IDIAP RESEARCH REPORT

IDIAP

Martigny - Valais - Suisse

An Introduction to Bayesian Network Theory and Usage

Todd A. Stephenson [a]

IDIAP–RR 00-03

February 2000

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 − 27 − 721 77 11
fax +41 − 27 − 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

[a] Dalle Molle Institute for Perceptual Artificial Intelligence

IDIAP Research Report 00-03

An Introduction to Bayesian Network Theory and Usage

Todd A. Stephenson

February 2000

Abstract. I present an introduction to some of the concepts within Bayesian networks to help a beginner become familiar with this field's theory. Bayesian networks are a combination of two different mathematical areas: graph theory and probability theory. So, I first give the basic definition of Bayesian networks. This is followed by an elaboration of the underlying graph theory that involves the arrangements of nodes and edges in a graph. Since Bayesian networks encode one's beliefs for a system of variables, I then proceed to discuss, in general, how to update these beliefs when one or more of the variables' values are no longer unknown (i.e., you have observed their values). Learning algorithms involve a combination of learning the probability distributions along with learning the network topology. I then conclude Part I by showing how Bayesian networks can be used in various domains, such as in the time-series problem of automatic speech recognition. In Part II I then give in more detail some of the algorithms needed for working with Bayesian networks.

Acknowledgements: This work is supported by the Swiss National Science Foundation under grant # 21-53960.98. I would also like to thank Andrew Morris and Samy Bengio for their comments on this paper.

Part I
Theory

1 Graphical Models

1.1 Graphical Models in General

A graphical model (Whittaker, 1990) is a tool that is used to visually illustrate and work with conditional independencies among variables in a given problem. Two variables that are conditionally independent have no direct impact on each other's value. For example, A is conditionally independent of C given B if $P(A|B,C) = P(A|B)$ (Cowell *et al.*, 1999, Section 5.1). Furthermore, the graphical model will show any intermediary variables that separate two conditionally independent variables. It is through these intermediary variables that two conditionally independent variables affect each other.

A graph is composed of a set of nodes (which in graphical models represent variables) and a set of edges. Each edge connects two nodes, and an edge can have an optional direction assigned to it. For directed edges, the edge is said to be from parent X_p to child X_c. For any given edge between variables X_1 and X_2, if there is a causal relationship between the variables, the edge will be directional, leading from the cause variable to the effect variable; if there is just a correlation between the two variables, the edge will be undirected (Cowell *et al.*, 1999, Section 3.1.1). So, say that you have two conditionally independent variables A and C. Now these two variables both are directly related to another variable, B. So you draw an edge between the nodes of the variables that are directly related, that is, between A and B and between B and C. Furthermore, say that the relations between A and B and between B and C each work equally in two directions; so, make both edges undirected. Figure 1 illustrates how both of A and C are dependent upon the variable B. However, there is no edge between A and C. So, A and C are conditionally independent, given variable B. Note that this is not the same as saying A and C are totally independent; it merely means that variable B encodes any information from A that impacts C, and vice-versa.

For each variable there is a probability distribution function (it can be continuous or discrete; though this paper deals only with discrete functions) whose definition depends on the edges leading into the variable. For example, the probability distribution for variable A depends only on the value of variable B while the probability distribution for the variable B in Figure 1 depends both on the value of variable A and on the value of variable C.

So, a graphical model is defined as follows. It consists of variables (nodes) $K = \{1, 2, \ldots, k\}$ with a set E of dependencies (edges) between the variables and a set P of probability distribution functions for each variable.

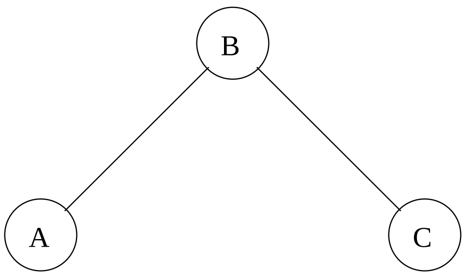

Figure 1: *A graphical model illustrating conditional independence (Cowell* et al.*, 1999, Definition 5.1). Variables A and C are conditionally independent, given the variable B. (Probabilities are omitted).*

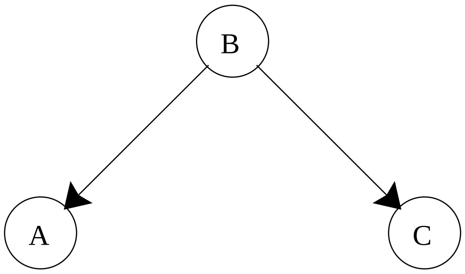

Figure 2: *A Bayesian network. (Probabilities are omitted).*

1.2 Bayesian Networks

Bayesian networks are a specific type of graphical model[1]. A Bayesian network is the specific type of graphical model which is a *directed acyclic graph* (DAG, see page 5) (Neapolitan, 1989). That is, all of the edges in the graph are *directed* (i.e., they point in a particular direction) and there are *no cycles* (i.e., there is no way to start from any node and travel along a set of directed edges in the correct direction and arrive back at the starting node).

Figure 2 illustrates a Bayesian network. Its set of edges is $E = \{(B, A), (B, C)\}$. This constitutes a DAG because (1) there are no undirected edges (i.e., there are no edges going in both directions between any vertices) and (2) there are no cycles (i.e., once you leave any given vertex and are following the direction of the edges, there is no way to cycle back to the original vertex). Furthermore, since A and C are conditionally independent of each other, you can say the following: $P(A|B, C) = P(A|B)$, which means that, for the factorization represented by this Bayesian network, the probability of A is conditioned only on B and that the value of C is irrelevant for this local probability. Likewise, you can say that $P(C|A, B) = P(C|B)$. The edges in the Bayesian network encode a particular factorization of the joint distribution. In this example, the joint distribution of all the variables, as factorized by this Bayesian network, is

$$P(A, B, C) = P(A|B) \cdot P(B) \cdot P(C|B) \tag{1}$$

In general, given nodes $\mathbf{X} = X_1, \ldots, X_n$, the joint probability function for any Bayesian network is

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|parents(X_i)) \tag{2}$$

That is, the joint probability of all of the variables is the product of the probabilities of each variable given its parents' values. This shows the meaning of the directed edges: they indicate which other variables a given variable is conditioned upon.

The edges in Bayesian networks are sometimes looked at as causal connections where each parent node causes an effect on its children. While this notion of causality is useful in constructing Bayesian networks, note that the joint probability represented by one set of edges can equally be represented by another set. For example, using Bayes' rule the joint probability from (1) becomes:

$$P(A|B) \cdot P(B) \cdot P(C|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \cdot P(B) \cdot P(C|B) = P(A) \cdot P(B|A) \cdot P(C|B) \tag{3}$$

This is represented graphically by Figure 3.

2 Graph Theory Terminology

Having defined Bayesian networks, I will now provide some of the theory that is needed for the later discussions of Bayesian networks. Although Bayesian networks combine probability theory and graph

[1] Other types of graphical models, some of which are similar to or synonyms for Bayesian networks, are (Bayesian) belief networks, causal networks, (probabilistic) independence networks, probabilistic networks, Markov fields.

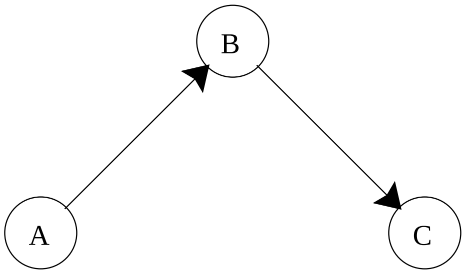

Figure 3: *A Bayesian network that represents the same joint probability as the one in Figure 2. The edge (B, A) has been reversed to become (A, B). (Probabilities are omitted).*

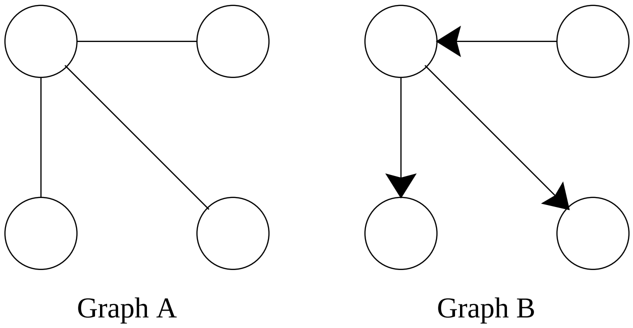

Figure 4: Directionality. *While graphs A and B have connections between the same nodes, they are not the same graph. Graph A has undirected edges while graph B has directed edges which point in certain directions.*

theory, I will be providing some of the basics of only graph theory here. For more information, please see Neapolitan (1989, chap. 3 & 7), which provides a good background into the graph theory from a Bayesian networks' perspective.

Brualdi (1992, Section 11.1) defines a graph as (1) a set of **nodes** (also called vertices) and (2) a set of **edges** (also called arcs), each edge being a pair of nodes. This is also called an undirected graph. If the two vertices within each edge (X_i, X_j) are ordered, then the edges have a direction assigned to them (that is, they lead specifically from node X_i to node X_j and not in the other direction); this is called a digraph (or, a **directed** graph). Brualdi (1992) also presents multigraphs and general digraphs, which allow multiple connections between any two nodes in graphs and digraphs, respectively. The following properties of a graph deal with what arrangement of edges appear among the nodes in a graph or digraph.

