Research Report 329
University of Chicago
Dept. of Statistics

Optimality Issues in Constructing a Markov Tree from Graphical Models

Russell Almond[1]
Augustine Kong[2]

Harvard University
Department of Statistics

November, 1991

This work was facilitated in part by Army Research Contract DAAL03-86K-0042. This report is a revision and update of material originally presented in Harvard University Technical Reports S-118 and S-121.

[1] Russell Almond is currently visiting at the University of Washington, Department of Statistics

[2] Augustine Kong is now at the University of Chicago Department of Statistics.

Optimality Issues in Constructing a Markov Tree from Graphical Models

Russell Almond and Augustine Kong

ABSTRACT

Several recent papers have described probability models which used graph and hypergraphs to represent relationships among the variables. Two related computing algorithms are commonly used to manipulate such models: the *peeling* algorithm which eliminates variables one at a time to find the marginal distribution of a single variable, and the *fusion and propagation* algorithm which simultaneously solves for many marginal distributions by passing messages in a *Tree of Cliques* whose nodes correspond to subsets of variables . The peeling algorithm requires an elimination order. As demonstrated in this paper, the elimination order can in turn be used to construct a Tree of Cliques for propagation and fusion. This paper addresses three computational issues: 1) The choice of elimination order determines the size of the largest node of the Tree of Cliques, which dominates the computational cost for the probability model using either peeling or fusion and propagation. We review heuristics for choosing an elimination order. (2) Inserting intersection nodes into the tree of cliques produces a *junction tree* which has a lower computational cost. We present an algorithm which produces a junction trees with a high computational efficiency. (3) *Augmenting* the tree of cliques with additional nodes can lead to a new tree structure which more clearly expresses the relationship between the original graphical model and the tree model.

Key Words: *Graphical Model, Tree of Cliques, Markov Tree, Junction Tree, Triangulation, Elimination Order.*

1. Introduction and Background

Of late, it has become increasingly obvious that algorithms designed for probabilistic manipulations in expert systems (Pearl[1988] and Lauritzen and Spiegelhalter[1988]) and those developed by geneticists to aid the analysis of pedigree data (Cannings *et al.*[1973]) are essentially the same. The common thread is that in both disciplines, the dependencies among the variables can be represented by *interaction* graphs. In general, the interaction graph is a set of nodes (or vertices)—representing variables—connected by edges—representing dependencies among variables. The variables corresponding to the nodes, depending on the application, can be genotypes of individuals or information from sensors. The interaction graph is important in three ways: (1) it provides a clear visual picture of how the joint probability distribution can be factored into simpler components, (2) it visually and mathematically represents independence conditions among the variables—specifically, variables which are not neighbors in the graph are conditionally independent given the variables separating them (Pearl[1988])—and (3) it can guide efficient local computation schemes.

This last use of the graph is particularly important as the most efficient computation plans can be found via graph theoretic manipulation alone, with little reference to the underlying probabilistic model. In fact, Bertelè and Brioschi[1972] use essentially the same graph manipulations to solve non-serial dynamic programming problems. While they use graphs to depict an additive factorization of an objective function, we use graphs to depict a multiplicative factorization of a probability distribution. Although the motivating problems are different, many of their graph theoretic results are directly relevant to our applications. Indeed, similar algorithms also work for cases where the joint relation among the variables is modeled by a belief function (Kong[1986], Dempster and Kong[1988], Shafer, Shenoy and Mellouli[1988], Almond[1989]) instead of a probability distribution. Shenoy and Shafer[1990] develop an axiomatic framework which incorporates all of the problems mentioned above.

One of the most efficient methods for calculating marginal distributions of complex graphical models is the *Fusion and Propagation Algorithm* (Dempster and Kong[1988], Lauritzen and Spiegelhalter[1988], Pearl[1988], Almond[1989], *etc.*). This algorithm can be thought of as a series of local processors arranged in a tree structure called the *Markov tree*. Each node (processor) in the tree combines incoming messages with local information to produce outgoing messages and marginal information about the variables. Each node has a local space of variables over which it does its processing. The challenge is to build a Markov tree which is optimal in the sense that it does as many calculations as possible over as small a space a possible, while at the same time maintains its consistency. That is the problem addressed in this paper. We will use a simple genetics example to illustrate the ideas while keeping in mind the broader applicability of the theory.

2. Background

This section illustrates, with a simple example, how a factorization of a multivariate probability distribution can be represented by a graph. This produces the *graphical model*. Several manipulations of the graphical model can produce marginal distributions. We describe one called *peeling* or *elimination* which can be used to construct marginal distributions from a graphical model. An improved tool for this task, the *fusion and propagation* algorithm discussed in Section 2.4, is based in part on the peeling algorithm.

2.1 Factorization of Probability Distributions

Let $X_1, ..., X_n$ be a collection of discrete random variables with joint outcome space $\Omega = \prod_{i \in \mathcal{V}} \Omega_i$ where $\mathcal{V} = \{1, \ldots, n\}$ is the index set of the variables. For $I \subset \mathcal{V}$, x_I denotes an element of the margin $\Omega_I = \prod_{i \in I} \Omega_i$, and x denotes an element of Ω. Let $\triangle$ be a set of non-empty subsets of $\mathcal{V}$. A probability measure p over Ω has potential representation $\langle \triangle, \psi \rangle$ if

$$p(x) = K^{-1} \prod_{I \in \triangle} \psi_I(x_I) \tag{2.1}$$

where

$$K = \sum_{x \in \Omega} \prod_{I \in \triangle} \psi_I(x_I) \tag{2.2}$$

is the normalization constant, and ψ_I are functions of x that only depend on x through x_I, the projection of x onto the margin Ω_I. The functions $\psi_I, I \in \triangle$, are referred to as *component potentials*, and $\triangle$ is the index set for the potentials.

Consider the family tree displayed in Figure 1a with 9 individuals. Let $X_i, i = 1, \ldots, 9$, denote the genotype of person i. The joint distribution of the genotypes has the factorization

$$p(x_1, \ldots, x_9) = p(x_1)p(x_2)p(x_5)p(x_6)p(x_4|x_1, x_2)p(x_5|x_1, x_2)p(x_7|x_3, x_4)p(x_8|x_5, x_6)p(x_9|x_7, x_8) \tag{2.3}$$

reflecting the fact that the genetic material of an offspring is inherited from the parents. Note that (2.3) is a potential representation with $\mathcal{V} = \{1, \ldots, 9\}$,

$$\triangle = \{\{1\}, \{2\}, \{5\}, \{6\}, \{1, 2, 4\}, \{1, 2, 5\}, \{3, 4, 7\}, \{5, 6, 8\}, \{7, 8, 9\}\} \tag{2.4}$$

and $K = 1$. The potentials themselves are the marginal and conditional distributions. For example, $\psi_{\{1\}}(x_1) = p(x_1)$ and $\psi_{\{1,2,4\}}(x_1, x_2, x_4) = p(x_4|x_1, x_2)$.

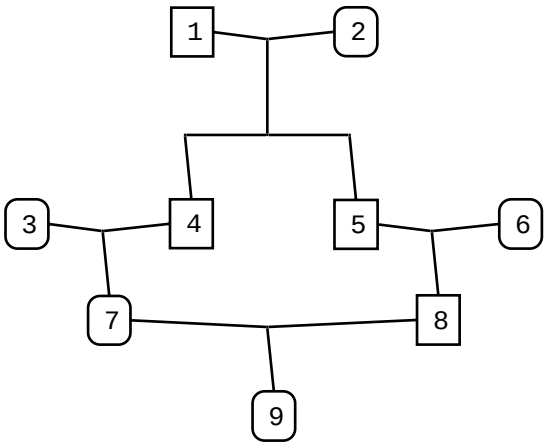


Figure 1a. Pedigree with married cousins

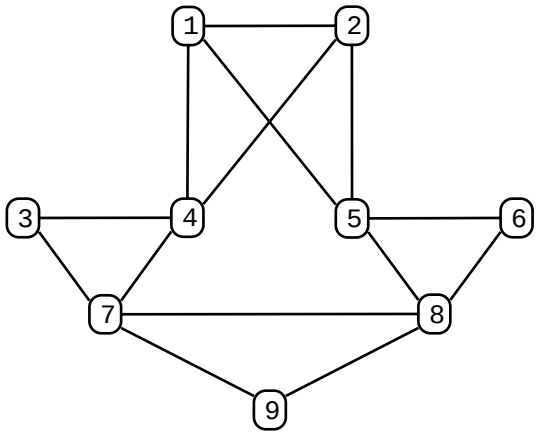


Figure 1b. Graphical Model representation

One strength of the potential representation is that it permits the simple introduction of new information about the variables through new potentials. For example, suppose some data are collected on the genotype of Person 9. Now the conditional joint distribution is

$$p(x_1,\ldots,x_9|\text{data}) = \frac{p(x_1,\ldots,x_9)p(\text{data}|x_1,\ldots,x_9)}{p(\text{data})} = \frac{1}{p(\text{data})}p(x_1,\ldots,x_9)p(\text{data}|x_9) \tag{2.5}$$

This has the potential representation

$$K^{-1}p(x_1)p(x_2)p(x_5)p(x_6)p(x_4|x_1,x_2)p(x_5|x_1,x_2)p(x_7|x_3,x_4)p(x_8|x_5,x_6)p(x_9|x_7,x_8)p(\text{data}|x_9) \tag{2.6}$$

where

$$K = p(\text{data}). \tag{2.7}$$

Here $\triangle$ is equal to

$$\{\{1\},\{2\},\{5\},\{6\},\{9\},\{1,2,4\},\{1,2,5\},\{3,4,7\},\{5,6,8\},\{7,8,9\}\} \tag{2.8}$$

which is (2.4) with an additional element $\{9\}$.

In general, given the potentials ψ_I of a potential representation, the standard problem is to compute either the marginal distributions of individual variables X_i, or K, the inverse of the normalization constant. Note that the the marginal distributions computed from the potential representation (2.6) are actually conditional on the observed data and K can be interpreted as the likelihood. To compute K by brute force based on (2.2) involves multiplying and adding numbers defined on the joint space Ω which usually is very large. Fortunately, in many cases, efficient computing strategies can be developed based on the graphical representation of the joint distribution. Geneticists analyzing pedigree data developed such computational algorithms under the name *peeling* (Hilden[1970], Cannings *et al.* [1972], Kong[1991a]). Operations researchers developed similar algorithms for non-serial dynamic programming under the name *elimination* (Bertelè and Brioschi[1972]). Although the calculations corresponding to the graph manipulations are different in the two cases, the underlying graph theory remains the same.

2.2 Graphs and Peeling (Elimination)

A *graph* $\mathcal{G}$ is a collection of *vertices* or *nodes* with *edges* or *links* joining some pairs of vertices. For the purpose of this paper, the vertices correspond to the variables X_i. Formally, a graph is a pair $\langle\mathcal{V},\mathcal{E}\rangle$ where $\mathcal{V}$ is the set of vertices (the index set of the variables in the model) and $\mathcal{E} = \{(i,j)|\exists$ an edge joining nodes i and $j\}$. If $\triangle$ is a set of subsets of $\mathcal{V}$, then the graph $\mathcal{G} = \langle\mathcal{V},\mathcal{E}\rangle$ which satisfies $\mathcal{E} = \{(i,j)|\exists I \in \triangle$such that $\{i,j\} \subset I\}$ is called the *2-section* of $\triangle$. For example, Figure 1b is the 2-section of (2.4). ($\triangle$ is actually a hypergraph, a generalization of a graph where "edges" are allowed to contain more than 2 vertices. For a discussion of this algorithm in terms of hypergraphs, see Kong[1986] or Almond[1989].) Note that different $\triangle$'s may share the same 2-section. In particular, elements of $\triangle$ which are singleton sets have no impact on the 2-section. Indeed, note that Figure 1b is also the 2-section of (2.8).

