvert\Pi_x\rvert = 0 \\ \text{CHECK}\left(\Xi_{\Pi_x}\right) & \text{otherwise} \end{cases}$ |
| DeleteD<br>$x \rightarrow y$ | $N_y$ is a clique | $\text{CHECK}\left(\Xi_x \cup \{x\}\right)$ |
| ReverseD<br>$x \rightarrow y$ | 1. Every semi-directed path from $x$ to $y$ that does not include the edge $x \rightarrow y$ contains a node in $\Omega_{y,x} \cup N_y$<br>2. $\Omega_{y,x}$ is a clique | 1. $\text{CHECK}\left(\Xi_x\right)$<br>2. $\text{CHECK}\left(\Xi_x\right)$ |
| MakeV<br>$x \rightarrow z \leftarrow y$ | Every undirected path between $x$ and $y$ contains a node in $N_{x,y}$ | $\text{CHECK}\left(N_{N_x} \setminus (N_x \cup \{x\})\right)$ |

Table 3: Validity conditions and set of valid nodes for a node $x$

subset of the nodes $V$ that are valid and a subset that need to be checked by the original conditions.

Table 3 shows for each operator, the original validity tests used, and the set of nodes for which this test is valid. In this table, some extra notation is used. CHECK is a function that uses the original validity test. $\Xi$ is used to refer to the children of a node. $X_{N|\Xi|\Pi}$ is used to refer to those nodes that can be reached from node $x$ by following neighbours $(N)$, children $(\Xi)$ or parents $(\Pi)$. E.g. $X_{N\Xi}$ are those nodes reachable by following the neighbours and children from node $x$. Finally when the notation $N_{\Pi_x}$ is used it means the union over the neighbours of the parents of $x$, i.e. $\bigcup_{\pi \in \Pi_x} N_\pi$.

From looking at Table 3, it can be seen that the number of validity checks for a given node $x$ is now bounded by e.g. $N_{N_x}$, i.e. by the number of nodes that are of distance 2 from $x$. If the number of parents, children and neighbours a node can have is given an upper bound $k$ (as is normally the case) then there are at most $k^2$ checks. This means the number of times an operator now needs to be checked is in $O(nk^2)$ as opposed to $O(n^2)$. This behaviour should lead to a speed up of validity checking, especially for large values of $n$.

### 4.2 Caching

Looking again at the behaviour of a search though the problem space, it can be seen that most moves affect only a subset of the nodes $V$. As an example, if node $x$ is not connected to node $y$ or $z$ then adding an undirected arc between $y$ and $z$ will not affect the validity of adding an undirected arc from $x$ to $y$ or $z$. This behaviour can be taken advantage of, by caching the values of validity tests for particular moves. Once this is done, a method needs to be found to update the cache after a particular move, by removing invalid and adding new valid moves. One particular procedure is given in Algorithm 1. An explanation of the algorithm is as follows.

**Algorithm 1** UPDATE-CACHE

**Input:** PDAG $\mathcal{P}^{new}$, $\mathcal{P}^{old}$, Operators $O$, Cache $Cache$
**Output:** Cache $Cache$

```
C ← CHANGED-NODES(P^new, P^old)
for each operator o ∈ O do
  for each changed node c ∈ C do
    Check ∪= CHECK-NODES (o, c, P^old)
    Check ∪= CHECK-NODES (o, c, P^new)
  end for
  for each node x ∈ Check do
    Cache \= CACHE-VALID(o, x, Cache)
    valid = VALID-NODES(o, P^new, x)
    Cache ∪= ⟨o, x, valid⟩
  end for
end for
return Cache
```

| | Alarm | Barley | Diabetes | HailFinder | Mildew | Win95pts |
|---|---|---|---|---|---|---|
| Original | $3.560 \times 10^3$ | $8.5345 \times 10^3$ | $3.8889 \times 10^3$ | $1.1404 \times 10^4$ | $2.3267 \times 10^3$ | $7.3302 \times 10^4$ |
| Fast | $0.9753 \times 10^3$ | $2.7647 \times 10^3$ | $1.0241 \times 10^3$ | $0.1431 \times 10^4$ | $0.6540 \times 10^3$ | $0.6527 \times 10^4$ |
| Original/Fast | 3.6503 | 3.0870 | 3.7976 | 7.9687 | 3.5579 | 11.2307 |
| $\|V\|$ | 37 | 48 | 36 | 56 | 35 | 76 |

Table 4: Running times at $t = 100$

| Operator | CHECK-NODES |
|---|---|
| InsertU | $X_N$ |
| DeleteU | $N_x \cup \{x\}$ |
| InsertD | $X_{N\Xi}$ |
| DeleteD | $N_x \cup \{x\}$ |
| ReverseD | $X_{N\Pi}$ |
| MakeV | $X_N$ |