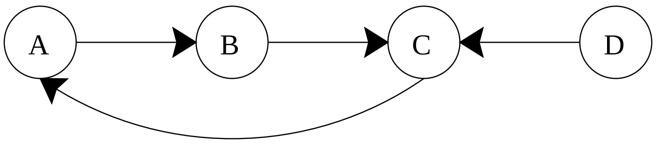

Figure 5: Chains and paths. *The vertex sequence $A - B - C - D$ is a chain of length 3 (but it is not a path). The vertex sequence $A - B - C - A - B$ is a path of length 4. The vertex sequence $A - B - C$ is a simple path of length 2. The vertex sequence $A - B - C - A - B - C - A$ is a cycle of length 6. The vertex sequence $A - B - C - A$ is a simple cycle of length 3.*

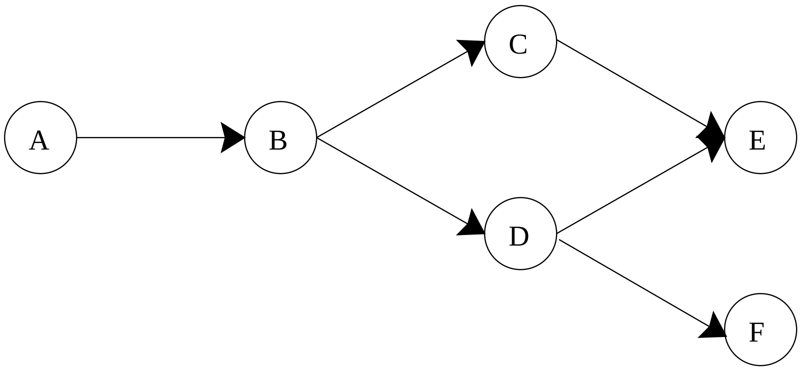

Figure 6: DAGs, parents/children, ancestors/descendants, family. *The above graph is a DAG. A is the parent of B while B is the child of A; B is the parent of C and D while C and D are the children of B; and so on. A is the ancestor of B, C, D, E, and F while B, C, D, E, and F are the descendants of A; and so on. A possible ancestral ordering for this graph is* A, B, C, D, E, F*; another acceptable ancestral ordering is* A, B, D, F, C, E*. Some sample families from this graph are* A, B *as the family of B;* B, C *as the family of C; and* C, D, E *as the family of E.*

A **chain** is a series of nodes where each successive node in the chain is connected to the previous node by an edge (regardless of the direction, if any, of the edge). A **path** is a chain with the further constraint for digraphs that each connecting edge in the chain has a directionality going in the same direction as the chain. A **cycle** is a path that starts and ends at the same node. A **simple path** is a path with unique nodes. A **simple cycle** is a cycle where, except for the start/end node, all nodes are unique (See Figure 5). A **directed acyclic graph**, or **DAG**, is a directed graph that has no cycles (See Figure 6).

A **parent/child** relationship in a directed graph occurs when there is an edge (X_1,X_2), from X_1 to X_2; X_1 is called the parent of X_2 and X_2 is called the child of X_1. In other words, the edge points from the parent to the child. An **ancestor/descendant** relationship, furthermore, is the extension of the parent/child relationship. For example, if X_1 is the parent of X_2 and if X_2 is the parent of X_3, then X_1 is an ancestor of X_3 and X_3 is a descendant of X_1. Like in human genealogies, this relationship extends further than just these two sets of parent/child relationships. An **ancestral ordering** is an ordering of nodes where each ancestor comes before its respective descendants. This is always and only possible in DAGs. (See Figure 6). A **family** is the set of vertices composed of X and the parents of X. For example, in Figure 6, the vertices $\{C, D, E\}$ are the family of the vertex E. Whereas the terms **parent** and **child** define the relationship between two vertices connected by a directed edge, the term **adjacent** (or **neighbor**) describes the relationship between two vertices connected by an undirected edge; the two nodes are said to be adjacent.

A **forest** is a DAG where each node has either one parent or none at all. A **tree** is a forest where only one node (called the root) has no parent; in other words, every node but the root has exactly one parent. (See Figures 7 & 8). It should be noted that while books in graphical modeling define trees as above (Cowell *et al.*, 1999; Neapolitan, 1989), others define a tree as being a connected, *undirected*, acyclic graph (Brualdi, 1992; Boffey, 1982). This brings about the term **directed tree** versus regular trees. In this paper I am only concerned with directed trees, and I will, therefore, use the directed definition from Cowell *et al.* (1999) and Neapolitan (1989).

A **moral graph** is made from a DAG. For a DAG, marry the parents of each node; this means that you add an undirected edge between each parent. After doing this, you remove the directionality from all of the original edges, resulting in a undirected graph. (See Figure 9).

For a given path (or cycle), a **chord** is an edge that does not appear in the path but which is

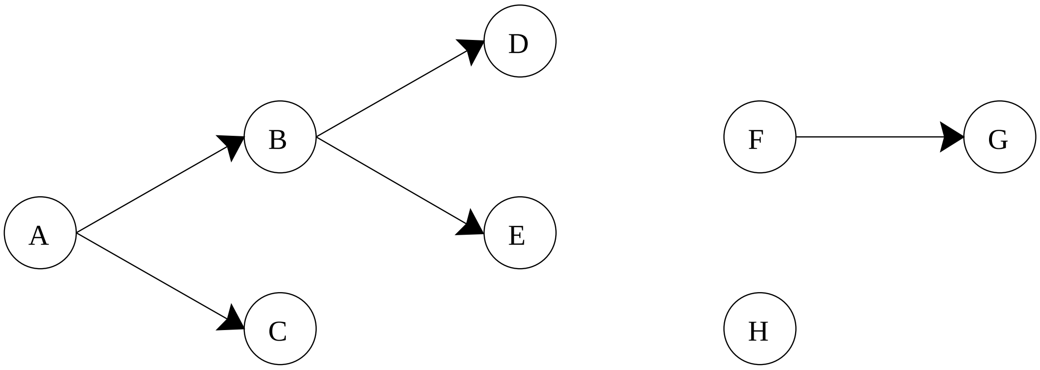

Figure 7: Forest. *The above graph, consisting of all nodes $A, B, \ldots, H$ is a forest. Every node has either one parent or none at all. B, C, D, E, G have one parent each while A, F, H have no parents. This graph is not a tree because it does not meet the requirement of only one of its nodes having no parent.*

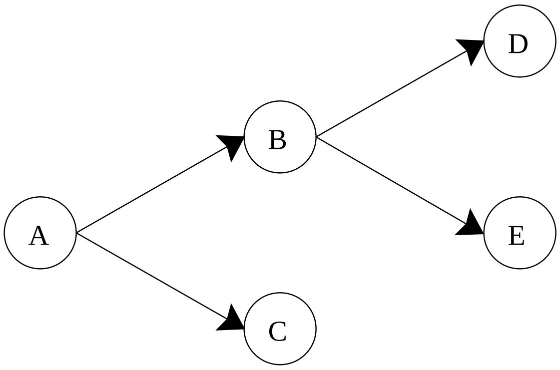

Figure 8: Trees. *The above graph is a tree. It is a DAG where every node has exactly one parent, except for the root node A which has no parent.*

between two nodes that occur in the path. The term **chordless** describes a simple path or (simple cycle) for which no chords exist. For example, consider the cycle $B - C - E - D - B$ from the graph in Figure 9. A chord for this cycle is the edge (C, D). However, the cycle $B - C - D - B$ is chordless as there are no edges between non-adjacent nodes in this cycle.

The term **triangulated** describes an *undirected* graph where any simple cycle with at least four nodes also has at least one chord. Note that it is not possible to have any chords in a cycle with three nodes. (See Figure 9).

The term **complete** describes an *undirected* graph where every node is connected to all other nodes.

A **clique** is a subset of nodes which is complete and can not be made any larger while still being complete. For example, say that the subset of nodes X_1, X_2, X_3 is complete; if they form a clique, this means that there is no other node X_i that can be added this subset with it still being complete. While Whittaker (1990, Section 3.1) specifies that a clique is maximal, this is not always strictly enforced. For example, Golumbic (1980, Section 1.1) makes a distinction between a clique and a maximal clique. (See Figure 10).

A **join tree** is a DAG that has been moralized and triangulated and then had its cliques become the nodes of a tree. See Figure 11 on page 8. It needs to have the characteristic known as the **running intersection property**, which means that any vertex (or vertex set) that is found in any two cliques C_i and C_j in the tree will also be found in all of the cliques found on the chain between C_i and C_j.

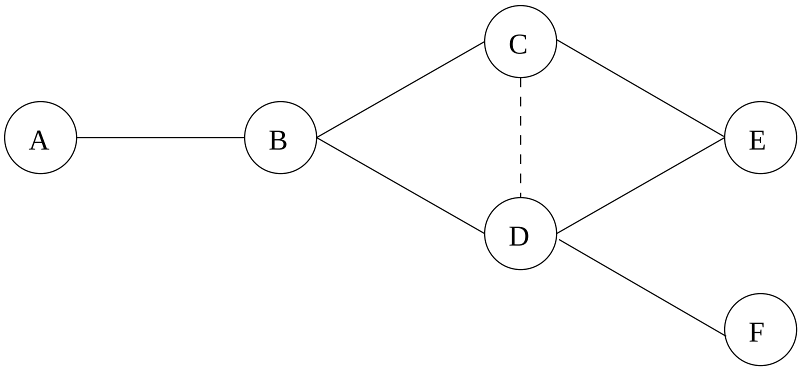

Figure 9: Moral and triangulated graphs. *The above graph is the moralized form of the graph in Figure 6 on page 5. The only node with "unwed" parents in that graph is E. In moralizing the graph, C and D, the parents of E, are connected. Then, the directions are removed from all the edges. This also happens to be a triangulated graph because its only simple cycle with a length of at least 4, $B-C-E-D-B$, does have a chord ($C-D$).*

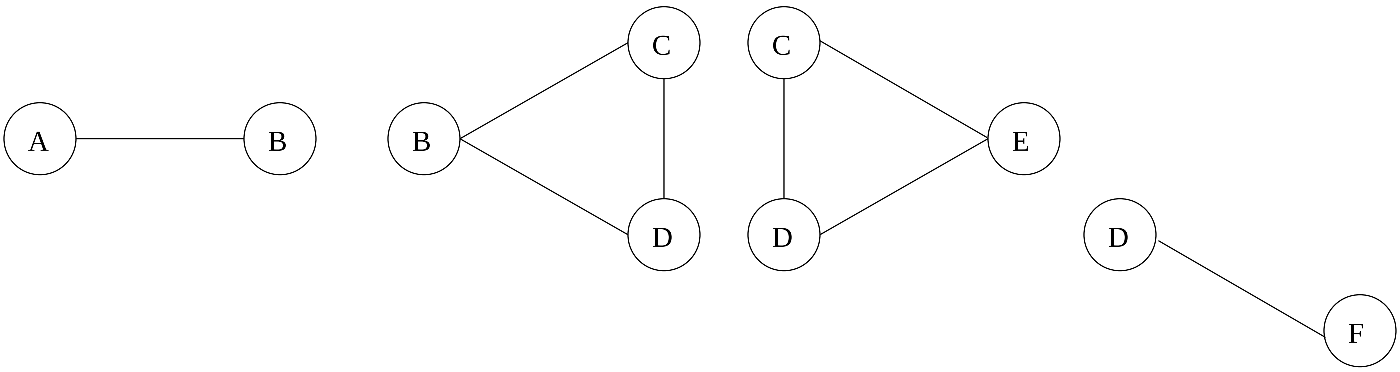

Figure 10: Cliques. *These subgraphs are all of the (maximal) cliques from the graph in Figure 9. These cliques are complete subgraphs (that is, they are fully connected).*

3 Inference in Bayesian Networks

Inference is the task of computing the probability of each value of a node in a Bayesian network when other variables' values are known. While there are different algorithms for doing inference in a Bayesian network, I will be looking at one called the junction tree algorithm, which works for any arbitrary Bayesian network. However, before describing this generic algorithm for arbitrary BNs, I will give a simple introduction to what inference involves using a *simple* Bayesian network. This will be then followed by an introduction to belief propagation, which serves as the foundation to the junction tree algorithm.