Two nodes of a graph $\mathcal{G}$ are called *neighbors* if there is an edge joining them. For a node i, the set of neighbors—the *neighborhood*—of i in $\mathcal{G}$ is denoted by $N(i|\mathcal{G})$. The closure of i in $\mathcal{G}$ is $Cl(i|\mathcal{G}) = \{i\} \cup N(i|\mathcal{G})$, that is the node and its neighborhood. For example, in Figure 1b, $N(5|\mathcal{G}) = \{1,2,6,8\}$, $Cl(5|\mathcal{G}) = \{1,2,5,6,8\}$, $N(9|\mathcal{G}) = \{7,8\}$ and $Cl(9|\mathcal{G}) = \{7,8,9\}$.

Suppose a joint distribution p, which may be conditional on data, has potential representation $\langle\triangle,\psi\rangle$. Let $\mathcal{G}$ be the 2-section of $\triangle$—the *graphical structure*—of p; the triple $\langle\triangle,\psi,\mathcal{G}\rangle$ is the *graphical model* of p. Given the potentials $\psi_I, I \in \triangle$, the *peeling* algorithm calculates the normalization constant K and the marginal distribution of one of the variables, say X_j. The peeling algorithm does this by "peeling away" the other variables one at a time.

Choose any variable $X_i, i \neq j$. For $i \in \mathcal{V}$, define $\triangle(i)$ to be $\{I|I \in \triangle, i \in I\}$ the set of potential indices containing i. Note that $\bigcup_{I\in\triangle(i)} I = Cl(i|\mathcal{G})$. Group the factors of the potential representation for p (2.1) into three pieces: the normalization constant, K^{-1}; all terms in $\triangle(i)$, $\prod_{I\in\triangle(i)}\psi_I(x_I)$; and all terms not in $\triangle(i)$,

$\prod_{I\in\triangle-\triangle(i)}\psi_I(x_I)$. Only the second group contains the variable X_i. To peel away X_i, replace this second group of potentials with a new potential defined over $\Omega_{N(i|\mathcal{G})}$, $\psi'_{N(i|\mathcal{G})}(x_{N(i|\mathcal{G})})$. To simplify the index set, if $N(i|\mathcal{G})\in\triangle$ we incorporate $\psi_{N(i|\mathcal{G})}$ into $\psi'_{N(i|\mathcal{G})}$. Specifically, the marginal distribution of $X_{\mathcal{V}-\{i\}}$—the joint distribution of all the variables except X_i—is written as:

$$p(x_{\mathcal{V}-\{i\}}) = K^{-1}\psi'_{N(i|\mathcal{G})}(x_{N(i|\mathcal{G})}) \prod_{I\in\triangle-\triangle(i), I\neq N(i|\mathcal{G})}\psi_I(x_I) \tag{2.9}$$

where

$$\psi'_{N(i|\mathcal{G})}(x_{N(i|\mathcal{G})}) = \begin{cases} \psi_{N(i|\mathcal{G})}(x_{N(i|\mathcal{G})})\sum_{x_i\in\Omega_i}\left(\prod_{I\in\triangle(i)}\psi_I(x_I)\right) & \text{if } N(i|\mathcal{G})\in\triangle \\ \sum_{x_i\in\Omega_i}\left(\prod_{I\in\triangle(i)}\psi_I(x_I)\right) & \text{otherwise} \end{cases} \tag{2.10}$$

The right hand side of (2.9) is a potential representation of $p(x_{S-\{i\}})$—$p(x_{S-\{i\}}) = \langle\triangle',\psi'\rangle$ where $\triangle' = (\triangle-\triangle(i))\cup\{N(i|\mathcal{G})\}$. This corresponds to a *reduced* graphical model $\mathcal{G}'$. Computing the new potential $\psi'_{N(i|\mathcal{G})}$ involves multiplying and summing numbers over the space $\Omega_{Cl(i|\mathcal{G})}$. The variable X_i which is summed out in expression (2.9) and (2.10) is said to be *peeled* or *eliminated* from the joint distribution.

The peeling procedure involves eliminating variables, one at a time, until only one variable is left. Let $\sigma = (i_n,\ldots,i_1)$, some permutation of the indices in $\mathcal{V}$, be the order the variables are peeled, where i_1 is the variable of interest. Denote the potential representation of the marginal distribution of $X_{\{i_1,\ldots,i_t\}}$ after the variables $(X_n, X_{n-1},\ldots,X_{n-t-1})$ have been peeled as $\langle\triangle^t,\psi^t\rangle$. Hence $\triangle^n=\triangle$ and for $t=n,\ldots,2$, $\triangle^{t-1} = (\triangle^t-\triangle^t(i_t))\cup\{N(i_t|\mathcal{G}^t)\}$ where $\mathcal{G}^t$ is the 2-section of $\triangle^t$. The peeling procedure computes a sequence of new potentials:

$$\psi^{t-1}_{N(i_t|\mathcal{G}^t)} = \begin{cases} \psi^t_{N(i_t|\mathcal{G}^t)}\sum_{x_t}\phi^{t-1} & \text{if } N(i_t|\mathcal{G}^t)\in\triangle^t \\ \sum_{x_t}\phi^{t-1} & \text{otherwise} \end{cases} \tag{2.11}$$

where

$$\phi^{t-1}(x_{Cl(i_t|\mathcal{G}^t)}) = \prod_{I\in\triangle^t(i_t)}\psi^t_I(x_I). \tag{2.12}$$

The sequence of spaces over which numbers are multiplied and summed are $\Omega_{Cl(i_t|\mathcal{G}^t)}$ for $t=n,\ldots,2$. After summing out all the variables except X_{i_1}, we get

$$p(x_{i_1}) = K^{-1}\psi^1_{\{i_1\}}(x_{i_1}), \tag{2.13}$$

the marginal distribution of X_{i_1}. Now K can be computed as

$$\sum_{x_{i_1}\in\Omega_{i_1}}\psi^1_{\{i_1\}}(x_{i_1}). \tag{2.14}$$

which we can think of as the peeling of the last variable. $\triangle^0$ is therefore the empty list and $\mathcal{G}^0$ is the empty graph.

Table 1 shows the peeling of the potential representation (2.6) following $\sigma=(9,6,3,8,7,5,4,2,1)$. Note that when Vertex 8 is peeled, an edge has to be added joining Vertices 5 and 7, and when Vertex 7 is peeled, an edge is added joining Vertices 4 and 5.

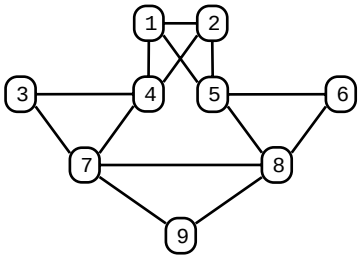
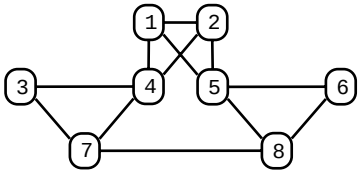
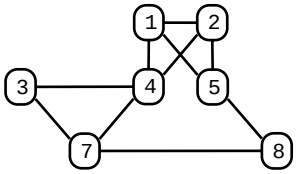
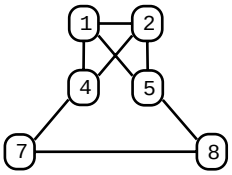
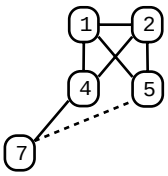
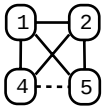
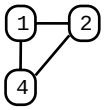
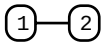
t	$\triangle^t$	$\mathcal{G}^t$	i_t	$\mathrm{N}(i_t \mid \mathcal{G}^t)$	$\mathrm{Cl}(i_t \mid \mathcal{G}^t)$
9	$\left\{ \begin{matrix} (1,2,4), (1,2,5), (3,4,7), (5,6,8), (7,8,9) \\ (1), (2), (3), (6), (9) \end{matrix} \right\}$		9	$\{7, 8\}$	$\{7, 8, 9\}$
8	$\left\{ \begin{matrix} (1,2,4), (1,2,5), (3,4,7), (5,6,8), (7,8) \\ (1), (2), (3), (6) \end{matrix} \right\}$		6	$\{5, 8\}$	$\{5, 6, 8\}$
7	$\left\{ \begin{matrix} (1,2,4), (1,2,5), (3,4,7), (5,8), (7,8) \\ (1), (2), (3) \end{matrix} \right\}$		3	$\{4, 7\}$	$\{3, 4, 7\}$
6	$\left\{ \begin{matrix} (1,2,4), (1,2,5), (4,7), (5,8), (7,8) \\ (1), (2) \end{matrix} \right\}$		8	$\{5, 7\}$	$\{5, 7, 8\}$
5	$\left\{ \begin{matrix} (1,2,4), (1,2,5), (4,7), (5,7) \\ (1), (2) \end{matrix} \right\}$		7	$\{4, 5\}$	$\{4, 5, 7\}$
4	$\left\{ \begin{matrix} (1,2,4), (1,2,5), (4,5) \\ (1), (2) \end{matrix} \right\}$		5	$\{1, 2, 4\}$	$\{1, 2, 4, 5\}$
3	$\left\{ \begin{matrix} (1,2,4) \\ (1), (2) \end{matrix} \right\}$		4	$\{1, 2\}$	$\{1, 2, 4\}$
2	$\{(1, 2),\ (1),\ (2)\}$		2	$\{1\}$	$\{1, 2\}$
1	$\{(1)\}$		1	$\{\}$	$\{1\}$

Table 1. Eliminations for the Pedigree example

For each stage t of the reduction of the graph, this table shows the successively smaller versions of the graph $\mathcal{G}^t$ and the list of factors $\triangle^t$ from which the variable i_t is removed. The table also shows the neighborhood of i_t, $\mathrm{N}(i_t \mid \mathcal{G}^t)$, and the closure $\mathrm{Cl}(i_t \mid \mathcal{G}^t)$. Edges filled-in during the elimination process are dotted.

2.3 Peeling Sequence and Computations

The order the variables are peeled can have a big effect on computations. Intuitively, the larger the size of the computation space when a variable is peeled, the higher the computational cost. More formally, for $J \subset \mathcal{V}$, define the *size* of J as

$$\prod_{i \in J} |\Omega_i|.$$

During peeling, the required memory and computations are related to to the sizes of the closures $Cl(i_t|\mathcal{G}^t)$, $t = n, \ldots, 1$. The exact relation depends on the application. However, in most cases, the order of magnitude of the amount of computations is determined by the size of the largest closures (see Kong [1988]). Hence a peeling order is "first order optimal" or "1-optimal" if it minimizes the size of the largest closures, and "optimal" if it minimizes the size of all the closures. Section 4 discusses optimality and computational cost issues in more detail.

In the genetics example, we may assume that $|\Omega_i|$ is the same for all i, and the size of any closure is (on the log scale) the number of variables in that closure. The peeling procedure summarized in Table 1, in which all closures contain at most four variables, is actually optimal. In contrast, it is not optimal if we start by peeling X_5 since $Cl(5|\mathcal{G}) = \{1, 2, 5, 6, 8\}$, involving 5 variables. In general, the task of finding an optimal peeling or elimination order is purely graph theoretic.

