Random Generation of Bayesian Networks

Jaime S. Ide and Fabio G. Cozman

Escola Politécnica,
University of São Paulo
Av. Prof. Mello Moraes, 2231 - São Paulo, SP - Brazil
jaime.ide@poli.usp.br, fgcozman@usp.br

Abstract. This paper presents new methods for generation of random Bayesian networks. Such methods can be used to test inference and learning algorithms for Bayesian networks, and to obtain insights on average properties of such networks. Any method that generates Bayesian networks must first generate directed acyclic graphs (the "structure" of the network) and then, for the generated graph, conditional probability distributions. No algorithm in the literature currently offers guarantees concerning the distribution of generated Bayesian networks. Using tools from the theory of Markov chains, we propose algorithms that can generate uniformly distributed samples of directed acyclic graphs. We introduce methods for the uniform generation of multi-connected and singly-connected networks for a given number of nodes; constraints on node degree and number of arcs can be easily imposed. After a directed acyclic graph is uniformly generated, the conditional distributions are produced by sampling Dirichlet distributions.

1 Introduction

In this paper we describe a solution to a problem that is very simple to state, but very hard to solve. Our problem is to randomly generate Bayesian networks *with an uniform distribution*. Why is this useful? Two points should suffice to indicate the need for randomly generated networks with a uniform distribution:

1. Many algorithms for inference and learning using Bayesian networks must be tested, and uniformly generated Bayesian networks offer a natural way to produce "unbiased" experiments.
2. Properties of Bayesian networks (such as the average number of connected components, average number of independent variables) are usually very hard to derive analytically, and uniformly generated Bayesian networks can be used for exploring such questions empirically.

Because Bayesian networks occupy a prominent position as a model for uncertainy in artificial intelligence [3], it would seem that algorithms for uniform generation of Bayesian networks would be easily available. Alas, this is not the case. One reason for this is that Bayesian networks are composed of directed acyclic graphs, and it is very hard to represent the space of such graphs. Consequently, it is not easy to guarantee that a given method actually produces a uniform distribution in that space. Another reason is that usually Bayesian networks are sparsely connected; to be able to investigate properties that are relevant to practical problems, we must generate directed acyclic graphs subject to constraints on number of arcs, degree of nodes, number of parents for nodes — generating such graphs while guaranteeing uniform distributions is quite a challenge. In this paper we present algorithms for uniformly generating random directed acyclic graphs through Markov chains.

In Section 2 we review the theory of Bayesian networks and the basic concepts used in this paper. We review the problem of Bayesian network generation and existing approaches in Section 3. In Section 4 we develop algorithms for generation of random directed acyclic graphs. We demonstrate that our methods can uniformly generate multi-connected and singly-connected Bayesian networks for a given number of nodes and limits on node degree and number of arcs (other constraints can be imposed by modifying the basic algorithms). In Section 5 we describe an implementation and tests with our methods.

2 Bayesian networks and graphs

This section summarizes the theory of Bayesian networks and introduces terminology used throughout the paper. All random variables are assumed to have a finite number of possible values. Denote by $p(X)$ the probability density of X, and by $p(X|Y)$ the probability density of X conditional on values of Y.

A Bayesian network represents a joint probability density over a set of variables $\mathbf{X}$ [3]. The joint density is specified through a directed acyclic graph.

A directed graph is composed of a set of nodes and a set of arcs. An arc (u, v) goes from a node u (the *parent*) to a node v (the *child*). A *path* is a sequence of nodes such that each pair of consecutive nodes is adjacent. A path is a *cycle* if it contains more than two nodes and the first and last nodes are the same. A cycle is *directed* if we can reach the same nodes while following arcs that are in the same direction. A directed graph is *acyclic* (it is a *DAG*) if it contains no directed cycles. A graph is *connected* if there exists a path between every pair of nodes. A graph is *singly-connected* if there exists exactly one path between every pair of nodes; otherwise, the graph is *multiply-connected* (or multi-connected for short). A singly-connected graph is also called a *polytree*. An *extreme sub-graph* of a polytree is a sub-graph that is connected to the remainder of the polytree by a single path.