3.1 An Introduction to Inference

Heckerman (1999) gives an example of a Bayesian network that models credit card fraud, as illustrated in Figure 12. From this BN you can see, for example, that if there is a case of credit card fraud, then the chances of gas or jewelry being bought is affected; also, the chance of jewelry being bought is further affected by the age and sex of the purchaser. Table 1 indicates the probabilities associated

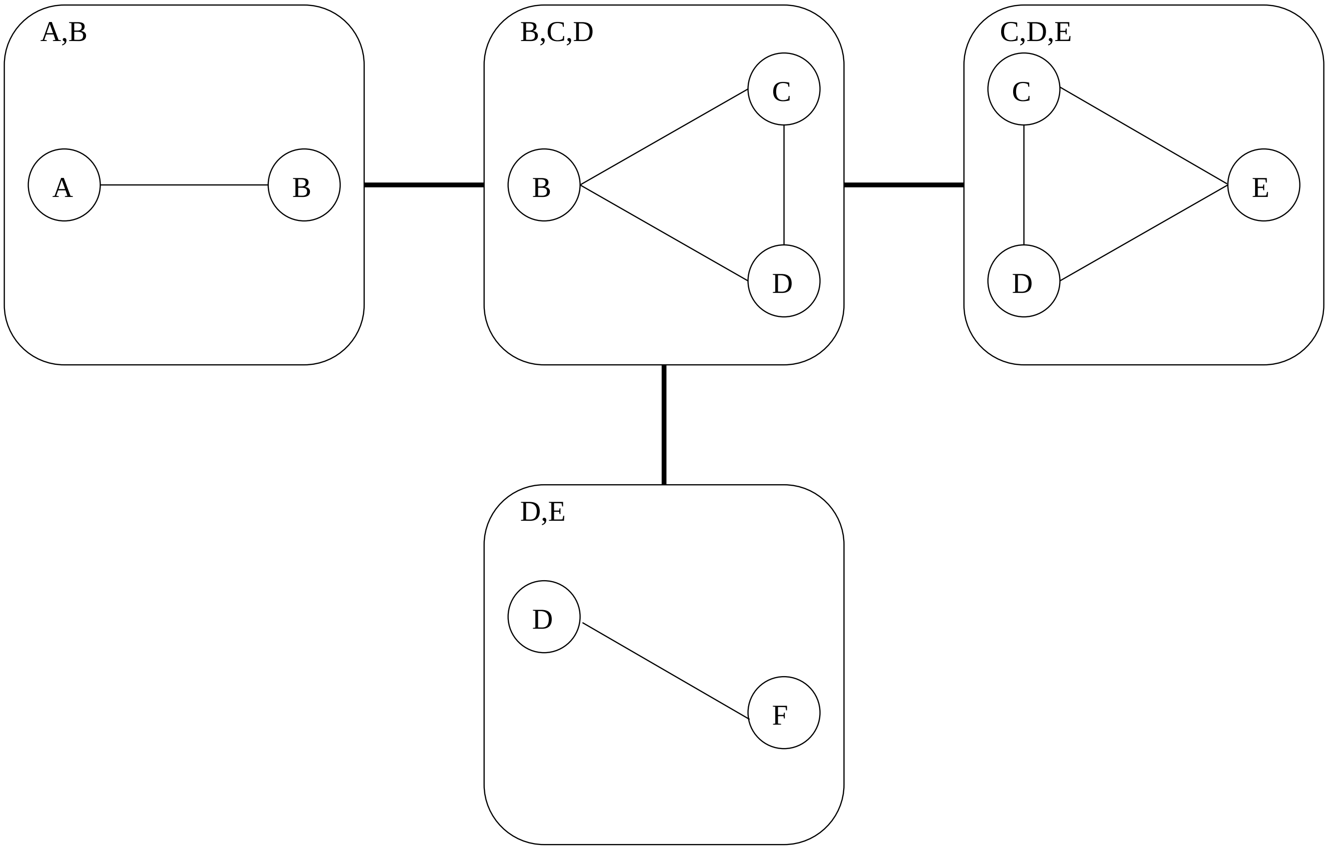

Figure 11: A Join Tree. *This is a join tree for the cliques in Figure 10. There are four nodes in this tree: (A,B), (B,C,D), (C,D,E), and (D,F).*

with each of the variables in this Bayesian network. For example, you can see that if $Fraud$ is Yes, then there is a probability of 0.2 that *Gas* will be Yes but if $Fraud$ is No, then there is a much smaller probability of 0.01 that *Gas* will be Yes. (That is, a person who is fraudulently using a credit card is twenty times more likely to buy gas than someone who is legitimately using a credit card). These different probabilities are commonly referred to as *beliefs* (Heckerman, 1999, Section 2).

Based on the discussion given in Heckerman (1999), inference in this Bayesian network proceeds as follows. Suppose that you notice a certain value for one or more of the variables in the network. If one variable has a definite (i.e., observed) value, your beliefs (i.e., probabilities) for the other variables need to be revised. This is what inference is: determining the updated (posterior) probability distribution for a variable based on the known values of the other variables. By itself, the prior probability for fraud is given as P(fraud=yes) = 0.00001. However, you notice that a young man is using a credit card to buy jewelry (but not any gas). That is, *Sex* = $Male$, *Age* =< 30, $Jewelry = Yes$, *Gas* = No (Table 2). You then want to infer whether he is using the card fraudulently. In other words, you need to calculate $P(f|j,g,s,a)$ (each letter is the first letter of its respective variable). You proceed as follows:

By Bayes' rule,

$$P(f|j,g,s,a) = \frac{P(j,g,s,a,f)}{P(j,g,s,a)} \tag{4}$$

Because the states of f are mutually exclusive and exhaustive, you can transform the denominator

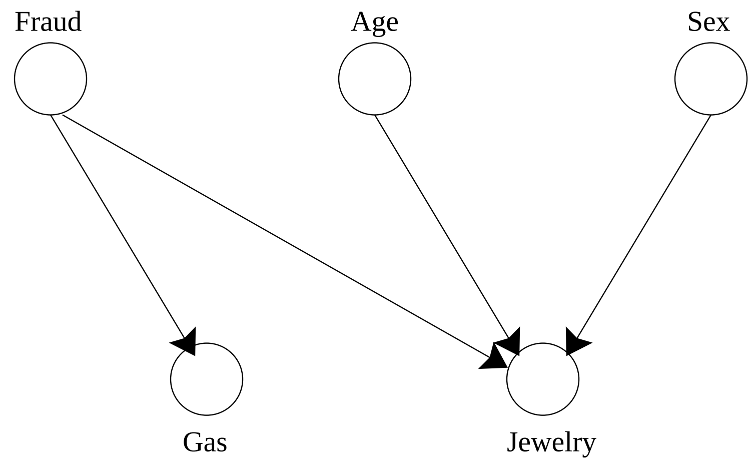

Figure 12: *A Bayesian network from Heckerman (1999) illustrating credit card fraud. There are five variables in this network. The probability of the Jewelry variable is dependent upon the values of the Fraud, Age, and Sex variables. Likewise, the value of the Gas variable is dependent upon the value of the Fraud variable. The Fraud, Age, and Sex variables themselves are not conditioned on any variable. See Table 1 for the local probability distributions for each variable.*

Probability			Conditions		
			Fraud	Age	Sex
Fraud = Yes	Fraud = No				
0.00001	0.99999		–	–	–
Age < 30	Age = 30-50	Age > 50			
0.25	0.40	0.35	–	–	–
Sex = Male	Sex = Female				
0.5	0.5		–	–	–
Gas = Yes	Gas = No				
0.2	0.8		Yes	–	–
0.01	0.99		No	–	–
Jewelry = Yes	Jewelry = No				
0.05	0.95		yes	*	*
0.0001	0.9999		no	<30	male
0.0004	0.9996		no	30-50	male
0.0002	0.9998		no	>50	male
0.0005	0.9995		no	<30	female
0.002	0.998		no	30-50	female
0.001	0.999		no	>50	female

Table 1: *Probabilities for the variables in Figure 12*

Variable	*Fraud*	*Jewelry*	*Gas*	*Sex*	*Age*
Value	?	*Yes*	*No*	*Male*	< 30

Table 2: *An example set of one unknown and four observations from the credit card fraud network in Figure 12. Values are observed for every variable except* $Fraud$*. These observed values can then be used to give an updated probability for the belief of* $Fraud$ *being* Yes*.*

of (4) to get:

$$P(f|j,g,s,a) = \frac{P(j,g,s,a,f)}{\sum_{f'} P(j,g,s,a,f')} \tag{5}$$

Now, using the product rule of probability, both the numerator and denominator of equation 5 can be factored as follows:

$$P(f|j,g,s,a) = \frac{P(j|g,s,a,f) * P(g|s,a,f) * P(s|a,f) * P(a|f) * P(f)}{\sum_{f'} P(j|g,s,a,f') * P(g|s,a,f') * P(s|a,f') * P(a|f') * P(f')} \tag{6}$$

Due to the conditional independencies, you can remove certain variables from the conditional lists in (6) (see Section 1.2). That is, if the variable x_0 is not a child of x_n in the graph, then

$$P(x_0|x_1,\ldots,x_n,\ldots,x_k) = P(x_0|x_1,\ldots,x_{n-1},x_{n+1},\ldots,x_k).$$

Using this simplification provides the following:

$$P(f|j,g,s,a) = \frac{P(j|s,a,f) * P(g|f) * P(s) * P(a) * P(f)}{\sum_{f'} P(j|s,a,f') * P(g|f') * P(s) * P(a) * P(f')} \tag{7}$$

At this point, you can then simplify equation 7 by cancelling common factors in the numerator and denominator. (Doing so indicates that the prior probabilities for both age and sex have no direct impact on the probability for fraud being computed).

$$P(f|j,g,s,a) = \frac{P(j|s,a,f) * P(g|f) * P(f)}{\sum_{f'} P(j|s,a,f') * P(g|f') * P(f')} \tag{8}$$

(8) can then be computed by inserting the values for the variables and reading the probabilities from Table 1.

$$P(f = Yes|j = Yes, g = No, s = Male, a =< 30)$$

$$= \frac{P(j = Yes|s = Male, a =< 30, f = Yes) * P(g = No|f = Yes) * P(f = Yes)}{\sum_{f' \in (yes,no)} P(j = Yes|s = Male, a =< 30, f') * P(g = No|f') * P(f')} \tag{9}$$

By substituting of the actual probabilities, you get:

$$P(f = Yes|j = Yes, g = No, s = Male, a =< 30)$$

$$= \frac{0.05 * 0.8 * 0.00001}{(0.05 * 0.8 * 0.00001) + (0.0001 * 0.99 * 0.99999)}$$

$$= 0.00402$$

Thus, while the prior probability of fraud is 0.00001, the inferred probability of fraud, given the other variables, is 0.00402 (over 400 times as probable).