A better understanding of elimination orders requires the concept of a *triangulated* graph. A graph is said to be triangulated if there does not exist a *cycle* of length 4 or above which does not have a *chord*—an edge connecting two non-adjacent vertices in a cycle. For example, the graph in Figure 1b is *not* triangulated. The cycle (1,4,7,8,5,1) has length 5, because 5 distinct vertices are involved, and it does not have a chord. Recall that in the elimination process summarized in Table 1, two edges, $(5, 7)$ and $(4, 5)$, are added in the intermediate steps. In general, the elimination of a vertex i_t forces the addition of edges connecting those neighbors of i_t which are not already connected in the current graphical structure, $\mathcal{G}^t$. Filling-in these edges in the original graph produces the graph displayed in Figure 2.

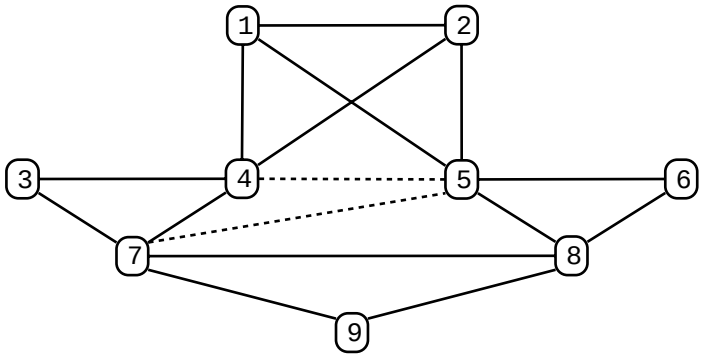


Figure 2. Filled-in (triangulated) model for pedigree data

Note that the graph in Figure 2 is triangulated. The same cycle (1,4,7,8,5,1) now has two chords, $(5, 7)$ and $(4, 5)$. This is not accidental. A graph created in this particular way—the *triangulated graph induced by a particular elimination order*, in this case (9,6,3,8,7,5,4,2,1)—is always triangulated. Let $\mathcal{G}_\sigma$ denotes the triangulated graph induced from $\mathcal{G}$ by the elimination order σ. Consider applying the elimination order σ to $\mathcal{G}_\sigma$. Because of how $\mathcal{G}_\sigma$ is constructed, it is clear that

$$Cl(i_t|\mathcal{G}_\sigma^t) = Cl(i_t|\mathcal{G}^t) \tag{2.15}$$

for all t. Also, $(\mathcal{G}_\sigma)_\sigma$, the induced graph when σ is applied to $\mathcal{G}_\sigma$, is equal to $\mathcal{G}_\sigma$ itself because no new edges is added in the elimination process. Indeed, an alternative definition to triangulated graphs is that a graph $\mathcal{G}$ is triangulated iff there exists an elimination order such that $\mathcal{G}_\sigma = \mathcal{G}$. Such a σ is called a *perfect* elimination order.

For any graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, a subset J of $\mathcal{V}$ is *completely connected* if all pairs of vertices in J are neighbors in $\mathcal{G}$. A *clique* is a maximal completely connected subset. For example, the graph in Figure 2 has six cliques: $\{1,2,3,4\}, \{4,5,7\}, \{5,7,8\}, \{3,4,7\}, \{5,6,8\}$ and $\{7,8,9\}$. Any graph is completely determined by its cliques. Let $\mathcal{G}$ be any graph, σ be some eliminating order and $\mathcal{G}_\sigma$ be the induced triangulated graph. Each of the closures $Cl(i_t|\mathcal{G}^t)$ which appeared during the elimination process is the subset of at least one of the cliques of $\mathcal{G}_\sigma$. It is also true that for any clique of $\mathcal{G}_\sigma$, there exists t such that $Cl(i_t|\mathcal{G}^t)$ is equal to that clique. These two facts (see Lemma 1 in Appendix I) imply that the largest closure is the same as the largest clique of $\mathcal{G}_\sigma$.

2.4 Propagation and Fusion

The peeling algorithm described above is essentially a clever way of exchanging the combination (multiplication) of potentials and the projection (summation) of potentials to efficiently compute the marginal distribution of one of the variables. Computing the marginal distributions of all n variables by applying the peeling procedure n separate times is clearly highly inefficient. Indeed, consider the peel procedure summarized in Table 1 which gives the marginal distribution of X_1. If we then compute of the marginal distribution of X_2 using the peeling order (9,6,3,8,7,5,4,1,2), the computations involve is identical to the earlier case up to the last one or two steps. The *fusion and propagation algorithm* (Pearl[1988], Dempster and Kong[1988], Lauritzen and Spiegelhalter[1988]) provides an efficient method for re-doing such calculations.

To understand propagation and fusion, we start from a triangulated graph resulting from some triangulation of the original graphical structure $\mathcal{G}$. The cliques of this triangulated graph are then organized in a tree structure (the exact method is described in Section 3). For example, Figure 3 (ignore the directions of the arrows at this moment) shows a tree of cliques corresponding to the triangulated graph in Figure 2. Each potential ψ_I in the potential representation is assigned to one of the cliques which contains I. Each clique $\mathbf{C} \in \mathcal{N}$, where $\mathcal{N}$ denotes the set of cliques, is then given a local value $\phi_{\mathbf{C}}$ equal to the product of all potentials assigned to it. A unit value (constant unit function) is assigned to any clique which has no assigned component potentials. Thus (2.1) becomes

$$p(x) = K^{-1} \prod_{\mathbf{C} \in \mathcal{N}} \phi_{\mathbf{C}}(x_{\mathbf{C}}) \tag{2.16}$$

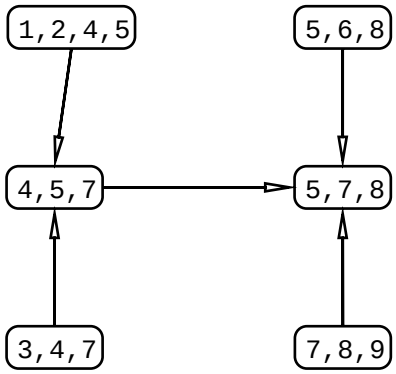


Figure 3a. Passing messages inwards

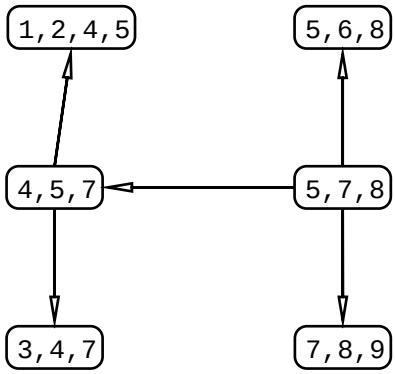


Figure 3b. Passing messages outwards

Figure 3 shows the message passing strategy for the simple pedigree example. The *leaves* of the tree (those nodes which are connected to exactly one other node) pass a message consisting of only their local information, thus the Nodes $\{1,2,4,5\}$, $\{3,4,7\}$, $\{5,6,8\}$ and $\{7,8,9\}$ are leaves and pass messages to their neighbors, as shown by the arrows of Figure 3a. As the message is passed, any variable which is not in

the receiving node is eliminated by summing out over the other variables. Thus, the message passed from Node $\{1,2,4,5\}$ to Node $\{4,5,7\}$ is

$$\phi_{\{1,2,4,5\}\Rightarrow\{4,5,7\}}(x_{\{4,5\}}) = \sum_{x_{\{1,2\}}\in\Omega_{\{1,2\}}} \phi(x_{\{1,2,4,5\}}) \ . \tag{2.17}$$

The Node $\{4,5,7\}$ combines all of its incoming messages with its local information and passes the result to its remaining neighbor, the Node $\{5,7,8\}$. After the all the messages corresponding to the arrows in Figure 3a are passed, the Node $\{5,7,8\}$ has all the information necessary to calculate the normalization constant and the marginal distribution over $\{5,7,8\}$. To obtain marginal distributions for other nodes the inner nodes pass messages (calculated in the same way) outwards along the path of Figure 3b. After both directions of propagation, each node has enough information to calculate both the normalization constant and its own local marginal distribution.

More generally, if a Node $\mathbf{C}_*$ in the tree of cliques has neighbors $\mathbf{C}_1, \ldots, \mathbf{C}_k$, then the message passed to from the Node $\mathbf{C}_*$ to the node $\mathbf{C}_j$ is

$$\phi_{\mathbf{C}_*\Rightarrow\mathbf{C}_j}(x_{\mathbf{C}^*\cap\mathbf{C}_j}) = \sum_{x_{\mathbf{C}_*-\mathbf{C}_j}\in\Omega_{\mathbf{C}_*-\mathbf{C}_j}} \left(\phi_{\mathbf{C}_*}(x_{\mathbf{C}_*}) \prod_{\substack{i=1,\ldots,k \\ i\neq j}} \phi_{\mathbf{C}_i\Rightarrow\mathbf{C}_*}(x_{\mathbf{C}_*\cap\mathbf{C}_i}) \right) \ ; \tag{2.18}$$

this is called the *fusion equation*. In essence, it is very simple. The message sent to Node $\mathbf{C}_j$ from the Node $\mathbf{C}_*$ is the product of the local potential and the messages coming in to Node $\mathbf{C}_*$ from every direction except the one from Node $\mathbf{C}_j$. This computation is done over the space $\mathbf{C}_*$ and then variables not in $\mathbf{C}_j$ are marginalized out by summing over them.

Fusion and propagation is just a clever way of simultaneously peeling all variables; that is, when a variable is not needed in one part of the tree, it is eliminated from the messages via marginalizations. This depends on the fact that under certain conditions, one can redistribute the marginalization (summation) and combination (multiplication) operations. These conditions (Shenoy and Shafer[1990]) essentially amount to a simple and obvious requirement: *if any two nodes in the tree of cliques contains a variable X, then all nodes lying on the (unique) path between those nodes must contain X.* Trees having this *separation property* are called *Markov Trees*.

Formally, let $\mathcal{T} = \langle \mathcal{N}, \mathcal{L} \rangle$ be a tree (graph with no cycles) in which the nodes in $\mathcal{N}$ are labeled as subsets of some index set $\mathcal{V}$. (To make the distinction clear, $\mathcal{T} = \langle \mathcal{N}, \mathcal{L} \rangle$ is referred to as a tree of *nodes* and *links* while $\mathcal{G} = \langle \mathcal{A}, \mathcal{E} \rangle$ is as a graph of vertices and edges.) The *separation property* is defined as: For any two nodes, $\mathbf{N}^1$ and $\mathbf{N}^2$, any other node $\mathbf{N}^3$ which lies in the path joining them must satisfy $\mathbf{N}^1 \cap \mathbf{N}^2 \subseteq \mathbf{N}^3$. A tree $\mathcal{T}$ which has the separation property is called a Markov Tree. Examining all nodes containing X_4 or X_5 shows that the tree in Figure 3 has the separation property. Section 3 describes how to build a tree of cliques which is a Markov tree.

2.5 Remarks

The importance of the graphical structure of the potential representation cannot be over emphasized. Apart from being a great tool for finding efficient computing strategies, it also provides a good summary of the relations among the variables. In particular, it follows from the Gibbs-Markov Equivalence Theorem (see Darroch, Lauritzen and Speed[1980]) that if two variables are separated by another set of variables in $\mathcal{G}$—any path connecting the two variables must include at least one member of the separating set—then the two variables are conditionally independent given the separating variables. For example, in Figure 1b it is immediately clear that X_1 and X_9 are conditionally independent given X_4 and X_5.