Table 5: The nodes that must be checked for a change at node $x$

The algorithm receives as input, the PDAG that has been modified by the last move in the search space $\mathcal{P}^{new}$, the PDAG before this modification took place $\mathcal{P}^{old}$, the set of operators being used $O$ and a set of cached validity tests $Cache$. The algorithm returns the modified cache $Cache$ at the end of the procedure. Firstly, UPDATE-CACHE calculates the nodes that have changed from PDAG $\mathcal{P}^{old}$ to $\mathcal{P}^{new}$. By this is meant those nodes where there has been a change in the edges connected to them. Next, for each operator $o$ and each changed node $c$, the other nodes that might have been affected by this change are identified by the CHECK-NODES procedure. The value for this is calculated differently for each operator — Table 5 gives values for each of Chickering's six operators. Then, for each operator and each node $x$ that must be checked, the cache entries with $x$ as the first argument are deleted. Next, the new set of nodes that are valid given the operator $o$ and the node $x$ are calculated as in Table 3. Finally the new set of valid nodes are entered into the cache.

It is hard to quantify the effect of the caching operations on the complexity of operator validity testing. In any event, it is likely to lower the amount of nodes checked from $n$.

## 5 Experimental results

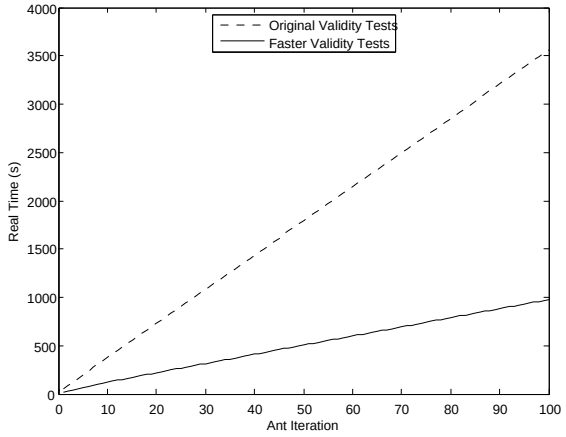
In order to test the applicability of the above-mentioned techniques, a series of experiments were run using an algorithm that would benefit from faster move validity checking — i.e. one with multiple restarts. This algorithm was the ACO-E algorithm by Daly & Shen (2007). Six example Bayesian networks were used in the overall experiment — Alarm, Barley, Diabetes, HailFinder, Mildew and Win95pts.[1] For each network, 100 experiments were run. Each individual run sampled 5000 data from the network and ran the ACO-E algorithm with Chickering's operators (Table 2), a BDeu scoring function and the parameters $t_{max} = 100$, $t_{step} = 6$, $m = 5$, $\rho = 0.1$, $q_0 = 0.9$ and $\beta = 3$. The results of the experiments can be seen in Figure 3. Here, the run time is shown for an experiment at each iteration of the ACO-E algorithm, averaged across the 100 experiment runs. Note that the graphs are designed to show the relative difference between the two experimental conditions. As such the absolute scale on each graph varies according to the example network. As can be seen from the diagrams, the validity checking techniques explored in this paper have resulted in a speed up over the original running times. To quantify these results further, the original running times and improved running times for each of the test networks at iteration $t = 100$ are shown in Table 4. Also given are the ratio of original to improved running times and the number of nodes $|V|$ in each network. The Pearson correlation coefficient was calculated over the ratio and number of nodes and came to the figure $r = 0.9287$. The critical value of $r$ for $\nu = 4$ degrees of freedom and $p = 0.01$ was found to be $r = 0.917$. Therefore there is a 99% probability that a positive correlation exists between the number of variables in a model and the factor of speed up given by the methods introduced in this paper. This would imply that the methods given are effective for wide data sets.

## 6 Conclusions and future directions

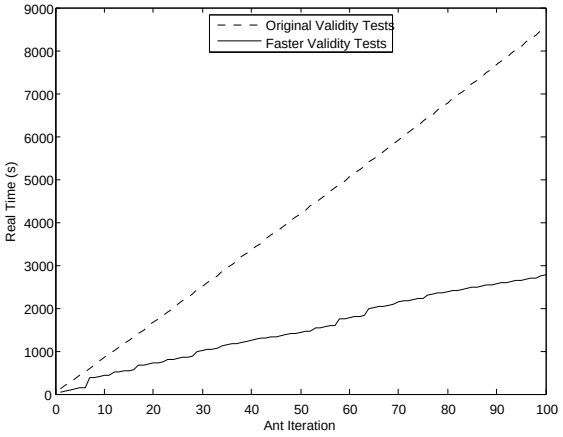
This paper has introduced two methods for speeding up the learning of equivalence classes of Bayesian network structures. The first decreases the amount of validity testing that needs to be done by constraining the nodes that are tested. The second uses a cache to store the result of validity tests and to reuse them where possible. The methods were shown to be effective in algorithms with multiple restarts and in wide data sets.