3.2 Belief Propagation

Belief propagation (Pearl, 1988) is the action of updating the beliefs in each variable when observations are given to some of the variables. While this can be done in any arbitrary Bayesian network, I will illustrate it here for the simpler case of the Bayesian network being structured as a tree. In Section 9 on page 24 I give the more detailed algorithm of belief propagation for any general Bayesian network.

3.2.1 Pearl's Method

According to Pearl (1988, Section 4.2), belief propagation proceeds as follows. Let $\mathbf{e}$ be the set of values for all the observed variables. For any variable X, $\mathbf{e}$ can be split into two subsets: $\mathbf{e}^-_{\mathbf{X}}$, which represents all of the observed variables that are descendants of X (including X itself if it is observed), and $\mathbf{e}^+_{\mathbf{X}}$, which represents all of the other observed variables. The impact of the observed variables on the beliefs of X can then be represented using the following two values:

$$\lambda(X) = P(\mathbf{e}^-_{\mathbf{X}}|X) \tag{10}$$

$$\pi(X) = P(X|\mathbf{e}^+_{\mathbf{X}}) \tag{11}$$

Since x has multiple discrete values, $\lambda(X)$ and $\pi(X)$ are actually vectors, whose elements are associated with each of the discrete values for X:

$$\lambda(X) = [\lambda(X = x_1), \lambda(X = x_2), \ldots, \lambda(X = x_l)] \tag{12}$$

$$\pi(X) = [\pi(X = x_1), \pi(X = x_2), \ldots, \pi(X = x_l)] \tag{13}$$

Using (10) and (11), you get the following posterior probability:

$$P(X|\mathbf{e}) = \alpha \cdot \lambda(X) \cdot \pi(X) \tag{14}$$

with $\alpha = \frac{1}{P(\mathbf{e})}$, and with pairwise multiplication of the items in $\lambda(X)$ and $\pi(X)$. (14) is the equation that you want to compute to find the new beliefs. The $\lambda(X)$ and $\pi(X)$ values are then passed between variables in an orderly fashion.

Computing λ $\lambda(X)$ is computed using $\lambda(Y_1), \lambda(Y_2), \ldots, \lambda(Y_m)$, where $Y_1, Y_2, \ldots, Y_m$ are all of the children of X. First, when X is observed as x^0, the elements of the vector $\lambda(X)$ are assigned as follows:

- $\lambda(x_i) = 0 \quad$ if $\quad x_i \neq x^0$
- $\lambda(x_i) = 1 \quad$ if $\quad x_i = x^0$

For the case where X is not observed, you have $\mathbf{e}^-_{\mathbf{X}} = \bigcup_{i=1}^m e^-_{Y_i}$. Using (10), $\lambda(X)$ is expanded as:

$$\begin{aligned} \lambda(X) &= P(\mathbf{e}^-_{\mathbf{X}}|X) && (15) \\ &= P(\mathbf{e}^-_{\mathbf{Y_1}}, \mathbf{e}^-_{\mathbf{Y_2}}, \ldots, \mathbf{e}^-_{\mathbf{Y_m}}|X) && (16) \\ &= P(\mathbf{e}^-_{\mathbf{Y_1}}|x) \cdot P(\mathbf{e}^-_{\mathbf{Y_2}}|X) \cdot \cdots \cdot P(\mathbf{e}^-_{\mathbf{Y_m}}|X) && (17) \\ &= \lambda_{Y_1}(X) \cdot \lambda_{Y_2}(X) \cdot \cdots \cdot \lambda_{Y_m}(X), && (18) \end{aligned}$$

using the fact that $e^-_{Y_1}, e^-_{Y_2}, \ldots, e^-_{Y_m}$ are conditionally independent of each other, and defining the following:

$$\lambda_{Y_i}(X) = P(\mathbf{e}^-_{\mathbf{Y_i}}|X).$$

For each $\lambda_{Y_i}(X)$, its computation proceeds as:

$$\begin{aligned} \lambda_{Y_i}(X) &= P(\mathbf{e}^-_{\mathbf{Y_i}}|X) && (19) \\ &= \sum_{y_i} P(\mathbf{e}^-_{\mathbf{Y_i}}, y_i|X) && (20) \\ &= \sum_{y_i} P(\mathbf{e}^-_{\mathbf{Y_i}}|y_i, X) \cdot P(y_i|X) && (21) \\ &= \sum_{y_i} P(\mathbf{e}^-_{\mathbf{Y_i}}|y_i) \cdot P(y_i|X) && (22) \\ &= \sum_{y_i} \lambda(y_i) \cdot P(y_i|X) && (23) \end{aligned}$$

So, this shows that to compute the value $\lambda(X)$, you only need the λ's from all of X's children as well as the conditional probabilities from all of X's children. Also, this means that to compute a variable's λ's, you need to first compute its children's λ's. Therefore, in a compact form, the vector $\lambda(X)$ is computed by:

$$\lambda(X) = \prod_{c \in \mathrm{children}(X)} \sum_{v \in c} \lambda(v) \cdot P(v|X) \tag{24}$$

Computing π Furthermore, $\pi(X)$ is computed using X's parent Y. Using (11):

$$\begin{aligned}
\pi(X) &= P(X|\mathbf{e}^+_\mathbf{X}) && (25)\\
&= \sum_{y_i} P(X, y_i|\mathbf{e}^+_\mathbf{X}) && (26)\\
&= \sum_{y_i} P(X|y_i, \mathbf{e}^+_\mathbf{X}) \cdot P(y_i|\mathbf{e}^+_\mathbf{X}) && (27)\\
&= \sum_{y_i} P(X|y_i) \cdot P(y_i|\mathbf{e}^+_\mathbf{X}) && (28)\\
&= \sum_{y_i} P(X|y_i) \cdot \pi(y_i) && (29)
\end{aligned}$$

So, this shows that to compute the value $\pi(x)$, you only need the π from X's parent as well as the conditional probabilities from X. Also, this means that to compute a variable's π, you need to first compute its parent's π.

3.2.2 Peot and Shachter's Method

Peot and Shachter (1991) provides a similiar propagation algorithm to Pearl's. A notable difference from Pearl's is the definition of the λ's and π's. Peot and Shachter (1991) defines them as follows:

$$\lambda(X) = P(\mathbf{e}^-_\mathbf{X}|X) \tag{30}$$

$$\pi(X) = P(X, \mathbf{e}^+_\mathbf{X}) \tag{31}$$

While (30) is the same as (10), (31) differs from (11) in that a joint probability, $P(X, \mathbf{e}^+_\mathbf{X})$, is used instead of a conditional probability, $P(X|\mathbf{e}^+_\mathbf{X})$. In contrast to (14) in Pearl's algorithm, this provides the following:

$$P(X, \mathbf{e}) = \lambda(X) \cdot \pi(X) \tag{32}$$

For more details on this method, see Peot and Shachter (1991).

3.3 Junction Tree

Cowell (1999) gives an introduction to a more general method of inference in Bayesian networks: the junction tree algorithm. Because of potential problems with doing inference in a DAG (because of the cycles when the directionalities are removed–see Pearl (1988, Section 4.4)), this algorithm is useful. To use it, a DAG is transformed into a tree whose nodes are cliques (cf. page 6). Any given node from the graph may occur in multiple cliques in this tree. This leads to the key property that this tree has: any node (or any subset of nodes) from the graph that appears in two different cliques, X_1 and X_2, must also appear in all of the cliques along the path that connects X_1 and X_2. This gives the tree the name of "junction tree" (Cowell *et al.*, 1999) The junction tree algorithm proceeds as follows (Cowell, 1999):

Structure	Observability	Method
Known	full	Sample statistics
Known	partial	EM or gradient ascent
Unknown	full	Search through model space
Unknown	partial	Structural EM

Table 3: *Learning methods depending on what is already known about the problem. Taken from Murphy and Mian (1999).*

1. Moralize the Bayesian network.

2. Triangulate the moralized graph.

3. Let the cliques of the triangulated graph be the nodes of a tree, which is the desired *junction* tree.

4. Propagate the λ and π values throughout the junction tree to do inference. This will provide the posterior probabilities for the unobserved variables.[2]

See Section 8 for the details on constructing the junction tree and Section 9 for details on performing propagation in the junction tree.

4 Learning in Bayesian Networks

Up to this point in the paper, I have assumed that everything in the Bayesian network was already learned. That is, in explaining how to do inference in Section 3, I showed how to do inference in a Bayesian network in which both the topology and the prior probability distributions were already known. In this section I will go into learning the topology and the probability distributions within a Bayesian network. Depending on the problem that is defined, either (or both) of these may be pre-defined by hand or may be learned. Table 3 breaks down the different possibilities of what has to be learned and what methods are to be utilized. Heckerman (1999) provides an introduction to some of the issues involved in learning within a Bayesian network.