In a Bayesian network, each node of its graph represents a random variable X_i in $\mathbf{X}$. The *parents* of X_i are denoted by $\mathrm{pa}(X_i)$. The semantics of the Bayesian network model is determined by the *Markov condition*: Every variable is independent of its nondescendants nonparents given its parents. This condition leads to a unique joint probability density [6]:

$$p(\mathbf{X}) = \prod_i p(X_i|\mathrm{pa}(X_i)) \,. \tag{1}$$

Every random variable X_i is associated with a conditional probability density $p(X_i|\mathrm{pa}(X_i))$. Figure 1 depicts examples of DAGs as Bayesian networks.

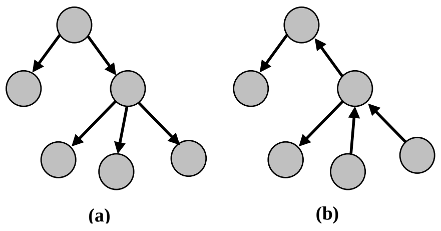


Fig. 1. Bayesian networks:(a) Tree, (b)Polytree, (c) Multi-connected Bayes net.

3 Generating Bayesian networks

To generate random Bayesian networks, the obvious method is to generate a random DAG, and then to generate the conditional probability distributions for that graph.

Given a DAG, it is relatively easy to generate uniformly distributed random conditional distributions. Suppose then that we are generating the distribution $p(X|\mathrm{pa}(X))$ for a fixed value of $\mathrm{pa}(X)$, where X has k values. A general method is to define a Dirichlet distribution over the k values of X with priors $(\alpha_1, \alpha_2, \ldots, \alpha_k)$; we then have to sample from k Gamma distributions and normalize these k samples [7].[1] If we want to generate a uniform distribution, we simply set all α's to 1. (It should be noted that, for the specific problem of uniformly generating distributions, Caprile has proposed a more efficient method than the one based on Gamma distributions [1].)

The real difficulty is to generate random DAGs that are uniformly distributed. Many authors have used random graphs to test Bayesian network algorithms, generating these graphs in some ad hoc manner. A typical example of such methods is given by the work of Xiang and Miller [9]. By creating some heuristic graph generator, it is usually impossible to guarantee any distribution on the generated neworks; consequently, any conclusion reached by using the generated graphs may be biased in some unknown direction. On the other hand, it can be argued that any generator that produces a uniform distribution on the space of *all* DAGs is not very useful. The problem is that practical Bayesian networks usually have a reasonably small degree; if a generator produces graphs that are too dense, these graphs are not representative examples of Bayesian networks. So, we must generate graphs uniformly over the space of graphs that are *connected*, *acyclic*, and *not very dense*. We assume that the number of arcs in a graph is a good indicator of how dense the graph is, so we assume that our problem is to uniformly generate connected DAGs with restrictions either on node degrees or on number of arcs. Other constraints can be imposed using straightforward modifications of our algorithms.

A type of Bayesian network that is of great practical interest is represented by polytree structures [6]. Polytrees seem to be sufficiently general to represent

[1] Thanks to Nir Friedman for pointing this method to us.

many real-world problems while being amenable to polynomial algorithms for computation of probabilities. So, we can establish another problem: to uniformly generate polytrees with n nodes. To the best of our knowledge, there exists no algorithm for random generation of polytrees so far.

4 Markov chains for generating connected DAGs

Our approach to generate random graphs is to use Markov chains. We are directly inspired by the work of Melançon et al on random graph generation [4]. The main difference between Melançon et al's work and ours is that they let their graphs be disconnected, a detail that makes considerable difference in the correctness proofs.

A few necessary concepts are briefly reviewed here. Consider a Markov chain over finite domains [2], and $P = (p_{ij})_{ij=1}^{N}$ to be a N x N matrix representing transition probabilities, where, $p_{ij} = Pr(X_{t+1} = j | X_t = i)$, for all t. The s-step transition probabilities is given by $P^s = p_{ij}^{(s)} = Pr(X_{t+s} = j | X_t = i)$, independent of t. We denote the initial distribution of the Markov chain by the vector $\pi^{(0)}$. A Markov chain is *irreducible* if for all i,j, there is s that satisfies $p_{ij}^{(s)} > 0$. A Markov chain is irreducible if and only if all pair of states intercommunicate. A Markov chain is *aperiodic* if the greatest common denominator of all s such that $p_{ii}^{(s)} > 0$ is $d = 1$. Aperiodicity is ensured when $p_{ii} > 0$. A Markov chain is *ergodic* if there exists a vector π (the stationary distribution) satisfying $\lim_{s \longrightarrow \infty} p_{ij}^{(s)} = \pi_j$, for all i and j. Any finite chain that is aperiodic and irreducible is ergodic. A non-negative transition matrix is called *doubly stochastic* if the rows and columns sum one (thay is, if $\sum_{j=1}^{N} P_{ij} = 1$ and $\sum_{i=1}^{N} P_{ij} = 1$). A Markov chain with a doubly stochastic transition matrix has a stationary distribution that is uniform.