Note that many authors, especially Lauritzen and Spiegelhalter[1988], Pearl[1988], and Shacter[1986], use directed graphical models as opposed to the undirected ones discussed here. The direction of the arrows indicate causality (Pearl[1988]) or some weaker notion such as convenient conditional distributions (especially in the influence diagram school, Schacter[1986]). In either case, the arrows indicate that the

distribution of each "child" node (one at which the arrows point) is given conditional on its parents (the nodes from which the arrows come). For example, Figure 4a shows the directed graph model corresponding to Figure 1. To get from the directed model to the undirected model, one simply forms the list of potentials and takes its 2-section. This process is partially illustrated in Figure 4b where the potentials are drawn as a "hypergraph," each "hyperedge" circling the variables in one of the component potentials. This is the "moralization" process described in Lauritzen and Spiegelhalter[1988]. Although directed models may be more convenient for elicitation and assessment processes, taking the factorization of the problem as the primary representation has two advantages: (1) it eliminates the need for "moralization" and (2) it allows the same results to be used for functions other than probability distributions which do not have a well defined concept of conditioning.

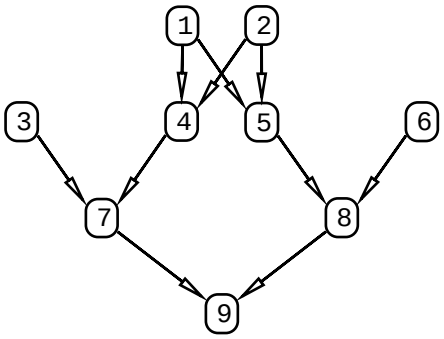


Figure 4a. Directed Model

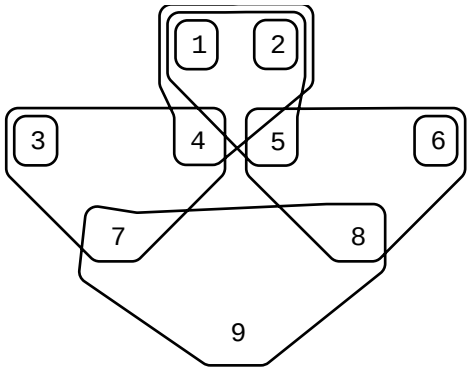


Figure 4b. Potential Hypergraph

3. Building a Markov Tree

Given a graph $\mathcal{G}$ and an elimination order σ, the cliques of the induced triangulated graph $\mathcal{G}_\sigma$ can be arranged into a Markov tree which supports the fusion and propagation algorithm. For this purpose, two elimination orders, σ and σ', are said to be *equivalent* if $\mathcal{G}_\sigma = \mathcal{G}_{\sigma'}$. Comparisons of non-equivalent elimination orders are addressed in the Section 4. This section starts by describing an algorithm for constructing a Tree of cliques given a graph and an elimination order, thn raises issues concerning the non-uniqueness of the Tree. An extended version of the tree of cliques—the *Junction tree*—partially resolves the issue of non-uniqueness and generally leads to more efficient computations.

3.1 Building the Tree of Cliques

Given a graph $\mathcal{G}$ and an elimination order σ, the procedure given below constructs a Markov Tree whose nodes corresponds to the cliques of the induced triangulated graph $\mathcal{G}_\sigma$. Note that it may sometimes be more natural to start the construction process with a graph which is already triangulated. In that case, the elimination order can be any perfect elimination order with respect to the triangulated graph.

To construct the Tree of Cliques we build a sequence of trees $\mathcal{T}^t, t = 1, \ldots, n$, moving *backwards* through the elimination order. At step 1, create a tree $\mathcal{T}^1$ with a single node labeled $\{i_1\}$. In general, at step t, create a node labeled $Cl(i_t|\mathcal{G}^t)$ in one of two ways. If the $\mathcal{T}^{t-1}$ has a node labeled $N(i_t|\mathcal{G}^t)$, then simply relabel that node as $Cl(i_t|\mathcal{G}^t)$. If not, create a new node labeled $Cl(i_t|\mathcal{G}^t)$ and attach it to an existing node $\mathbf{N}$ which satisfies $N(i_t|\mathcal{G}^t) \subset \mathbf{N}$. The existence of such a $\mathbf{N}$ is given by Lemma 2a in Appendix I. There can however be more than one existing node which contains $N(i_t|\mathcal{G}^t)$. This issue of non-uniqueness is addressed in Section 3.3. Lemmas 2a and 2b in Appendix I show that the final tree $\mathcal{T} = \mathcal{T}^n$ has nodes that correspond to the cliques of $\mathcal{G}_\sigma$ and also satisfies the separation property. Appendix II provides an implementation of this algorithm in Common Lisp.

The above construction algorithm is applied to the example displayed in Table 1, with the results presented in Figure 5. Note that although in this example all the relabelings occurred in the first part of the

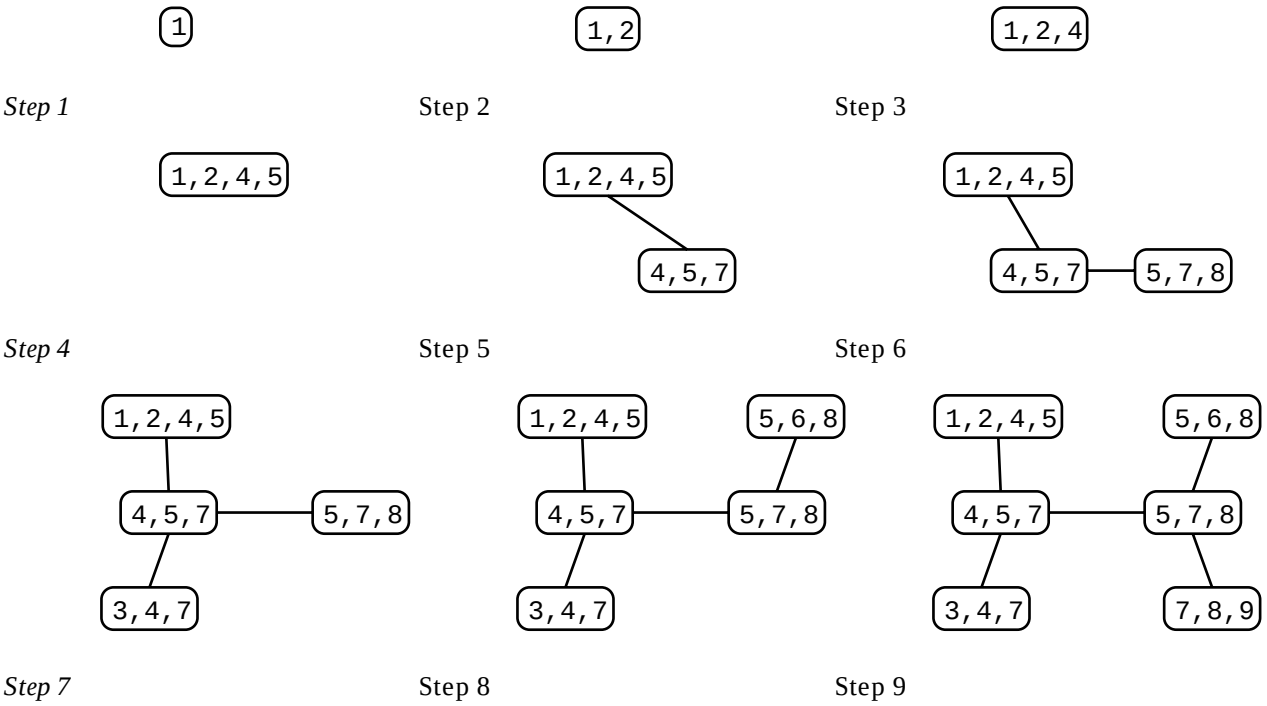


Figure 5. Tree of Cliques from Example Pedigree

construction, this does not hold for all graphs. Also, here the choice of neighbor for each new node added happens to be unique; Section 3.3 examines the case where this does not hold.

3.2 Augmenting the tree of cliques

The procedure outlined above takes a graphical structure $\mathcal{G}$ and produces a corresponding tree structure $\mathcal{T}$, but one step yet remains to turn the graphical model $\langle \triangle, \psi, \mathcal{G} \rangle$ into the Markov tree model $\langle \mathcal{V}, \phi, \mathcal{T} \rangle$ which is used in the fusion and propagation algorithm, that is the construction of the new potentials ϕ corresponding to the nodes of the tree. This is actually very simple. For each $I \in \triangle$ we choose a node $\tau(I)$ in the tree of cliques which is such that $I \subset \tau(I)$. The existence of the mapping $\tau : \triangle \to \mathcal{N}$ is guaranteed by the nature of the tree of cliques (Lemma 3 in Appendix I). The potential associated with a given node, $\mathbf{N}$, in the tree of cliques then becomes:

$$\phi(x_{\mathbf{N}}) = \prod_{I:\tau(I)=\mathbf{N}} \psi_I(x_I). \tag{3.1}$$

Any node $\mathbf{N}$ which is not in the image of $\triangle$ under τ is assigned a unit potential.

If the mapping τ was injective (one-to-one)—that is each component $I \in \triangle$ corresponds to a unique node $\tau(I)$ in the tree of cliques—then the Markov tree model would be "modular" in the sense that separate components of the problem were kept separate. An injective mapping is invertible, so the component associated with each node of the tree (if there is one) is readily identifyable. This is not in general the case. However, the fusion and propagation algorithm relies only on the fact that the tree of cliques is a Markov tree; in particular, the tree of cliques can be *augmented* by additional nodes as long as the separation property is not violated. It is always possible to add a node which is a subset of another node and have the result still be a Markov tree. As all of the elements of $\triangle$ are either a node in the tree or a subset of a node in the tree (Lemma 3), it is always possible to produces an augmented tree of cliques $\mathcal{T}'$ by adding new nodes corresponding to elements of $\triangle$.

Figure 6a illustrates the augmentation process. There are six components in $\triangle$ not included in the tree of cliques (Figure 5): $\{1,2,4\}, \{1,2,5\}, \{1\}, \{2\}, \{3\}$, and $\{6\}$. These are added as extra nodes, and connected to any node containing them. There is now a 1 to 1 mapping from the components of $\triangle$ to the

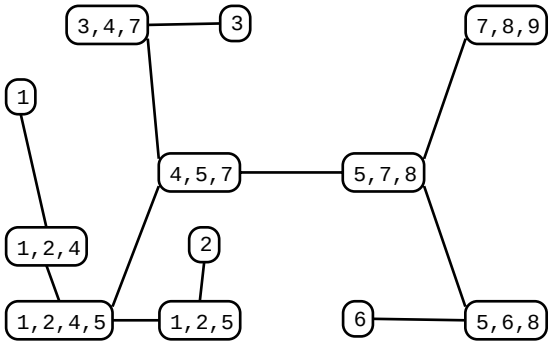


Figure 6a. Augmented Tree of Cliques

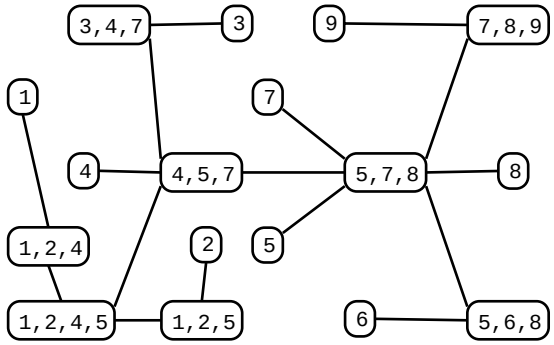


Figure 6b. Doubly Augmented Tree

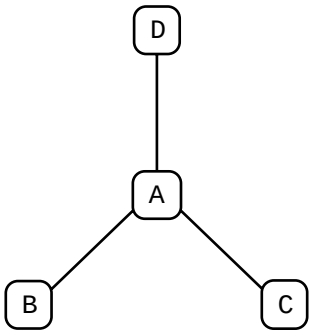
nodes of $\mathcal{T}$. There are three nodes which do not correspond to a potential of the original model: these are $\{1,2,4,5\}$, $\{4,5,7\}$, and $\{5,7,8\}$. The potentials associated with those nodes are the unit potentials over the corresponding space (which has no effect on the calculations).