4.1 Known structure and full observability

4.1.1 Likelihood method

As can be seen from Table 3, if you already know the structure of the Bayesian network and have a full sampling of the data, you determine the probability distributions by computing statistics from the data samples. What this means is that if you want to compute the probability of, say, $P(X_1|X_2, X_3)$, you collect the data for all the various possible values of X_1, X_2, and X_3 and use the data to estimate the actual probability. That is,

$$P(X_1 = x_1|X_2 = x_2, X_3 = x_3) = \frac{P(X_1 = x_1, X_2 = x_2, X_3 = x_3)}{P(X_2 = x_2, X_3 = x_3)} \tag{33}$$

$$\approx \frac{\text{Number of samples with} \quad X_1 = x_1, X_2 = x_2, X_3 = x_3}{\text{Number of samples with} \quad X_2 = x_2, X_3 = x_3} \tag{34}$$

[2]Note that each λ and π in a *junction* tree can actually have multiple dimensions, each dimension representing a different variable in the clique that the λ and π are associated with. This multi-dimensional array has an element for each possible instantiation of the component variables for the respective clique.

4.1.2 Bayesian method

If the data is sparse, then Dirichlet priors can be used (Murphy and Mian, 1999). Heckerman (1999) goes into the complete Bayesian solution. In the Bayesian method, the local probability distribution for each variable is calculated by:

$$p(X_i|\text{parents}(X_i), \theta_\mathbf{i}, S^h), \tag{35}$$

where $\theta_\mathbf{i}$ are the parameters for the probability distribution of X_i and S^h is the topology of the Bayesian network. The $\theta_\mathbf{i}$ for a variable can be viewed as a two-dimensional matrix: one dimension for each possible instantiation of X_i and one dimension for each possible instantiation of parents(X_i). So, each probability parameter for the whole Bayesian network is identified by θ_{ijk}, where i ranges over all the variables, j ranges over all the possible parent instantiations for variable x_i, and k ranges over all the instantiations for X_i itself. In other words,

$$p(x_i^k|\text{parents}^j(X_i), \theta, S^h) = \theta_{ijk} \tag{36}$$

The goal of the training is to maximize the posterior distribution

$$p(\theta|D, S^h). \tag{37}$$

(37) can then be factored as follows, assuming parameter independence:

$$p(\theta|D, S^h) = \prod_{i=1}^{|\mathbf{X}|} \prod_{j=1}^{|\text{parents}^j(X_i)|} p(\theta_\mathbf{ij}|D, S^h) \tag{38}$$

The factors in (38) can be represented by Dirichlet distributions:

$$p(\theta_\mathbf{ij}|D, S^h) = \text{Dir}(\theta_\mathbf{ij}|\alpha_{ij1} + N_{ij1}, \alpha_{ij2} + N_{ij2}, \ldots, \alpha_{ij|x_i|} + N_{ij|x_i|}) \tag{39}$$

In (39), N_{ijk} represents the number of times that the case ($X_i = k$, parents(X_i) $= j$) appears in D while α_{ijk} represents your prior *beliefs* about how often the case ($X_i = k$, parents(X_i) $= j$) would occur, not having seen D.

Heckerman (1999) then goes on to show how, using (39), you obtain:

$$p(\mathbf{X}|D, S^h) = \prod_{i=1}^{|X|} \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}}, \tag{40}$$

which results in the following update formula for the probability distribution parameters:

$$\theta_{ijk} = \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}}. \tag{41}$$

4.2 Known structure and partial observability

As shown in Table 3 on page 13, if you know the structure but have not observed all of the data, you can use expectation-maximization(EM) or gradient ascent to learn the parameters. Exact Bayesian methods are intractable in this scenario; so, approximations to the Bayesian methods are used.

4.2.1 Likelihood method

The likelihood method for learning probabilities in Bayesian networks with partial observability is the expectation-maximization (EM) algorithm (Lauritzen, 1995), which is a generalization of the Baum-Welch algorithm (Baum, 1972). Zweig (1998) discusses how to use EM in discrete Bayesian networks, upon which the following is based.

Expectation Extending (14) on page 11 to cliques, the probability of clique $\mathbf{Clique_i}$ having instantiation c_i given the observations $\mathbf{e}$ is

$$P(\mathbf{Clique_i} = c_{ia}|\mathbf{e}) = \alpha \cdot \lambda_{ia} \cdot \pi_{ia}, \tag{42}$$

where a ranges over the possible instantiations of $\mathbf{Clique_i}$. So, this means, that by converting the Bayesian network to a junction tree and running the inference algorithm on it, you can use (42) to compute the probabilities for each clique, given the observations.

Now, you want to have a list of counts N_{ijk} for each variable X_i with an instantiation of k and with its parents having an instantiation of j. So, to determine the value for an N_{ijk}, first identify a clique where $\mathbf{X_i}$ occurs with its parents. (Its parents need to be in the clique also so that $P(X_i|\text{parents}(X_i))$ can be computed from the clique probabilities.) If $\mathbf{V^i_{jk}}$ is the set of clique instantiations that have $X_i = k$ and parents$(X_i) = j$, then N_{ijk} is:

$$N_{ijk} = \sum_{w \in \mathbf{V^i_{jk}}} P(\mathbf{Clique_i} = w|\mathbf{e}) \tag{43}$$

(43) marginalizes out any variables in $Clique_i$ which are not part of X_i's family. The values of N_{ijk} can be accumulated with each training sample given to the Bayesian network.

Maximization Once all of the N_{ijk} values have been computed, they are then used to re-estimate the local, conditional probability distributions for each variable. That is,

$$P(X_i = k|\text{parents}(X_i) = j) = \frac{N_{ijk}}{\sum_k N_{ijk}} = \frac{N_{ijk}}{N_{ij}} \tag{44}$$

4.2.2 Bayesian methods

Exact methods for computing the probability distribution using Bayesian methods are intractable because the number of parameters increases exponentially with the amount of missing data (Cowell *et al.*, 1999). One alternative is to use maximum likelihood with prior counts. Like in Section 4.1.2, Heckerman (1999, Section 6.3) shows how prior counts α_{ijk} can be incorporated into learning with partial observability, giving a revised expectation-maximization algorithm. The revised EM algorithm is as given in Section 4.2.1, but with the following maximization step, revised from (44):

$$P(X_i = k|\text{parents}(X_i) = j) = \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}} \tag{45}$$

4.3 Unknown structure and full observability

Heckerman (1999) continues discussing learning in Bayesian networks with methods of learning structure with complete training data. Two things are needed for this task:

- a metric for comparing potential structures against each other
- a search algorithm for finding potential structures

While the list of potential structures could be determined by using all of the edges possible in the set of nodes, the size of this list grows exponentially with the number of nodes. Therefore, the search algorithm will only find likely candidates.

4.3.1 Structure comparison metric

One method for evaluating a potential structure is to compute the joint probability $p(D, S^h)$, for the data D and structure S^h. After factoring this and taking its log, you have the following to compute:

$$\log p(D, S^h) = \log p(D|S^h) + \log p(S^h), \tag{46}$$

which breaks the task down into computing the posterior probability (the likelihood) and the prior probability of the structure of the data.

One method for computing $\log p(D|S^h)$ is to use the Bayesian information criterion (BIC) (Schwarz, 1978):

$$\log p(D|S^h) \approx \log p(D|\hat{\theta_{\mathbf{S}}}, S^h) - \frac{d}{2} \log N, \tag{47}$$

where $\theta_{\mathbf{S}}$ are the network parameters, d is the dimensions of $g(\theta_{\mathbf{s}}) \equiv \log(p(D|\theta_{\mathbf{s}}, S^h) \cdot p(\theta_{\mathbf{s}}|S^h))$, and N is taken from the size of D. Heckerman (1999) relates BIC as being the negative of Minimum Description Length (MDL) (Rissanen, 1987).

Regarding how to determine the prior probability $p(S^h)$ from (46), there are different methods (Heckerman, 1999). One is to have an expert (e.g., a person) provide a set of possible structures and then to assign all of these an equal prior while assigning all structures not in this set a prior of 0. Another possibility is to set a prior network. The prior probability of any network is decreased depending on the deviation between a potential network and the defined prior network (the deviation being calculated according to a pre-determined metric).

4.3.2 Structure search methods

Since you are assuming that all of the data has been observed, you know what the nodes in the graph are. So, you just have to find the appropriate connections between the variables. Doing this over all the possibilities is NP-hard (Heckerman, 1999). So, there are different options available for doing a selective search among the possibilities in order to find a good model. One possibility is to do a greedy search.

In a greedy search, you have at each step a list of all the possible changes that can be made to the graph. These changes are of the following type: adding an edge, in any direction, to the graph; reversing the direction of any edge in the graph; and removing an edge from the graph. The list of possible changes is restricted to those that keep the graph as a DAG (see page 5 for a definition of DAG). So, at each step you chose the change which increases $\log p(S^h)$, the second element of (46), the most. You continue iterating until there is no change that will increase the probability. Each iteration will always provide a potential structure that has not been previously chosen (because all previously examined structures have a lower value for $\log p(S^h)$ than the current). See Heckerman (1999) for more details about this and other search algorithms.

4.4 Unknown structure and partial observability

The final learning scenario from Table 3 is when neither the structure nor the data are fully known. Friedman (1997) explains one method for learning both the network structure and the data, the model-selection expectation-maximization (MS-EM) algorithm. (In Friedman (1998), the algorithm is called the structural EM algorithm (SEM)).

Murphy and Mian (1999) give a summary of the structural EM algorithm as follows. You can describe learning both the structure and the variables as a combination of the methods from section 4.3 and of adding new variables/nodes. You iterate as follows: (1) add a new node to the network, representing a hidden variable; (2) for this given set of nodes, find the "best" (as good as you can get) network connections; (3) continue as long as the network keeps improving.

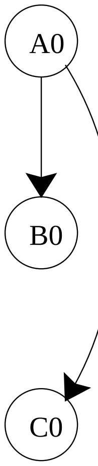

Figure 13: *A possible prior network for a dynamic Bayesian network with three static variables A, B, C. (Probabilities are omitted).*

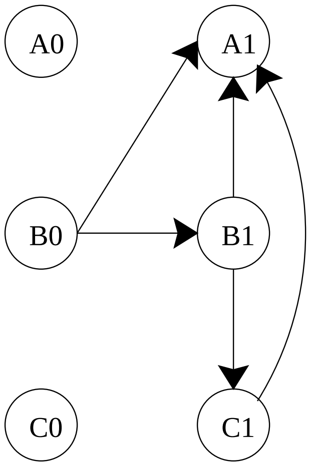

Figure 14: *A possible transition network for a dynamic Bayesian network with three static variables A, B, C. (Probabilities are omitted).*

5 Dynamic Bayesian Networks

In my previous discussions about Bayesian networks, none of them had any relation to time. That is, all the variables in the network were from a single point in time. Another domain of interest deals with how the variables in a Bayesian network change with time, which is what a dynamic Bayesian network (DBN) addresses. Figure 2 on page 3 illustrates what a *static* Bayesian network looks like. (None of the variables in it have reference to time; so it can be referred to as a static Bayesian network).