We can generate random graphs by simulating Markov chains. To have a Markov chain, it is enough that we can "move" from a graph to another graph in some probabilistic way that depends only on the current graph. Such a Markov chain will be irreducible if it can reach any graph from any graph. Also, the chain will be aperiodic if there exists a self-loop probability, i.e. there is a chance that the next generated graph is the same as the current one. If the moves are governed by a doubly stochastic transition matrix, the unique stationary distribution for the process is uniform over the space of possible moves.

4.1 Generating multi-connected DAGs

Consider a set of n nodes (from 0 to $n-1$) and the Markov chain described by Algorithm 1. We start with a connected graph. The loop between lines 3 and 7 construct the next state from the current state (this procedure defines a transition matrix). Our transitions are limited to 2 operations: adding and removing arcs provided the graph is still acyclic and connected. If we did not need to keep the graph connected, the following theorems would be immediate as pointed

Algorithm 1: Generating Multi-connected DAG's

```
Input: number of nodes ($n$), number of iterations ($N$).
Output: Return a connected DAG with $n$ nodes.
01. Inicialize a simple ordered tree with $n$ nodes, where all nodes have just one parent,
except the first one that does not have any parent;
02. Repeat the next loop $N$ times:
03.         Generate uniformly a pair of distinct nodes $i$ and $j$;
04.         If the arc $(i, j)$ exists in the actual graph, delete the arc, provided that the
underlying graph remains connected;
05.         else
06.         Add the arc, provided that the underlying graph remains acyclic;
07.         Otherwise keep the same state;
08. Return the current graph after $N$ iterations.
```

Fig. 2. Algorithm for Generating multi-connected DAGs.

out by Melançon et al. We have decided to present detailed proofs, resorting to constructive arguments where possible; the proofs can work as guiding tools if the reader wishes to modify the constraints imposed on graphs (for example, to limit the number of parents of a node).

Theorem 1 *The transition matrix defined by the Algorithm 1 is doubly stochastic.*

Proof. Note that we have constructed our chain to have a symmetric transition matrix; paths between two states have the same probability in both directions. There is a self-loop probability (line 7) that one minus the probability of other moves. Therefore, rows and columns of the transition matrix add to one. QED

Theorem 2 *The Markov chain generated by algorithms 1 is irreducible.*

Proof. A Markov chain is irreducible if any two states of this chain intercommunicate, that is, there is a probability from any state reach another state. Suppose that we have a multi-connected DAG with n nodes; if we prove that from this graph we can reach a simple ordered tree (Figure 3), the opposite transformation is also true, because of the symmetry of our transition matrix — and therefore we could reach any state from any other. We start by finding a loop cutset and removing enough arcs to obtain a polytree from the multi-connected DAG [6]. For each pair of extreme sub-graphs of the polytree, we have three possible cases described at Figure 4. In all three cases, we can add an arc between the last node of an extreme sub-graph and the first node, and remove the arc as depicted in the figure. Doing this we get a unique extreme sub-graph. If we have more than 2 extreme sub-graphs connected to a node, we repeat this process by pairs; we can do this recursively until get a simple polytree.

Now that we have a simple polytree, we want to get a simple tree, i.e. all arcs directed in one direction. Starting at the right extreme sub-graph of this simple

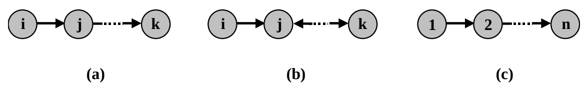


Fig. 3. (a) Simple tree, (b) Simple polytree, (c) Simple ordered tree.