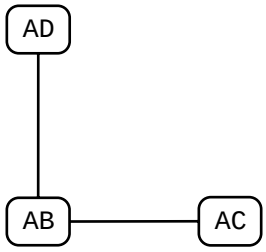
Augmenting has two effects in the tree model. First, it is easier to modify one of the components to represent new information or perform sensitivity analysis. For example, suppose that new information is obtained or hypothesised about Person 1. Under the unaugmented tree of cliques model, the information about Person 1 is stored at Node $\{1,2,4,5\}$; to modify that information, it must be separated from other information stored in that node (like the conditional distribution of 4 given 1 and 2) and a new $\{1,2,4,5\}$ potential computed before the change is propagated by passing new messages. In the augmented tree of cliques, the information about Person 1 is stored in Node 1 with no other information; the new information can be easily substituted for the old and the effects propagated through the system. Second, when tracking influence back through the tree model, identifying nodes of the tree model that produce unusual results is possible, but identifying the component potential (or potentials) in the original that produce the effects may not be possible in the unaugmented tree. For example, if information about Person 2 had a strong impact on our conclusions, we could trace that influence back to Node 2 in the augmented tree of cliques, but only to Node $\{1,2,4,5\}$ in the unaugmented tree. Thus, augmentation facilitates two important tools of model exploration.

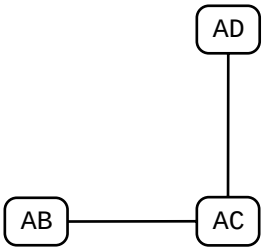
Further augmentation of the tree Model can be useful. In the genetics example, new data about Person 9 may be incorporated into the tree model by the creation of a new node $\{9\}$. Figure 6b shows a *double augmentation* where all single variables are added as nodes to the augmented tree. The single variables are important because: (a) the marginal distributions over single variables are most likely to be of interest and (b) new data (or hypothetical data for sensitivity analysis) are most likely to be entered though the single variables. The new nodes created by double augmentation are assigned unit potentials.

3.3 Building a Junction Tree

In general, more than one tree of cliques can be constructed from a given triangulated graph. For example, consider the graph in Figure 7a which is triangulated. Three possible tree of cliques, displayed as Figures 7b–d, can be constructed from this graph. Note that $\{D,C,B,A\}$ and $\{C,B,D,A\}$ are both perfect elimination orders and hence equivalent. Suppose the elimination order $\{D,C,B,A\}$ is chosen and the algorithm for constructing a tree of cliques is applied. $\{A,C\}$ and $\{A,D\}$. Working backwards through the elimination order, after processing Vertex C, there will be a tree with two nodes, $\{A,B\}$ and $\{A,C\}$, linked together. When the Vertex D is processed, the new clique node $\{A,D\}$ created can be attached to either $\{A,B\}$ or $\{A,C\}$ resulting with Figure 7b or Figure 7c respectively. Note that the prefectly legitimate tree







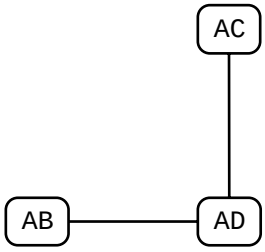


Figure 7a. A Model Graph *Figure 7b. Tree Model 1* *Figure 7c. Tree Model 2* *Figure 7d. Tree Model 3*

of cliques in Figure 7d will never be constructed in this case. Of course, Figure 7d can be constructed if the elimination order $\{C, B, D, A\}$ is used instead.

Not only is the non-uniqueness of the tree cliques inherently non-pleasing from a theoretical point a view, it also raises the question of whether there is a practical difference among these different trees from a computation perspective. In general, the computational cost at each node is proportional to the number of links (ℓ) attached to that node squared (ℓ messages must be calculated each requiring the multiplication of ℓ potentials). Therefore, it is better to have less links attached to nodes which correspond to large outcome spaces. In this sense, if D has a outcome space smaller than that of B and C, then Tree of Cliques in Figure 7d is superior to those in Figure 7b and Figure 7c. However, even Figure 7d can be improved on.

The key idea is to insert intersection nodes into the tree of cliques. These intersection nodes form a smaller space for combining information sources. For example, the *junction tree* model presented in Figure 8c, which is also a Markov Tree, has several advantages over the three trees of cliques presented in Figure 7. First, more combinations take place in the smaller Node $\{A\}$ rather than in one of the larger nodes, such as $\{A, B\}$ or $\{A, C\}$. Second, it is conceptually simpler. From Figure 8c it is easy to see that the three cliques have the common element A. This is much less apparent from any of Figures 7b–d.

Jensen[1988] introduces the term *junction tree*, using a slightly different definition from us. He starts by defining the *junction graph*—a graph produced by starting with the cliques of the graphical model and connecting each pair of cliques with a non-empty intersection through an intermediate *intersection node* (intersection nodes are marked with square boxes). Jensen then defines a junction tree to be a Markov spanning tree of the junction graph. We define a junction tree to be a Markov tree with the additional property that for every pair of neighboring nodes, one is a subset of the other. This definition includes Jensen's junction trees as a special case. For example, consider the graph shown in Figure 7a. Its junction graph is given in Figure 8a. The graph in Figure 8b is a junction tree by Jensen's definition. Both the graphs in Figure 8b and 8c are junction trees by our definition.

It is often the case that a model based on a well chosen junction tree will have a lower associated computational cost than a model based on the tree of cliques. An adaptation of the tree of cliques construction algorithm to produce junction trees which have some optimal properties is presented below.

Building the junction tree requires the introduction of an artificial distinction between *clique nodes* (shown in the figures with round boxes) and *intersection nodes* (shown with square boxes). The algorithm for building the junction tree is similar to the algorithm for building the tree of cliques described in Section 3.1 where we work backwards through the elimination order. At step 1, a tree with a single clique node, $\{i_1\}$ is constructed. In general, at step t, let $\mathbf{N}^* = N(i_t|\mathcal{G}^t)$ and $\mathbf{C}^* = Cl(i_t|\mathcal{G}^t)$. The are four distinct cases :

A.1 If $\mathbf{N}^*$ is an existing intersection node, create a new clique node labeled as $\mathbf{C}^*$ and attach it to $\mathbf{N}^*$.

A.2 If $\mathbf{N}^*$ is not an existing intersection, but is a subset of some existing intersection node. Let $\mathbf{N}$ be the smallest among such intersection nodes (if necessary, break ties in an arbitrary fashion) . Create a new intersection node $\mathbf{N}^*$ and a new clique node $\mathbf{C}^*$. Attach $\mathbf{C}^*$ to $\mathbf{N}^*$ and $\mathbf{N}^*$ to $\mathbf{N}$.

B.1 If $\mathbf{N}^*$ is an existing clique node, simply relabel it $\mathbf{C}^*$.

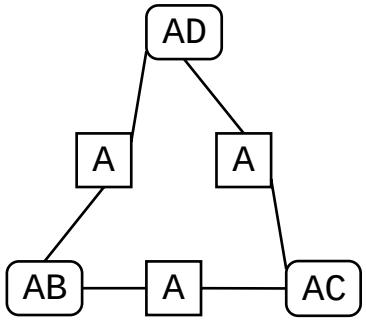


Figure 8a. Junction Graph

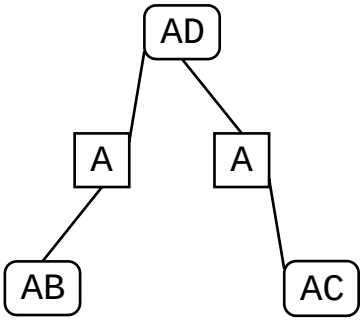


Figure 8b. Jensen Junction Tree

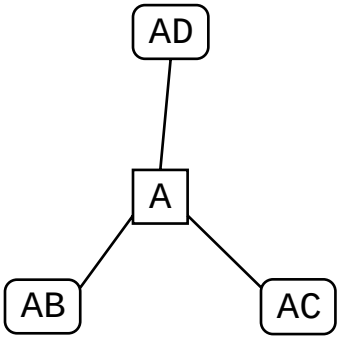


Figure 8c. Our Junction Tree

B.2 If $\mathbf{N}^*$ is the subset of an existing clique node $\mathbf{C}$ (but not of an existing intersection node), create a new intersection node $\mathbf{N}^*$ and a new clique node $\mathbf{C}^*$. Attach $\mathbf{C}^*$ to $\mathbf{N}^*$ and $\mathbf{N}^*$ to $\mathbf{C}$.

In addition , when a new intersection node $\mathbf{N}^*$ is created (either Step A.2 or B.2), carefully check all of the neighbors of the node ($\mathbf{N}$ or $\mathbf{C}$) to which $\mathbf{N}^*$ was attached. If any of the neighbors are subsets of $\mathbf{N}^*$, unlink those neighbors from $\mathbf{N}$ or $\mathbf{C}$ and relink them to $\mathbf{N}^*$. Appendix II contains a LISP implementation of this procedure. As an example, consider the triangulated graph shown in the top left-hand corner of Table 2. Let $\sigma = (A, B, C, D, E, F)$ be the perfect elimination order. Table 2 illustrates both the elimination process and the sequential construction of the Junction tree. Note in particular that when Vertex B is added ($t = 5$), the link joining the intersection node $\{F\}$ to the clique node $\{C, E, F\}$ is redirected to the newly created intersection node $\{E, F\}$.

Before listing the properties of Junction Trees constructed following the above procedure, we introduce the concept of *maximal* intersections. For a clique $\mathbf{C}$ of the triangulated graph $\mathcal{G}_\sigma$, let $\mathcal{I}_\mathbf{C}$ be the collection of *maximal* sets in $\{\mathbf{N}|\mathbf{N} = \mathbf{C} \cap \mathbf{C}'$ where $\mathbf{C}'$ is some other clique of $\mathcal{G}_\sigma\}$. An element of $\mathcal{I}_\mathbf{C}$ is called a maximal intersection of $\mathbf{C}$. Note that a set $\mathbf{N} = \mathbf{C} \cap \mathbf{C}'$ which is a maximal intersection of $\mathbf{C}$ need not be a maximal intersection of $\mathbf{C}'$. For example, in Table 2, the intersection between the two cliques $\{D, F\}$ and $\{A, E, F\}$ is $\{F\}$. The set $\{F\}$ is a maximal intersection of $\{D, F\}$, but not a maximal intersection of $\{A, E, F\}$.