For a dynamic Bayesian network, the following needs to be defined: (1) a prior network and (2) a transition network (Friedman *et al.*, 1998). Figure 13 indicates a prior network. It represents the prior probabilities for all of the variables in the network at $t = 0$, the initial time slice. Figure 14 illustrates a transition network for the same dynamic Bayesian network. The transition network illustrates, for all time slices $t = 1, 2, \ldots, n$, what the probabilities are for each variable, conditioned on other variables (possibly from the previous time slice); that is, it gives the edges that lead into each variable.

For a dynamic Bayesian network, the set of variables and probability definitions are the same for each time slice (Zweig, 1998, Section 2.5.2), with the exception of the prior network in the initial

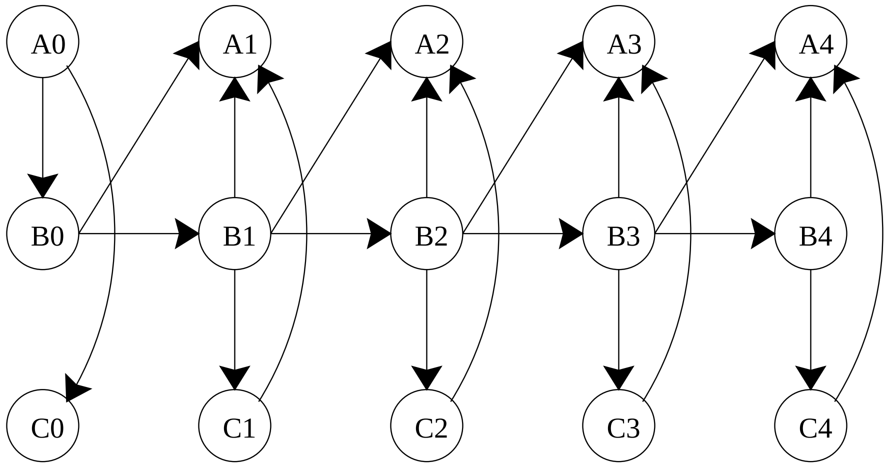

Figure 15: *A dynamic Bayesian network for five time slices. The connections for time* 0 *are from the prior network in Figure 13 while the connections for times* $1, 2, 3, \&4$ *are from the transition network in Figure 14. (Probabilities are omitted).*

time slice having its own probability distributions. So, given the prior and transition networks, you can construct a dynamic Bayesian network of any length as follows. You first supply the same set of (unconnected) variables for each time slice. Within time 0, the connections among the nodes and the local probability functions are the same as in the prior network. Then, for every time $t = 1, 2, \ldots, n$, the connections from $t-1$ to t, the connections within t, and the local probability functions within t are specified by the transition network. This configuration is assuming that this is a first-order Markov network (Zweig, 1998). Figure 15 on page 18 illustrates a dynamic Bayesian network for five time slices, built from the prior and transition networks in Figures 13 & 14.

6 Bayesian Networks for Automatic Speech Recognition

The area of application that most concerns my work is speech recognition. The application of dynamic Bayesian networks to automatic speech recognition (ASR) has been investigated in Zweig (1998).

6.1 Advantage of using DBNs in ASR

Three advantages pointed out by Zweig and Russell (1997) for using DBNs are (1) by exploiting the conditional independencies allowed by Bayesian networks, you can streamline the factorization of the joint probability and thus reduce the number of parameters, (2) the generality of the inference and learning algorithms allows a researcher to easily explore different topologies with the same, unmodified Bayesian network program, and (3) the tying of a variable across all time slices leads to better transitional modeling of things such as coarticulation in speech.

6.2 Topology of the Dynamic Bayesian Network in ASR

Based on Zweig (1998, Chapter 6) and Zweig and Russell (1997), the basic topology of a DBN that is analogous to the topology of a standard hidden Markov model (HMM) speech recognizer (Rabiner and Juang, 1993) has four variables (see Figure 16):

- **position** variable - this variable holds which part of the model (e.g., the 1st phoneme, 2nd phoneme, etc.) is associated with this time slice.
- **phone** variable - this variable gives the phone associated with the current position.

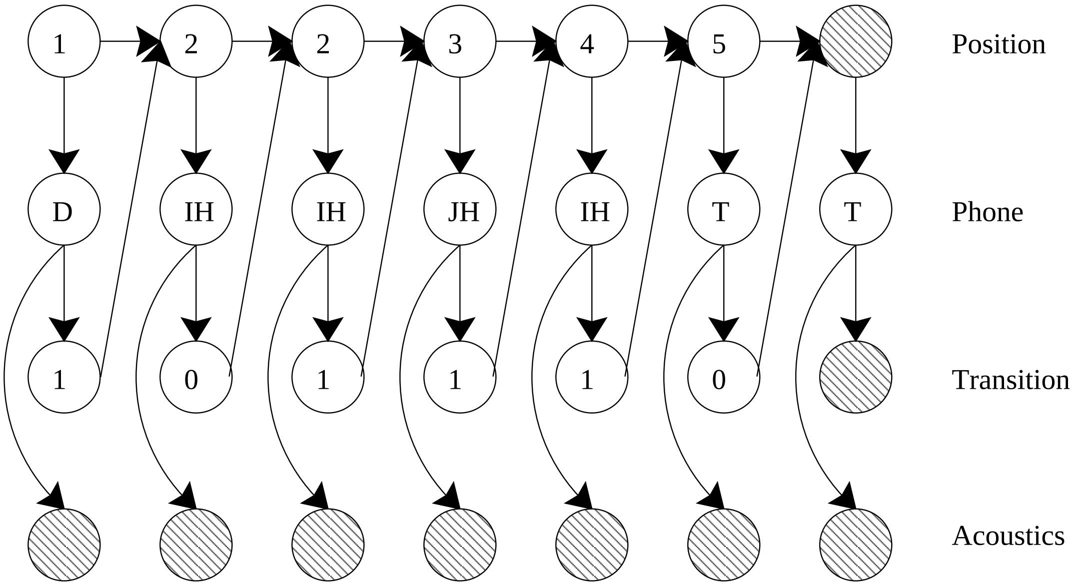

Figure 16: *The basic variables in a dynamic Bayesian network for speech recognition, based on Zweig (1998, Section 6.1) and Zweig and Russell (1997), that illustrate a word model for 'digit' with an utterance that covers seven time frames. For each time slice, the **position** variable indicates which part of the word the time slice is modeling, and this then determines what the value of **phone** will be. The **transition** variable indicates whether or not this was the last time slice for this position in the word. The **acoustics** variables are observed and are shaded to indicate such. The final position and transition variables are treated as observed (as 5 and true, respectively), under the assumption that when you reach the final time frame, you are in the final position in the word and are about to transition out of it (the final phone, in this example, is implicitly observed since it depends on the value of position).*

- **transition** variable - this variable indicates whether this is the last time frame for this phone. Based on the current phone variable, it gives the HMM equivalent of transition probabilities.
- **acoustic** variable - this variable holds the acoustic feature, which is typically observed in speech recognition applications. Based on the current phone model, it gives the HMM equivalent of emission probabilities.

7 Other Examples of Bayesian Networks

There are other areas of applications for Bayesian networks besides speech recognition. These include computer troubleshooting and medical diagnosis, among others.

7.1 Computer Troubleshooting

Perhaps the best known application of Bayesian networks in actual production is the printer problem troubleshooter for Microsoft Windows 95 (Heckerman *et al.*, 1995). Figure 17 on page 20 gives this Bayesian network, as taken from the Microsoft Belief Networks (MSBN) program[3]. The network is used to help an amateur computer user to determine the source of the problem when he has trouble