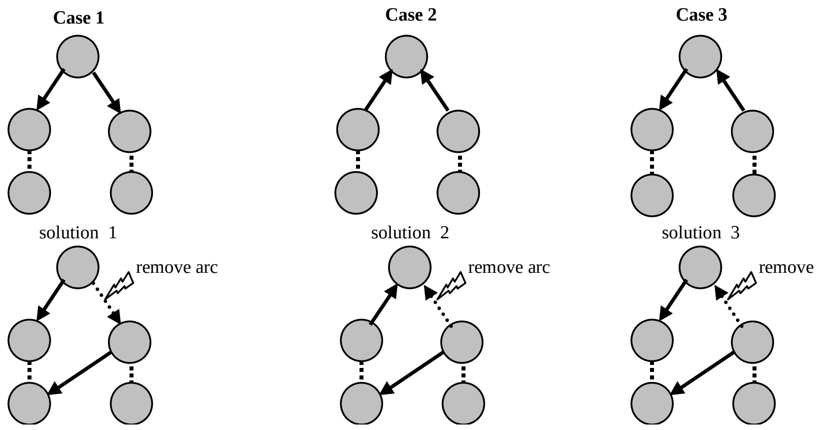


Fig. 4. Three possible cases for transforming a polytree into a simple polytree.

polytree, we have to invert all arcs that are directed to the left. We run over all arcs, starting at the right side; if an arc is directed to the right, it does not need to be inverted; otherwise, we have 2 cases (Figure 5). Suppose that we have three nodes i, j, k. Add an arc between nodes i and k with appropriate direction, invert (remove and add) the arc (j, i) and at the end remove arc between i and k. Repeat this process until all arcs are processed. Notice that in the last arc we only have one possibility. At the end we get a simple tree.

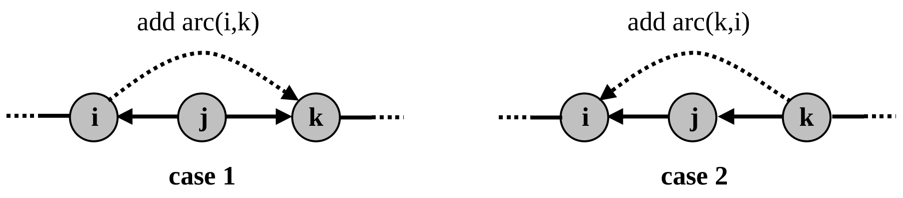


Fig. 5. Two possible cases for transforming a simple polytree into a simple tree (arcs are inverted to the right.

The last step is to get to a simple ordered tree from the simple tree. The idea of the ordering process is illustrated in Figure 6. Start at node 0 and go on until the last node $(n-1)$. Suppose that j is the processed node; add a possible arc $(p, 0)$ and remove the arc (i, j) (step 2); then add an arc (i, k) and remove arc (j, k) (step 3); the last step is to add an arc $(j, j + 1)$ directed to the next in order and to remove arc $(p, 0)$. So, from any multi-connected DAG, it is possible to reach a simple ordered tree. The opposite proof is analogous. Consequently, we have that from any multi-connected DAG is possible to reach any other, i.e. this Markov chain is irreducible. QED

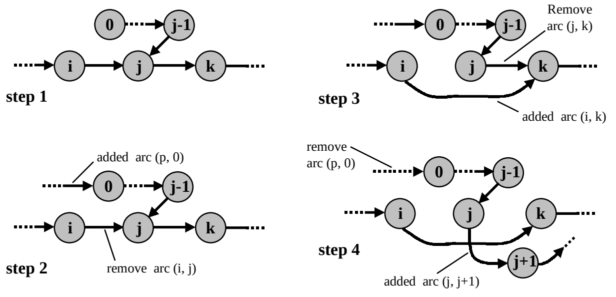


Fig. 6. Basic moves to obtain a simple ordered tree.

Theorem 3 *The Markov chain generated by Algorithm 1 is aperiodic.*

Proof. In any state there is an arc that will make the graph cyclic, so it is always possible to stay at the same state (there is a self-loop probability greater than zero). QED

Theorem 4 *The Markov chain generated by Algorithm is ergodic and its unique stationary converges to a uniform distribution.*

Proof. Follows from the previous theorems. QED

It is important to note that additional requirements, such as limitations on the number of arcs or on maximum degree, can be easily added to line 6. The transition matrix probabilities will change, but the proofs all carry through without problems.