The Junction Tree constructed satisfies the following:

(I) It is a Markov Tree where the clique nodes are the cliques of the triangulated graph $\mathcal{G}_\sigma$ and the intersection nodes are the intersections of some pairs of cliques of the triangulated graph $\mathcal{G}_\sigma$. All the cliques are represented.

(II) The nodes correspond to distinct subsets of variables (vertices of $\mathcal{G}$). Note that this is not a property of Jensen's Junction Trees.

(III) The set of neighbors of a clique node $\mathbf{C}$ is $\mathcal{I}_\mathbf{C}$, which also implies all neighbors of a clique node are intersection nodes. An intersection node may have both types of neighbors. (In general, for any two neighboring nodes, one must be the subset of the other.)

(IV) Among all *connected* Markov Trees which satisfy (I), which includes all Trees of Cliques and Jensen's Junction Trees, the cliques of $\mathcal{G}$ have the minimum number of links in this Junction Tree.

(V) For two Junction trees constructed based on two equivalent elimination orders, the neighbors of the clique nodes are identical. In other words, difference can only occur with respect to links among intersection nodes. From the perspective of computational efficiency, these differences are secondary.

The proofs of the these facts are given in Appendix I.

Similar to the way the tree of cliques was augmented, the junction tree can be augmented. In this case, carefully search all nodes to make sure that each additional node is attached to the smallest node which contains it. In many cases, nodes which the augmentation process would have added to the tree of cliques will already be present as intersection nodes.

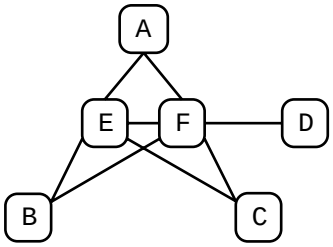
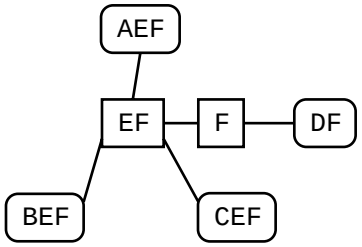
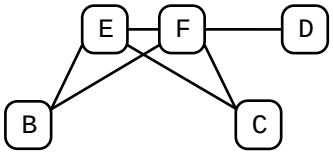
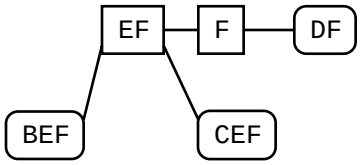
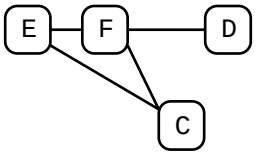
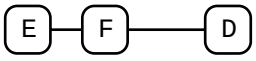
t	$\mathcal{G}^t$	$\mathcal{J}^t$	i_t	$\mathrm{N}(i_t \mid \mathcal{G}^t)$	$\mathrm{Cl}(i_t \mid \mathcal{G}^t)$
6			A	$\{E,F\}$	$\{A,E,F\}$
			$\triangle^6 = \left\{ \begin{smallmatrix} (AEF),(BEF),(CEF), \\ (DF),(EF) \end{smallmatrix} \right\}$		
5			B	$\{E,F\}$	$\{B,E,F\}$
			$\triangle^5 = \{(BEF),(CEF),(DF),(EF)\}$		
4			C	$\{E,F\}$	$\{C,E,F\}$
			$\triangle^4 = \{(CEF),(DF),(EF)\}$		
3			D	$\{F\}$	$\{D,F\}$
			$\triangle^3 = \{(DF),(EF)\}$		
2			E	$\{F\}$	$\{E,F\}$
			$\triangle^2 = \{(EF)\}$		
1			F	$\{\}$	$\{M\}$
			$\triangle^1 = \{(F)\}$		

Table 2. Building a Junction Tree

For each stage t of the reduction of the graph, this table shows the successively smaller versions of the graph $\mathcal{G}^t$ from which the variable i_t is removed, and the junction tree $\mathcal{J}^t$. The table also shows the neighborhood of i_t in $\mathcal{G}^t$, $\mathrm{N}(i_t \mid \mathcal{G}^t)$, which is added as an edge (edges) to the graph $\mathcal{G}^{t-1}$ as part of the elimination process, and the closure of i_t in $\mathcal{G}^t$, $\mathrm{Cl}(i_t \mid \mathcal{G}^t)$, which becomes a clique node in the junction tree. Clique nodes are in round boxes and intersection nodes are in square boxes.

4. Elimination Orders

For a given graph $\mathcal{G}$, two elimination orders σ and σ' are not equivalent if the induced triangulated graphs $\mathcal{G}_\sigma$ and $\mathcal{G}_{\sigma'}$ are not the same. The two different triangulated graphs will not have the same collection of cliques and, as a consequence, the corresponding Markov Trees constructed will also be different. Since the the Markov Tree determines the computational cost—both execution speed and storage requirements—of fusion and propagation, it is important to find an elimination order which leads to efficient computations. The problem of finding an optimal elimination order is very difficult. Many authors have developed heuristics for choosing optimal elimination orders to produce optimal trees of cliques; these are reviewed

below.

As discussed in Section 1, the size of the frame determines both the cost of combining distributions over that frame and storing distributions local to that frame. The nodes of the tree of cliques are the local support spaces for the storage of local information and the combination of distributions in local computation algorithms. To store a potential over a node $\mathbf{N}$ of the tree of cliques, with corresponding outcome space $\Omega_{\mathbf{N}}$, requires a table of $|\Omega_{\mathbf{N}}|$ (the *size of* $\mathbf{N}$) numbers. If the node $\mathbf{N}$ contained n binary variables, the table would have 2^n entries. Combination of two potentials over this space would require the multiplication of $|\Omega_{\mathbf{N}}|$ numbers. It is thus clear that smaller cliques are better than larger cliques, especially as very large cliques could run into physical memory limitations for storage or computation.

Finding a 1-optimal elimination order—order for which the largest clique created is a small as possible—is a NP-complete problem (Arnborg *et al.*[1987]). The related problem of finding a minimum fill-in is also NP-complete (Yannakakis[1981]). Therefore, for graphs $\mathcal{G}$ which have a large number of vertices, rather than concentrating on an algorithm to find an optimal tree of cliques, we focus on heuristics for producing good trees of cliques.

4.1 Simple Heuristics

As a starting point, consider the following procedure, called the *one step look ahead*, for finding elimination orders. (Kong[1986a] introduces this heuristic for finding optimal elimination orders for the peeling algorithm; however, it is equally applicable to elimination orders for producing trees of cliques.) This procedure is also known under many names: *the greedy algorithm,* (Lange and Boehnke[1983]), and the *minimum degree algorithm* (Bertelè and Brioschi[1972]).

One Step Look Ahead, Smallest Clique (one-sc). *At each step of the elimination procedure, look at the graph $\mathcal{G}^t$.*

1) *If there exists a leaf (that is an variable, X_j such that vertices in the set $N(j \mid \mathcal{G}^t)$ are all connected), eliminate that leaf.* (Kong[1986a] has shown that when it can be done, this is always an optimal strategy).
2) *If there is no leaf, eliminate X_j such that the size of $Cl(j \mid \mathcal{G}^t)$ is as small as possible.*
3) *If there exists more than one such node, break ties arbitrarily.*

In particular, at the second stage of the one step look ahead procedure, the criterion for selecting variables is the size of the clique produced when the variable is eliminated—the smaller the better. Therefore, we refer to this procedure as the *one step, smallest clique.* The eliminated vertex has the smallest neighborhood, hence Bertelè and Brioschi[1972] call it the minimum degree algorithm. Variations on this procedure result from using other criteria in the second stage of one step look ahead.

Another measure of cost associated with the elimination of a given variable is the *fill-in number*, or *deficiency*. The fill-in number for a variable, i, to be eliminated from the graph $\mathcal{G}^t$ is the number of pairs $j, k \in N(i|\mathcal{G}^t)$ which are not connected. In other words, it is the number of simple edges which would need to be fill-in if we were to eliminate i from $\mathcal{G}^t$. For situations which involve more than binary variables, the fill-in number is generalized as $\frac{1}{2}\sum log_2|\Omega_{\{j,k\}}|$ where the sum is again taken over the pairs $j, k \in N(i|\mathcal{G}^t)$ which are not connected. Using the fewest fill-ins criteria in Stage 2 creates a variation of the one step look ahead procedure called *one step look ahead, fewest fill-ins* procedure.

One Step Look Ahead, Fewest Fill-ins (one-ff). *As in the one step look ahead, smallest cliques procedure except replace Stage 2 with Stage 2a:*

2a) *Eliminate i (or X_i) which has the smallest fill-in number with respect to $\mathcal{G}^t$.*

Note that a leaf always has a fill-in number of zero so that the first stage is redundant in the fewest fill-ins procedure. Bertelè and Brioschi[1972] discuss this procedure under the name *minimum deficiency*.

In Stage 3 of both procedures ties are broken arbitrarily. Using one of the other candidate criterion to break ties, creates "one and a half step look ahead" procedures. In particular, using Stages (1), (2), (2a)

and then (3) makes the *one step look ahead, smallest cliques, break ties with fewest fill-ins* procedure. Reversing the order of (2) and (2a) makes the *one step look ahead, fewest fill-ins, break ties with smallest cliques* procedure. Bertelè and Brioschi[1972] discuss these heuristics under the names *minimum degree-minimum deficiency* and *minimum deficiency-minimum degree.*

Although these heuristics often work well, they do not produce an optimal elimination order in all cases. The example in Figure 10 taken from Bertelè and Brioschi[1972] illustrates the point. The elimination ordering chosen by all four one-step algorithms (up to symmetry) is: $V_5, V_9, V_1, V_2, V_3, V_4, V_6, V_7, V_8$. This produces two cliques of 5 vertices and one of 7 vertices. On the other hand, the ordering: $V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9$ produces four cliques of 6 vertices.

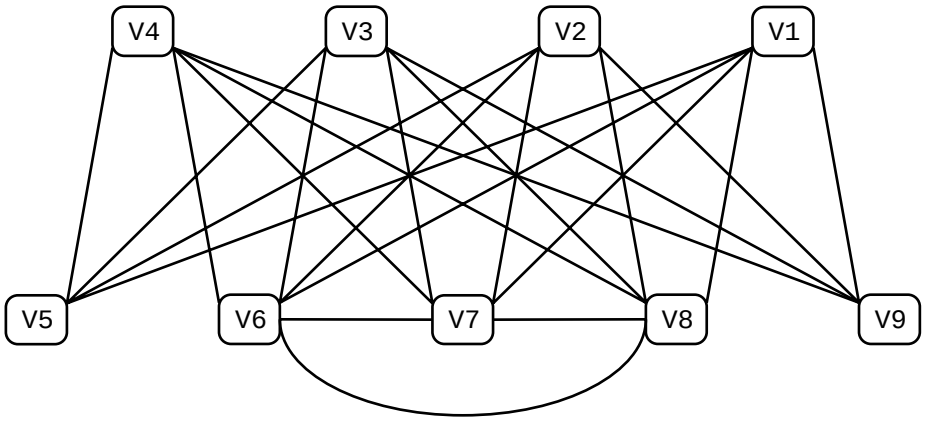


Figure 10. The One-Step heuristics don't work well here (Bertelè and Brioschi[1972])

Even though not always optimal, the one step ahead algorithm and its various variations work optimally, or near optimally in a large number of cases. Unsurprisingly the simple strategies (smallest cliques and fewest fill-ins) take less time than the compound strategies. The fewest fill-ins heuristic is often as effective as the compound heuristics. Intuitively, reducing unnecessary fill-ins leads to smaller cliques at later stages of the elimination process, so this result is not surprising. Also, smaller cliques naturally require fewer fill-ins.