[3]`http://www.research.microsoft.com/msbn/default.htm`

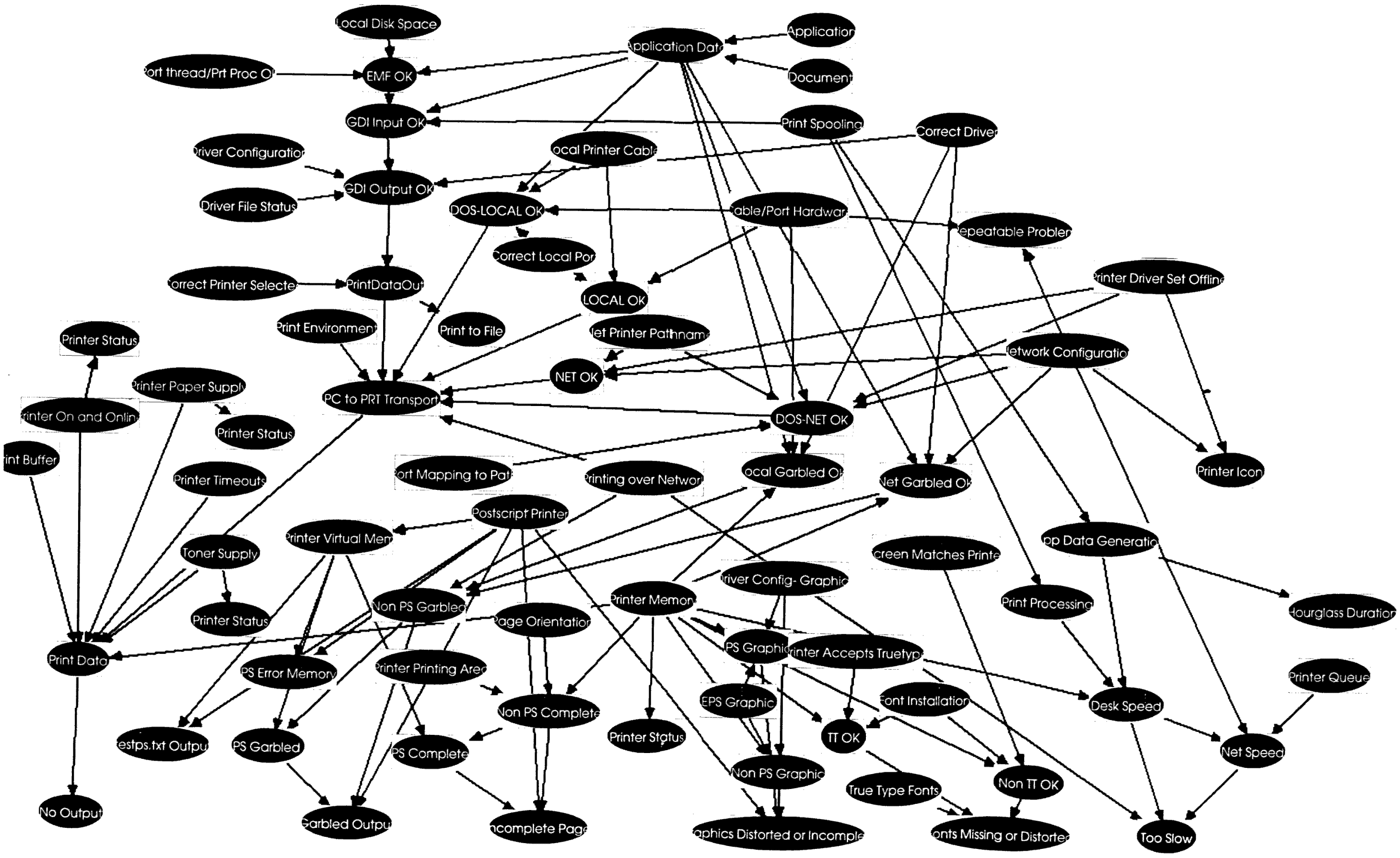

Figure 17: *The printer problem troubleshooter for Microsoft Windows 95, as taken from the Microsoft Belief Networks (MSBN) program. (Probabilities are omitted).*

printing. An interface to the Bayesian network enables the user to indicate what he observes about the problem. With certain variables observed, the tool then points the user to the most probable cause. Microsoft is continuing to deploy applications using Bayesian networks for troubleshooting[4].

7.2 Medical Diagnosis

By inputing observed symptoms, a Bayesian network for medical diagnosis can give the probability of different diseases that could be causing those symptoms In the network, the diseases are the causes, and the symptoms are the effects. So, if certain symptoms are observed, you can infer the probabilities for the different causes. This is an example of a network whose structure and probabilities do not necessarily have to be built by training methods; rather, an expert (such as a medical doctor) gives his own *belief* regarding the different probabilities and relationships relevant to the network. See Ògúnyemí (1999) for recent work that gives a detailed explanation of a sample medical system that is based on Bayesian networks.

[4] See the following for an example: `http://support.microsoft.com/support/tshoot/default.asp`

Part II
Algorithms

8 Constructing the junction tree

8.1 Moralizing the Bayesian network

Moralizing (as defined on page 5) a Bayesian network is concerned with "marrying" the parents of each child. The procedure is as follows. For every child in the initial graph (such as the graph in Figure 18), add undirected edges between all of its parents from the initial graph. Each parent of a given node should then be directly connected to each other parent of that given node (Jensen, 1996). Then, make all of the original edges in the entire graph undirected. So, you now end up with an undirected graph (such as in Figure 19).

8.2 Triangulating the graph

Triangulation is the process of making a graph triangulated (as defined on page 6). To do so, you first need to number the vertices of the graph. Any arbitrary numbering of the nodes will work. The resultant ordering, $1, \ldots, n$, can then be used to make the triangulated graph. The following algorithm, taken from Pearl (1988) but originally from Tarjan and Yannakakis (1984), is how to use this numbering to construct the triangulated graph (See Figure 20). Proceeding from node n, decreasing to node 1:

1. Determine the lower-numbered nodes which are adjacent to the current node, including those which may have been made adjacent to this node earlier in this algorithm.

2. Connect these nodes to each other.

Instead of choosing an arbitrary numbering of the nodes in order to do the triangulation, you can also use a maximum cardinality search (Neapolitan, 1989) (see Figure 20):

1. Give any node a value of 1

2. For each subsequent number, pick an unnumbered node that neighbors the most already numbered nodes (if there is a tie, you can pick any of the nodes in the tie).

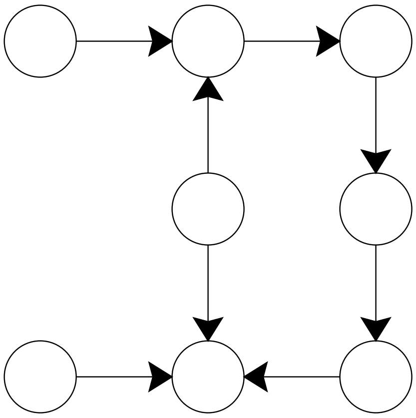

Figure 18: *A DAG*

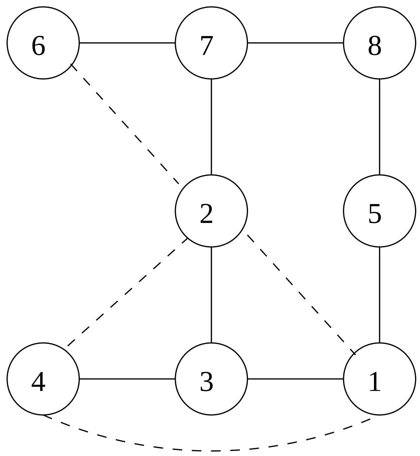

Figure 19: *The moralized version of the DAG in Figure 18 (with the vertices numbered according to a maximum cardinality search after the graph was moralized)*

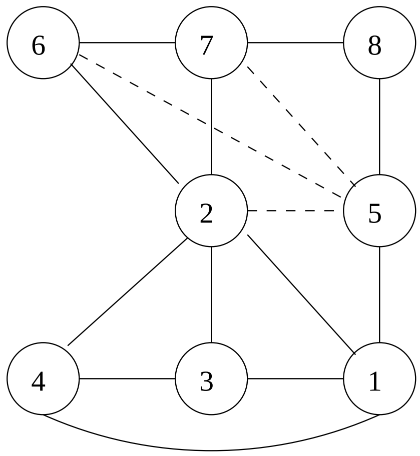

Figure 20: *The triangulated graph from Figure 19*

One advantage of using the maximum cardinality search is that it can also be used as a test to determine if a graph is already triangulated. That is, if you use a numbering from a maximum cardinality search to do the triangulation, you will only have to add an edge if the graph is not already triangulated.

8.3 Constructing the junction tree

Having a triangulated graph, you can now move into constructing the junction tree. As a preliminary step, you need an algorithm for producing a perfect vertex elimination scheme from a triangulated graph. A perfect vertex elimination scheme is where each vertex subset $X_i = \{v_j \in Adj(v_i) | j < i\}$ is complete (as defined on page 6 (Golumbic, 1980, Section 4.2)). This means that the lower numbered neighbors of any given vertex form a complete subgraph (the numbering used in the triangulation step meets this criterion). This definition from Golumbic (1980) is modified so as to conform with the conventions that Neapolitan (1989) uses, who numbers the nodes from $1, 2, \ldots, n$ as opposed to from $n, \ldots, 2, 1$.

The following algorithm for producing the junction tree is based on Golumbic (1980, Section 4.7). It produces a list of maximal cliques.

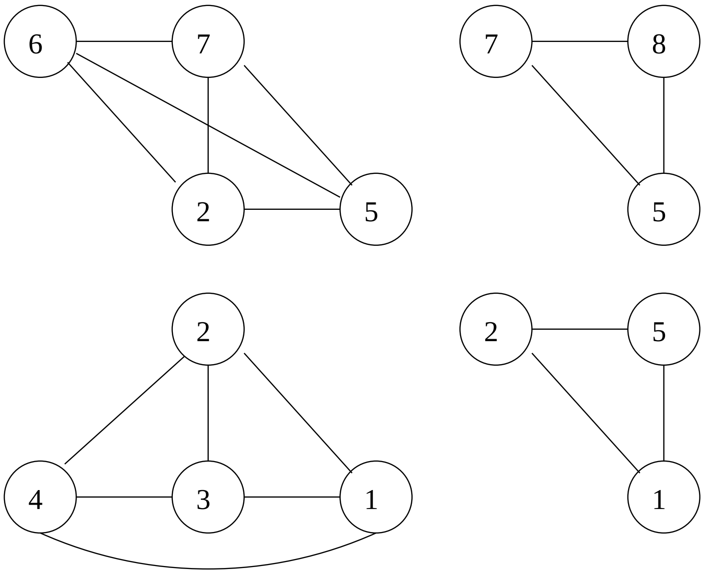

Figure 21: *The cliques from Figure 20. Note that it is common for variables to occur in multiple cliques.*

Given: Graph $G = (V, E)$
Given: ordering of vertices $v_1, \ldots, v_n$: σ (a perfect elimination order)
Initialize: $\chi = 1; S[1, \ldots, n] = [0, \ldots, 0]$, Cliques = []
for each decreasing vertex v in σ
    if v has neighbors
        if any of v's neighbors occur earlier than v in σ
            $X = v$'s earlier occurring neighbors in σ
            $u =$ the last occurring element of X
            $S[u] =$ the greater of either $S[u]$ or $|X| - 1$
            if $S[v] < |X|$
                $Cliques = Cliques + X, v$
                $\chi =$ the greater of either χ or $1 + |X|$
        else
            break out of **for** loop
    else
        $Cliques = Cliques + v$
return $Cliques, \chi$

Having a list of the cliques in the graph, construct the junction tree (Pearl, 1988, Section 3.2.4):

1. Sort the clique list according to the highest numbered node (using the *maximum cardinality* ordering) in the clique.

2. Connect each clique to a previous clique in the sorted list with which it shares the most number of nodes.

9 Inferring in the junction tree

In sections 3.2.1 & 3.2.2 I mentioned two different approaches to belief propagation. In the following pages I will give more details about how to perform belief propagation according to the approach used