4.2 Generating polytrees

The process of generating polytrees is similar to Algorithm 1; we focus on the differences between algorithms and do not give a detailed description of properties and proofs. Consider again n nodes, and the transition matrix defined by Algorithm 2.

In line 1 we start with a simple ordered tree (this is a valid polytree). The loop between line 3 to 7 is the construction of the transition matrix. Line 4

```
Algorithm 2: Generating Polytree
Input: number of nodes ($n$), number of iterations ($N$).
Output: Return a polytree with $n$ nodes.
01. Inicialize a simple ordered tree with $n$ nodes as in Algorithm 1.
02. Repeat the next loop $N$ times:
03.         Generate uniformly a pair of distinct nodes $i$ and $j$;
04.         If the arc $(i, j)$ exists in the actual graph, keep the same state;
05.         else
06.         Invert the arc with probability $1/2$ to $(j, i)$, and then
07.         Find the predecessor node $k$ in the path between $i$ and $j$, remove the arc
    between $k$ and $j$, and add an arc $(i, j)$ or arc $(j, i)$ depending on the result of line 06.
08. Return the current graph after $N$ iterations.
```

Fig. 7. Algorithm for generating polytrees.

ensures a self-loop probability greater than zero, to produce an aperiodic Markov chain. Line 6 is important to obtain symmetry for the transition matrix. Figure 8 illustrates a transition process between two neighbor states. Suppose that at state A we obtain the arc (i, j) with probability $p = 1/(n(n-1))$. As described in line 7, through a "remove and add" operation we get to a state B with probability $p_{AB} = 1/(2n(n-1))$. Note that the opposite transition state B to state A has the same probability $p_{BA} = p_{AB} = 1/(2n(n-1))$. This is possible because of the 50 percent probability factor (line 6). Therefore, Algorithm 2 produces a doubly-stochastic matrix just as Algorithm 1.

In Algorithm 1, "add" and "remove" operations are distinct, while at Algorithm 2, these operations are combined (line 7), because we cannot remove arcs from a polytree and keep it connected, and we cannot add arcs to a polytree and keep it as a polytree. The proof for irreducibility for Algorithm 1 follows the proof of Theorem 2. The operation in line 7 is simply a composition of operations, and it is easy to see that any polytree intercommunicates with any other. In addition, the "invert" operation makes it easy to invert arcs direction. We

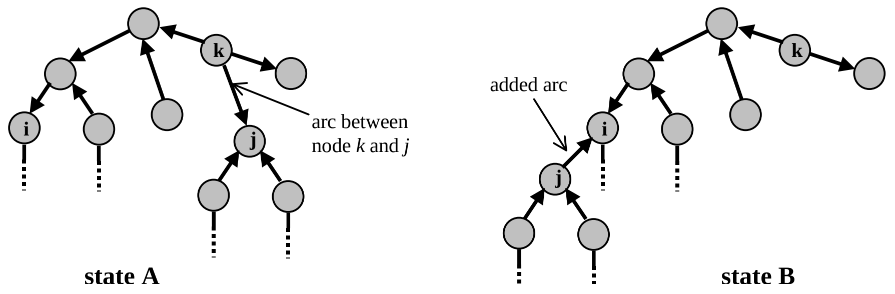
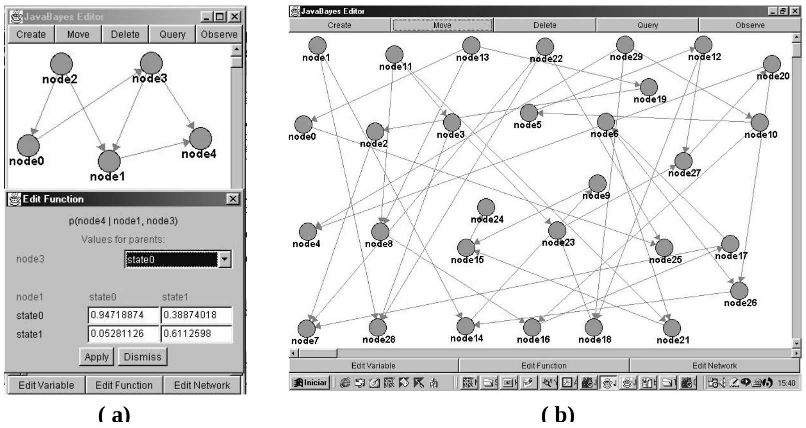