4.2 Initial and Final Theorems

The one-step look ahead algorithm differs slightly from the greedy algorithm in that if it discovers a leaf vertex—a vertex which has all of its neighbors connected—it stops immediately, and eliminates that node first. It can do this because there exists an initial theorem (Kong[1986]) which states deleting a leaf vertex first never results in a larger tree of cliques. To motivate this, look at the graph in Figure 1b. The Vertex 9 is a leaf; all of its neighbors $(7, 8)$ are connected (part of the potential $\psi_{\{7,8,9\}}$). When eliminated, this node will add the clique $\{7, 8, 9\}$ to the tree of cliques, but because $\{7, 8, 9\}$ was already a clique in the graph, it would have appeared in the final tree of cliques anyway. Bertelè and Brioschi[1972] also note this result and call the resulting theorem an *initial theorem*.

Zhang[1988] develops a different initial theorem, one for eliminating *bridges*. A bridge is a vertex which has exactly two neighbors and is contained in exactly two edges; eliminating the bridge effectively shortens the length of a cycle in the graph. Bertelè and Brioschi[1972] discuss several additional initial theorems, although some of them exploit special properties of the problem domain in which they are working (non-serial dynamic programming).

In many cases, it is not necessary to search for an optimal deletion order over the whole graph. In particular, the graph may separate very naturally into two or more pieces, overlapping only at a small separating set (consisting of two or three vertices). Let $\mathbf{X}$ and $\mathbf{Y}$ be two connected sets of vertices such that $\mathbf{X} \cup \mathbf{Y} = \mathcal{V}$, that is $\mathbf{X}$ and $\mathbf{Y}$ together cover the complete graph. Let $\mathbf{\Sigma} = \mathbf{X} \cap \mathbf{Y}$ be their intersection. If $\mathbf{\Sigma}$ is a complete graph (all vertices connected), then $\mathbf{\Sigma}$ is a *complete separator*. The problem of finding and optimal elimination ordering then divides into two smaller problem: finding an optimal elimination ordering for $\mathbf{X} - \mathbf{\Sigma}$ and then finding an optimal elimination ordering for $\mathbf{Y} - \mathbf{\Sigma}$. Eliminating all of the vertices not in $\mathbf{\Sigma}$ according to those orders, leave the vertices in $\mathbf{\Sigma}$ which, because the graph is complete, are all leaf vertices and can be eliminated in arbitrary order. This idea is discussed in Zhang[1988] and Mellouli[1987]. Bertelè and Brioschi[1972] discus this idea under the name *final theorem* because the nodes in $\mathbf{\Sigma}$ can be placed last in the ordering.

4.3 Remarks

Lauritzen and Spiegelhalter[1988] recommend using a procedure called *maximum cardinality search*, from a paper by Tarjan and Yannakakis[1984]. Maximum cardinality search provides a fast mechanism for checking whether or not a model hypergraph is triangulated; however, the tree of cliques produced from the maximum cardinality search elimination ordering has no optimality properties. An earlier paper by Rose, Tarjan and Lueker[1976], discuses a procedure called *lexicographic search*, which produces a "minimal" although not minimum, fill-in. Although both these procedures are designed to minimize fill-ins, not the size of the largest cliques, they are being used to create trees of cliques from model hypergraphs.

Even though the maximum cardinality search and the lexicographic search are faster than the one step look ahead algorithms, the trees they produce are less cost effective, even on the simple example of the Rose *et al.* graph (Almond[1989]). As the computations in the tree model are usually performed many times whereas Markov tree only have to be constructed once, it makes sense to trade computational time in the construction of the tree model for computational efficiency in the fusion and propagation algorithm. Rather than going to a faster search method, we recommend working to improve the speed of the one step heuristics by precalculating statistics such as neighborhoods, clique sizes, and deficiency or by usinging initial and final theorems to make the search space smaller.

Occasionally, there problems will arise for which the one step look ahead will not be adequate. In this case, techniques like simulated annealing (Thomas[1985]) and branch and bound (or A*) search can be used to find an optimal or nearly optimal ordering. Bertelè and Brioschi[1972] explore other heuristics and representations such as eliminating variables in blocks.

5.0 Extensions and Applications

The algorithm described here for building the tree of cliques is used in versions 1.1 and 1.2 of the `BELIEF` package (Almond[1989], see Appendix II) which implements the fusion and propagation algorithm. The planned revision of that package will replace that algorithm with algorithm described here for building a junction tree. Experience with the simple heuristics and algorithms shown here indicates that they should be practical in a wide variety of problems.

This paper chose to focus on the use of the tree of cliques in probabilistic calculations; however, essentially similar techniques can be used for non-serial dynamic programming (Shenoy[1990], Bertelè and Brioschi[1972]). Our original work on this problem (Kong[1986a,b], Almond[1989]) was motivated by the application these techniques to graphical belief function models. For the belief function formulation, finding efficient trees is even more important because worst case costs rise very rapidly with the size of the clique. This common thread suggest a broad class of problems to which these methods are applicable.

Shenoy and Shafer[1990] have produced an axiomatic framework for general information propagation using two operators: combination and projection. The fusion and propagation algorithm applies to any problem in which the combination and projection operations can be suitably distributed. In the case of probability propagation, combination corresponds to multiplication of potentials, projection corresponds to marginalization. For discrete programming, combination corresponds to addition and projection to maximizing over eliminated variables. For belief functions, combination corresponds to Dempster's rule,

projection to minimal extension or marginalization. Shenoy[1991] has shown how a standard decision theory problem mixing probabilities and utilities can be placed in this framework; there is one extra technicality here—time precedence of observations and decision puts constraints of the elimination ordering—but otherwise the same results are used.

As one last example, consider the (relatively) simple fault tree examined in Almond[1989]. System failure depended on the state of 13 subsystems in a complex pattern (including loops in the graphical model). The a failure of any of the 13 subsystems could be caused by any number of basic events (in series). There were a total of 115 basic events. Searching through the full model graph (with over 130 nodes) for optimal deletion orders was very time consuming, even with many leaves in the model. (There are also search costs related to the number of edges in the model). On the other hand, constructing the model was quite simple when domain specific knowledge about the problem was added. First, the 13 subsystem failure events were each separators, separating the graph into the 13 subsystem fault trees and the top level system logic diagram. Furthermore, each subsystem had a very simple graphical structure: a tree. Therefore, an optimal deletion order could determined immediately without using any of the searching algorithms presented in Section 4. A tree of cliques was developed for each subsystem and for the top level system separately. These were then attached at the subsystem failure event nodes. This modular construction of the tree of cliques was far more efficient than the more general routines described here.

As problems get larger and larger, performing all the calculations via fusion and propagation is no longer feasible. These problems will demand variations on the techniques described here, such as combination of peeling with Gibbs sampling (Kong[1991b]). As demonstrated in Kong[1991a], the peeling algorithm can be extended to simulate *joint* outcomes of unobserved variables conditioned on observed data on other variables, which can be part of a Gibbs sampling scheme. The graphical manipulations discussed in this paper apply equally well to such simulations.

References

Almond, Russell G. [1989]. *Fusion and Propagation in Graphical Belief Models: An Implementation and an Example.* Ph.D. dissertation and Harvard University, Department of Statistics Technical Report S-130.

Arnborg, S., Corneil, D. G. and Proskurowski, A.[1987].. "Complexity of finding embeddings in a k-tree." *SIAM Journal of Algebraic and Discrete Methods*, **8**, 277-284.

Bertelè, Umberto and Brioschi, Francesco [1972]. *Nonserial Dynamic Programming*, "Mathematics in Science and Enginnering" series, Volume 91. Academic Press.

Cannings, C., Thompson, E. A., Skolnick, M. H. [1978].. "Probability functions on complex pedigrees." *Advances in Applied Probability*, **10**, 26–61.

Darroch, J. N., Lauritzen, S. L., and Speed, T. P.[1980]. "Markov Fields and Log-Linear Interaction Models for Contingency Tables," *The Annals of Statistics*, **8**, 522-539.

Dempster, Arthur P. and Kong, Augustine [1988]."Uncertain Evidence and Artificial Analysis." *Journal of Statistical Planning and Inference*, **20**, 355-368.

Hilden, J. [1970]. "GENEX—An algebraic approach to pedigree proability calculus." *Clinical Genetics*, **1**, 319–348.

Jensen, Finn V., [1988]. "Junction trees and decomposable hypergraphs." JUDEX Research Report, Aalborg, Denmark.

Kong, Augustine[1986a]. *Multivariate Belief Functions and Graphical Models* Ph.D. thesis, Technical Report S-107, Harvard University, Department of Statistics.

Kong, Augustine[1986b]., "Construction of a Tree of Cliques from a Triangulated Graph." Technical Report Technical Report S-118, Harvard University, Department of Statistics.

Kong, Augustine[1991b]. " Analysis of Pedigree Data Using Methods Combining Peeling and Gibbs Sampling." to appear in the *Proceedings of the 23rd Symposium on the Interface between Computer Science and Statistics.*

Lauritzen, David J. and Spiegelhalter, Steffen L. [1988]. "Fast Manipulation of Probabilities with Local Representations–With Applications to Expert Systems (with discussion)." *Journal of the Royal Statistical Society, Series B,*, **50**, 205-247.

Lange, K. and Boehnke, Michael [1983]. "Extensions to Pedigree Analysis. V. Optimal Calculation of Mendelian Likelihoods ," *Hum. Hered.*, **33**, 291-301.

Mellouli, K. [1987]. *On the propagation of beliefs in networks using the Dempster-Shafer theory of evidence*, Ph.D. dissertation, School of Business, University of Kansas, Lawrence, KS.

Pearl, Judea [1988]. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, California.

Rose, Donald J., Tarjan, Robert E. and Lueker, George S.[1976]. "Algorithmic Aspects of Vertex Elimination on Graphs." *Siam J. Comput.*, **5**, 266-283.

Schacter, R. D.[1986]. "Evaluating Influence Diagrams." *Operations Research*, **34**, 871-82.

Shafer, Glenn and Pearl, Judea[1990]. *Readings in Uncertain Reasoning* Morgan Kaufmann Publishers, Inc. San mateo, California.

Shafer, Glenn, Shenoy, Prakash P. and Mellouli, Khaled[1986]. "Propagating Belief Functions in Qualitative Markov Trees." *International Journal of Approximalte Reasoning*, **1**, 349–400.

Shenoy, Prakash P. and Shafer, Glenn [1990]. "Axioms for Probability and Belief-Function Propagation." in *Uncertainty in Artificial Intelligence*, **4**, 169-198. Reprinted in Shafer and Pearl[1990].

Shenoy, Prakash P.[1990]. "Valuation-Based Systems for Discrete Optimization." *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, Boston, MA, pp. 334-343.

Shenoy, Prakash P.[1991]. "A Fusion Algorithm for Solving Bayesian Decision Problems," in *Uncertainty in Artificial Intelligence, Proceedings of the Seventh Conference*, pp 361-369.

Tarjan, Robert E. and Yannakakis, Mihalis [1984]."Simple Linear-Time Algorithms to test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs." *Siam J. Comput.*, **13**, 566-579.

Thoma, H. Mathis [1989]. *Factorization of Belief Functions.* Ph.D. Thesis. Harvard University, Department of Statistics.