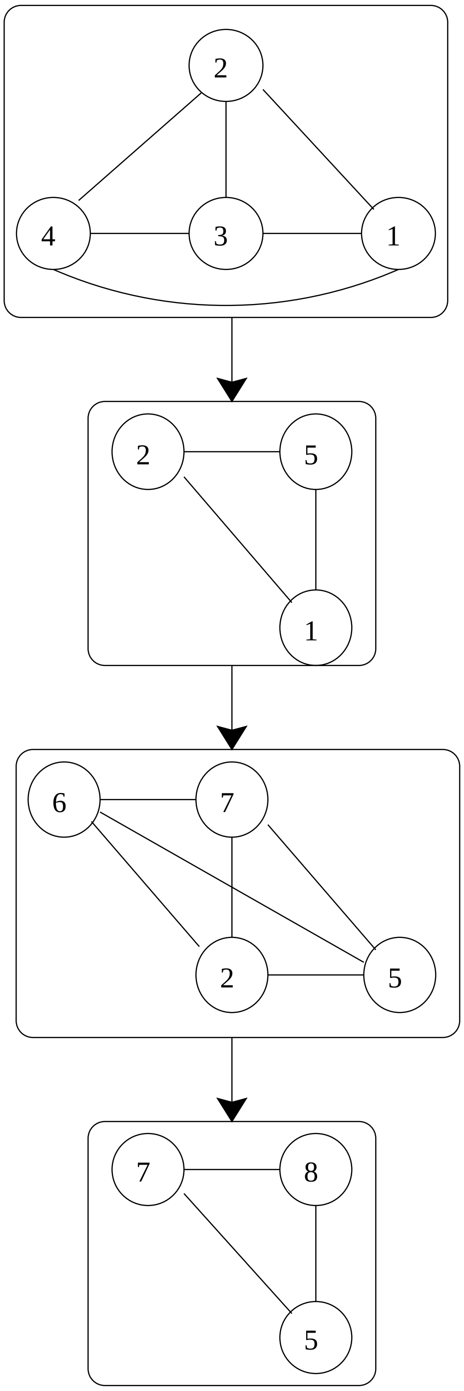

Figure 22: *The cliques from Figure 21 formed into a clique tree. Note that for any given variable x in the tree, that all of the cliques that it occurs in are connected together (a.k.a., running intersection property). While this clique tree is a chain, nodes in a tree in general are allowed to have more than one child.*

in Pearl (1988). For details about the other method, you can find details both in Peot and Shachter (1991) and in Zweig (1998), who applies this method to automatic speech recognition.

9.1 The clique potential

Having set out the structure of the junction tree, you now determine the potential of each of its cliques (Neapolitan, 1989). Each clique has a set of potentials, ψ, which represents the cross product of all the possible values of its nodes. Specifically, $Clique_i$ is composed of r_i nodes: $X_1, X_2, \ldots, X_{r_i}$. Furthermore, each of these nodes takes on $s^t_{r_i}$ values. Then there are $\prod_{r=1}^{r_i} s^t_r$ different combinations of variable instantiations across all the r_i variables in $Clique_i$. Let ψ_i represent the ψ values for $Clique_i$, and let ψ_{ij} represent the j^{th} of all the possible variable instantiation combinations for $Clique_i$.

for each $vertex_l$
    Assign a $vertex_l$ to any single clique where it occurs with its parents
for i $= 1, \ldots, N$ (N =the number of cliques):
    for $j = 1, \ldots, J$ (J =the number of possible variable instantiation combinations in $Clique_i$)
        Initialize ψ_{ij}, the clique's potential for this combination, to 1.
    for $j = 1, \ldots, J$
        For each $vertex_l$ assigned to $Clique_i$
            $\psi ij = \psi ij * P(vertex_{lj} | parents(vertex_l)_j)$

So, while a variable may occur in multiple cliques, its probability distribution is initially included in the calculation of only one clique's potential.

9.2 Inferring in the junction tree

Inferring is now done on a copy of the junction tree by the passing of π and λ values between cliques in the tree. The π's and λ's are similar to the ψ's in that they hold values for each of the possible joint variable instantiation combinations of its member variables. Each clique generates one λ, which it sends to its parent clique (the root sends it nowhere). Furthermore, each clique generates a different π for each of its children cliques (if it has no children, it does not generate any π's). The member variables for the λ's and π's is the intersection of the variables in the sending clique and the variables in the receiving clique. If this intersection is $\{\emptyset\}$, then the λ or π will be a scalar instead of a vector. (Note that the λ generated by the root clique will be a scalar as it has no clique to pass its λ up to, thus giving a $\{\emptyset\}$ intersection of variables).

This is a modified version of what is found in Neapolitan (1989, section 7.3). For further clarifications on the meanings of the different steps of this algorithm and for a partially worked out example, please look there.

Definitions
$S_i = Clique_i \cap$ parent$(Clique_i)$
$R_i = Clique_i - S_i$
main
for every $Clique_i$
    Let $\boldsymbol{O_i}$ be the observed variable subset for $Clique_i$
        $CliqueH_i = Clique_i - \boldsymbol{O_i}$
        $\psi H_i = \psi_i(\boldsymbol{X} = \boldsymbol{O_i})$
        $Clique_i = CliqueH_i$
        $\psi_i = \psi H_i$
        $S_i = S_i - \boldsymbol{O_i}$
        $R_i = R_i - \boldsymbol{O_i}$

for every leaf $Clique_c$
    do $\lambda\_prop(c)$
whenever vertex $Clique_p$ receives a λ_c from a child $Clique_c$
    for each instantiation j of ψ_p's variables
        $\psi_{pj} = \psi_{pj} * \lambda_{ck}$ (where $k \subseteq j$, k being an instantiation of λ_c's variables)
    if $Clique_p$ is the root
        $P_p = \psi_p$
        if $P_j = 0$ for all instantiations of $Clique_p$'s variables
            exit (the joint probability is 0)
        $\pi\_prop(p)$
whenever vertex $Clique_c$ receives a π_p from its parent $Clique_p$
    for each instantiation j of P_c's variables
        $P_{cj} = \psi_{cj} \cdot \pi_{pk}$ (where $k \subseteq j$, k being an instantiation of π_p's variables)
    if $Clique_c$ is not a leaf
        $\pi\_prop(c)$
when all P_i are calculated
    for any variable v in $Clique_i$, to determine $P(v = v')$
        $P(v = v') = \sum_{w \in \boldsymbol{W}} P(Clique_i = w)$,
            ($\boldsymbol{W}$ being instantiations of $Clique_i$'s variables where $v = v'$)

$\lambda\_prop(c)$
if $R_c = \emptyset$
    $\lambda_c = \psi_c$
else if $S_c = \emptyset$
    $\lambda_c = \sum_{R_{cl}} \psi_{cl}$ (λ is a single value)
else
    for each instantiation k of S_c's variables
        $\lambda_{ck} = \sum_{R_{cl}} \psi_c$
        if $\lambda_{ck} \neq 0$
            for each instantiation l of ψ_c's variables, where $k \subseteq l$
                $\psi_{cl} = \psi_{cl} / \lambda(S_{ck})$
if $Clique_c$ is not the root
    pass λ_c to $Clique_c$'s parent clique

$\pi\_prop(p)$
for each child $Clique_c$ of $Clique_p$
    if $S_c = \emptyset$
        $\pi_c = 1$ (π_c is a single value)
    else if $Clique_p - S_c = \emptyset$
        $\pi_c = P(Clique_p)$
    else
        for each instantiation k of π_c's variables, where $k \subset l$
            $\pi_{ck} = \sum_{(Clique_p - S_c)_l} P(Clique_p)$
    pass π_c to $Clique_c$

10 Conclusions

In this paper I have given an introduction to some of the concepts surrounding discrete Bayesian networks. I identified some of the concepts from graph theory that serve as the basis for the algorithms used in Bayesian networks. These algorithms allow you to calculate updated (i.e. posterior) probabilities given your observed data. Learning algorithms for Bayesian networks were also introduced; these involve learning the local probability distributions and possibly also the topology of the network. Furthermore, these concepts from Bayesian networks can be extended to handle time series problems, including those relating to speech recognition. Finally, I also gave in detail some of the important algorithms for working with Bayesian networks.

References

Baum, L. E. (1972). An equality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, **3**, 1–8.

Boffey, T. B. (1982). *Graph Theory in Operations Research*. Macmillan Computer Science Series. The Macmillan Press, Ltd.

Brualdi, R. A. (1992). *Introductory Combinatorics*. Elsevier Science Publishing Co., Inc., New York, second edition.

Cowell, R. (1999). Introduction to inference for Bayesian networks. In Jordan (1999), pages 9–26.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., and Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer-Verlag New York, Inc.

Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. In J. Douglas H. Fisher, editor, *Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97)*, pages 125–133, Nashville, TN. Morgan Kaufmann Publishers, Inc., San Francisco.

Friedman, N. (1998). The Bayesian structural EM algorithm. In *Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*.

Friedman, N., Murphy, K., and Russell, S. (1998). Learning the structure of dynamic probabilistic networks. *UAI*.

Golumbic, M. C. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

Heckerman, D. (1999). A tutorial on learning with Bayesian networks. In Jordan (1999), pages 301–354.

Heckerman, D., Breese, J., and Rommelse, K. (1995). Decision-theoretic troubleshooting. *Communications of the ACM*, **38**(3), 49–56.

Jensen, F. V. (1996). *An Introduction to Bayesian Networks*. UCL Press Ltd., London.

Jordan, M. I., editor (1999). *Learning in Graphical Models*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, first mit press edition.

Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, **19**, 191–201.

Murphy, K. and Mian, S. (1999). Modelling gene expression data using dynamic Bayesian networks. Technical report, Computer Science Division, University of California, Berkeley, CA.

Neapolitan, R. E. (1989). *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. A Wiley-Interscience Publication. John Wiley & Sons, Inc., New York.

Ògúnyemí, O. I. (1999). *TraumaSCAN: Assessing Penetrating Injury with Abductive and Geometric Reasoning*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Francisco, California, revised second printing edition.

Peot, M. A. and Shachter, R. D. (1991). Fusion and propagation with multiple observations in belief networks. *Artificial Intelligence*, **48**, 299–318.

Rabiner, L. and Juang, B.-H. (1993). *Fundamentals of Speech Recognition*. PTR Prentice-Hall, Inc., Englewood Cliffs, NJ.

Rissanen, J. (1987). Stochastic complexity (with discussion). *Journal of the Royal Statistical Society, Series B*, **49**, 223–239 and 253–265.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, **6**, 461–464.

Tarjan, R. E. and Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM J. Computing*, **13**, 566–79.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics*. John Wiley & Sons Ltd., Chichester, UK.

Zweig, G. and Russell, S. (1997). Compositional modeling with DPNs. Technical Report UCB/CSD-97-970, Computer Science Division (EECS), University of California at Berkeley, Berkeley, CA.

Zweig, G. G. (1998). *Speech Recognition with Dynamic Bayesian Networks*. Ph.D. thesis, University of California, Berkeley.