In the future, more tests need to be run on wider data sets, e.g. those with hundreds of variables instead of tens of variables. This would confirm the applicability of the methods for domains with a very large number of attributes. Also, tests need to be run across a range of different algorithms, such as simulated annealing, random walking or indeed any stochastic local search that might benefit from restarting. These would show the generalisation capability of the techniques given in this paper.

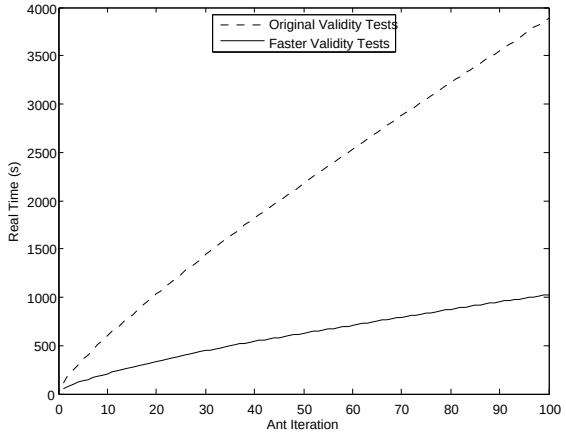
1. http://www.norsys.com/netlibrary/index.htm



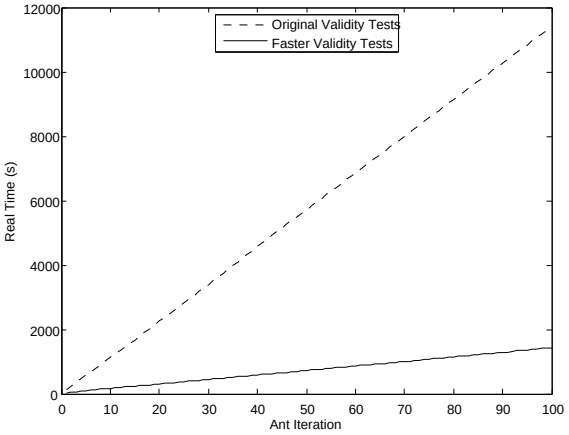

(a) Alarm



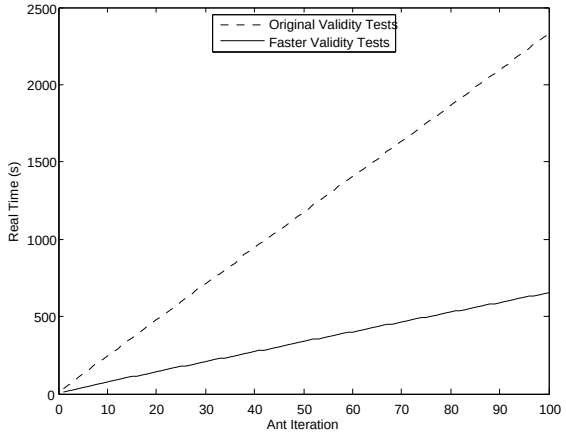

(b) Barley



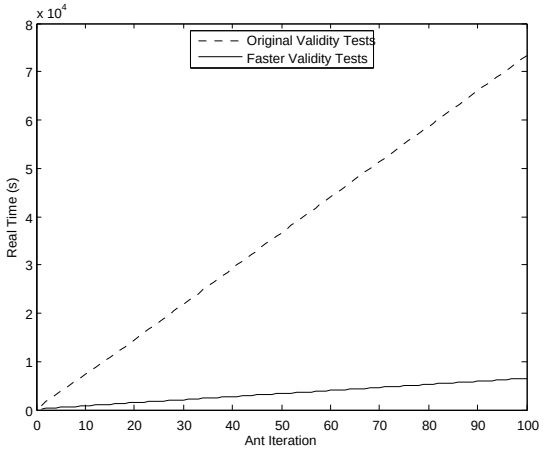

(c) Diabetes




(d) HailFinder




(e) Mildew




(f) Win95pts

Figure 3: Comparison of original and fast validity checking

## References


Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control* **19**(6), 716–723.

Buntine, W. (1991). Theory Refinement on Bayesian Networks. In B. D'Ambrosio, P. Smets & P. Bonissone (Eds.), *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, 52–60. San Mateo, CA: Morgan Kaufmann.

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.

Cooper, G.F. & Herskovits, E. (1992). A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning* **9**(4), 309–347.

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. (2007). Learning Bayesian Network Equivalence Classes with Ant Colony Optimization. *Under review for journal publication*.

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.

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

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

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

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.

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics* **6**(2), 461–464.

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.

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.