Fig. 8. Example of transition: the polytree is cut in two parts; a new polytree is constructed merging these parts randomly, through a single "add and remove" operation.



p(node4 | node1, node3)

Values for parents:

node3 state0

node1	state0	state1
state0	0.94718874	0.38874018
state1	0.05281126	0.6112598

Fig. 9. Random Bayesian networks (a) with 5 nodes, showing random distribution; (b) with 20 nodes. Networks viewed in the JavaBayes system.

therefore have an aperiodic irredicible Markov chain whose state space contains all possible polytrees with n nodes, and that converges to a uniform stationary distribution.

5 Experimental results

The algorithms for generating Bayes net structures and probability functions have been implemented in Java. The resulting program is called BNGenerator and is freely available under the GNU license. The program saves generated networks in the XML format read by the freely distributed JavaBayes system. In figure 9, we have the graphical representation of networks generated with different parameters. In figure 10, we have a simple histogram of samples generated with 4 nodes, illustrating that the networks have a uniform distribution.

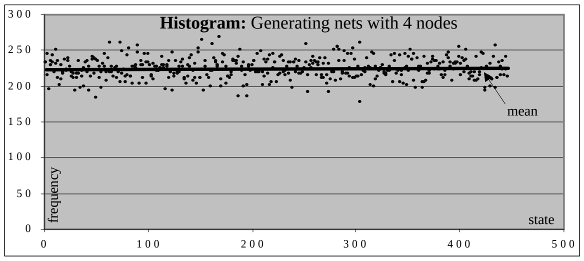


Fig. 10. Histogram of 100.000 generated nets with 4 nodes.

6 Conclusion

We can summarize this paper as follows: we have introduced algorithms for generation of uniformly distributed random Bayesian networks, both as multi-connected networks and polytrees. Our algorithms are flexible enough to allow specification of maximum numbers of arcs and maximum degrees, and to incorporate any of the usual characteristics of Bayesian networks. We suggest that the methods presented here provide the best available scheme at the moment for producing valid tests and experiments with Bayesian networks. A disadvantage of our methods, compared to existing ad hoc schemes, is that many networks have to be generated before a sample can be taken (that is, it is necessary to wait for the Markov chains to converge, so the value of N in the algorithms must be high). In our implementation we have observed that the algorithms are fast, so we can easily wait for thousands of iterations before obtaining a sample.

Acknowledgements

We thank Nir Friedman for suggesting the Dirichlet distribution method, and Robert Castelo for pointing us to Melançon et al's work. We thank Guy Melançon for confirming that the idea of Algorithm 1 was sound and for making his DagAlea software available. We also thank Jaap Suermondt and Alessandra Potrich for providing important ideas, and Y. Xiang, P. Smets, D. Dash, M. Horsh, E. Santos, and B. D'Ambrosio for suggesting valuable procedures.

References

1. Caprile, B.: Uniformly Generating Distribution Functions for Discrete Random Variables (2000).
2. Gamerman, D.: Markov Chain Monte Carlo. Stochastic simulation for Bayesian inference. Texts in Statistical Science Series. Chapman and Hall, London (1997)
3. Jensen, F.V.: An Introduction to Bayesian Networks. Springer-Verlag, NewYork (1996)
4. Melançon, G., Bousque-Melou, M.: Random Generation of Dags for Graph Drawing. Dutch Research Center for Mathematical and Computer Science (CWI). Technical Report INS-R0005 February (2000)
5. Chartrand, G., Oellermann, O.R.: Applied and Algorithmic Graph Theory. International Series in Pure and Applied Mathematics. McGraw-Hill, New York (1993).
6. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kauffman (1988)
7. Ripley, B.D.: Stochastic Simulation. Wiley Series in Probability and mathematical Statistics. John Wiley and Sons, Inc., New York (1987).
8. Sinclair, A.: Algoritms for Random Generation and Counting: A Markov Chain Approach. Progress in Theoretical Computer Science. Birkhaüser, Boston (1993).
9. Xiang, Y., Miller, T.: A Well-Behaved Algorithm for Simulating Dependence Structures of bayesian Networks. International Journal of Applied Mathematics, Vol. 1, No. 8 (1999), 923–932