Thomas, Alan[1985]. *Data Structures, Methods of Approximation and Optimal Computation, for Pedigree Analysis.* Ph.D. Thesis, Cambridge University.

Yannakakis, Mihalis [1981]. "Computing the Minimum Fill-In is NP-Complete." *Siam J. Alg. Disc. Meth.*, **2**, 77-79.

Zhang, L.[1988]. "Studies on finding hypertree covers of hypergraphs," Working Paper No. 198, School of Business, University of Kansas, Lawrence, KS.

Appendix I. Proofs of Technical Lemmas

Lemma 1: The cliques of the induced triangulated graph $\mathcal{G}_\sigma$ are the maximal sets among the closures $Cl(i_t|\mathcal{G}^t), t = n, \ldots, 1$. *Because applying the elimination order σ to $\mathcal{G}$ and $\mathcal{G}_\sigma$ results in the same tree, we can consider the elimination process with respect to $\mathcal{G}_\sigma$. First, recall that σ is a perfect elimination order with respect to $\mathcal{G}_\sigma$. As a consequence, $Cl(i_t|\mathcal{G}^t_\sigma)$ must be the subset of some clique of $\mathcal{G}^t_\sigma$, because if not, some edges will have to be added. Again, since no new edges are added in the elimination process, any clique of $\mathcal{G}^t_\sigma$ is the subset of some clique of $\mathcal{G}_\sigma$, which implies $Cl(i_t|\mathcal{G}^t_\sigma)$ is the subset of some clique of $\mathcal{G}_\sigma$. What remains to be shown is that each clique C of $\mathcal{G}_\sigma$ is equal to $Cl(i_t|\mathcal{G}^t_\sigma)$ for some i_t. Among the vertices in C, let i_l be the first one to be eliminated. Obviously, $Cl(i_t|\mathcal{G}^t_\sigma) \supset C$. The fact that the elimination order is perfect implies that $Cl(i_t|\mathcal{G}^t_\sigma)$ is exactly C.* $\vdash$

In Section 3.1, a procedure for constructing a sequence of trees $\mathcal{T}^t, t = 1, \ldots, n$, is given. The following three lemmas demonstrate that the procedure always work and the final tree $\mathcal{T} = \mathcal{T}^t$ has the desired properties.

Lemma 2a: There exists a node of $\mathcal{T}^{t-1}$ which contains $\mathrm{N}(i_t|\mathcal{G}^t)$ for $t \geq 2$. *The result obviously holds for $t = 2$ Assume that it holds for $t-1$ and prove that it holds for t. Note that, because of the construction process, for each $k < t$, $Cl(i_k|\mathcal{G}^k)$ must be the subset of at least one of the nodes in $\mathcal{T}^{t-1}$. Among the vertices in $\mathrm{N}(i_t|\mathcal{G}^t)$, let i_l be the first one to be eliminated. Since all the vertices in $\mathrm{N}(i_t|\mathcal{G}^t)$ are mutual neighbors after the elimination of i_t, $Cl(i_l|\mathcal{G}^l) \supset \mathrm{N}(i_t|\mathcal{G}^t)$. Since $Cl(i_l|\mathcal{G}^l)$ has to be a subset of at least one of the nodes of $\mathcal{T}^{t-1}$, so does $\mathrm{N}(i_t|\mathcal{G}^t)$.* $\vdash$

Lemma 2b: There is a one to one correspondence between the nodes of $\mathcal{T} = \mathcal{T}^n$ and the cliques of the induced triangulated graph $\mathcal{G}_\sigma$. *Note from the construction process that the nodes of $\mathcal{T}^n$ are a subset of the collection of closures $Cl(i_t|\mathcal{G}^t), t = 1, \ldots n$. A particular closure $Cl(i_l|\mathcal{G}^l)$ will definitely be a node of the final tree $\mathcal{T}^n$ if it is not the subset of another closure. It follows from Lemma 1 that the nodes of $\mathcal{T}^n$ include all the cliques of $\mathcal{G}_\sigma$. To show that the nodes are all cliques, we only have to show that none of the nodes of $\mathcal{T}^n$ is the subset of another node. This can be proved by induction. It is obviously true for $\mathcal{T}^1$. Assume that it is true for $\mathcal{T}^{t-1}$. The new node $Cl(i_t|\mathcal{G}^t)$ created at step t is the only node of $\mathcal{T}^t$ that contains i_t and hence cannot be a subset of another node. If the new node is created by relabelling a node $\mathrm{N}(i_t|\mathcal{G}^t)$, $\mathrm{N}(i_t|\mathcal{G}^t)$ cannot contain another node (induction hypothesis). Otherwise, $\mathrm{N}(i_t|\mathcal{G}^t)$ is a proper subset of another node $\mathbf{N}$. Any other node which is a subset of $Cl(i_t|\mathcal{G}^t)$ must also be a subset of $\mathrm{N}(i_t|\mathcal{G}^t)$ and hence $\mathbf{N}$ which violates the induction hypothesis.* $\vdash$

Lemma 2c: $\mathcal{T} = \mathcal{T}^n$ is a Markov Tree. *The proof is again an induction on the construction process. The tree $\mathcal{T}^1$ obviously satisfies the separation property. Now, assume that $\mathcal{T}^{t-1}$ satisfies the separation property. If the new node of $\mathcal{T}^t$ is created by relabeling a node $\mathrm{N}(i_t|\mathcal{G}^t)$ as $Cl(i_t|\mathcal{G}^t)$, then $\mathcal{T}^t$ will continue to be a Markov Tree as $Cl(i_t|\mathcal{G}^t)$ is the only node containing i_t and so will have no effect on the separation property. Otherwise, $Cl(i_t|\mathcal{G}^t)$ is created from scratch and is attached to a node $\mathbf{N}$ of $\mathcal{T}^{t-1}$ which contains $\mathrm{N}(i_t|\mathcal{G}^t)$. Note that $\mathbf{N}$ is the only neighbor of $Cl(i_t|\mathcal{G}^t)$ in $\mathcal{T}^t$. The path between any two nodes of $\mathcal{T}^{t-1}$ is not affected by the introduction of the new node. Hence, if the separation property is violated by $\mathcal{T}^t$, it will have to be that there is a node $\mathbf{N}'$ such that the path between $Cl(i_t|\mathcal{G}^t)$ and $\mathbf{N}'$ includes another node $\mathbf{N}''$ such that $Cl(i_t|\mathcal{G}^t) \cap \mathbf{N}'$ is not a subset of $\mathbf{N}''$. This implies that $\mathbf{N} \cap \mathbf{N}'$ is not a subset of $\mathbf{N}''$ since $Cl(i_t|\mathcal{G}^t)$ is the only node containing i_t and $\mathrm{N}(i_t|\mathcal{G}^t)$ is a subset of $\mathbf{N}$. Because $\mathbf{N}$ is the sole neighbor of $Cl(i_t|\mathcal{G}^t)$, $\mathbf{N}''$ must also be in the path between $\mathbf{N}$ and $\mathbf{N}'$. This means the separation property is violated by $\mathcal{T}^{t-1}$ which contradicts our assumption.* $\vdash$

Lemma 3: Existence of τ. *Let $\mathcal{G}$ be the 2-section of $\triangle$ and let $\mathcal{G}_\sigma$ be the induced triangulated graph. Let I be an element of $\triangle$. I is a completely connected subset in $\mathcal{G}$ and so has to be a completely connected in $\mathcal{G}_\sigma$. As a consequence, I has to be a a subset of some clique of $\mathcal{G}_\sigma$.* $\vdash$

Junction Trees constructed following the procedure described in Section 3.3 have the properties (I) through (V). The proof of (I) is similar to the proofs of Lemmas 2b and 2c, and will be omitted here. Property (II) is obvious from the construction process. The proofs of (III) and (IV) are given below. (V) is the direct consequence of (I) through (III).

Lemma 4a: The set of neighbors of a clique node $\mathbf{C}$ is the set of maximal intersection nodes, $\mathcal{I}_{\mathbf{C}}$ (Property III). *Let $\mathbf{C}$ be a clique node of the constructed Junction Tree. The fact that all neighbors of $\mathbf{C}$ are intersection nodes follows from the construction process. By induction, each neighbor of $\mathbf{C}$ is the intersection between $\mathbf{C}$ and some other clique(s). Let $\mathbf{N}$ be a maximal intersection and let $\mathbf{C}'$ be a clique such that $\mathbf{C} \cap \mathbf{C}' = \mathbf{N}$. By considering the path between $\mathbf{C}$ and $\mathbf{C}'$, we see that $\mathbf{N}$ must be a neighbor of $\mathbf{C}$ because otherwise the separation property will be violated. This implies that the set of neighbors of $\mathbf{C}$ contains $\mathcal{I}_{\mathbf{C}}$. However, note that none of the neighboring intersection nodes is the subset of another. This is a consequence of the way we sometimes redirect the links after applying either Step A.2 or B.2. So the set of neighbors must be exactly $\mathcal{I}_{\mathbf{C}}$.* $\vdash$

Lemma 4b: (IV) in Section 3.3 . *Consider any connected Markov Tree $\mathcal{T}^*$ which satisfies (I). For a clique $\mathbf{C}$, let $\mathcal{I}_{\mathbf{C}} = \{\mathbf{N}_j | j = 1, \ldots m\}$. Because of Lemma 4a, it is sufficient for us to show that $\mathbf{C}$ must have at least m neighbors in $\mathcal{T}^*$. For $j = 1, \ldots m$, let $\mathbf{C}_j$ be a clique node such that $\mathbf{C} \cap \mathbf{C}_j = \mathbf{N}_j$. For each j, for the path between $\mathbf{C}$ and $\mathbf{C}_j$ to satisfy the separation property, the clique node $\mathbf{C}$ must have a neighboring node $\mathbf{N}_j^* \supset \mathbf{C} \cap \mathbf{C}_j = \mathbf{N}_j$ in $\mathcal{T}^*$. The node $\mathbf{N}_j^*$, whether it is a clique node or an intersection node, must be the subset of at least one clique other than $\mathbf{C}$. We now demonstrate that these $\mathbf{N}_j^*$'s must be all distinct. To show by contradiction, suppose there exists $j \neq k$ such that $\mathbf{N}_j^* = \mathbf{N}_k^* = \mathbf{N}^*$. The clique other than $\mathbf{C}$ which contains $\mathbf{N}^*$ must also contain $\mathbf{N}_j \cup \mathbf{N}_k$. But then the intersection of $\mathbf{C}$ and this other clique must also contain $\mathbf{N}_j \cup \mathbf{N}_k$ which contradicts the assumption that $\mathbf{N}_j$ and $\mathbf{N}_k$ are distinct maximal intersections of $\mathbf{C}$. The fact that the $\mathbf{N}_j^*$'s are all distinct implies $\mathbf{C}$ must have at least m neighbors in $\mathcal{T}^*$.* $\vdash$

Appendix II. Lisp Implementation

Code for the construction algorithm for the tree of cliques and all of the heuristics for producing deletion orderings is provided in the `BELIEF` package, which is available via anonymous ftp from `stat.washington.edu`. The code for the junction tree construction procedure has not yet been implemented, but will be released with the next version. The current plan is to incorporate it into a package of `GRAF` functions for manipulating graphs.

The following routines are the implementation of the two key algorithms, the one for producing the tree of cliques and the one for producing the junction tree. They are presented in LISP format. The code assumes a Common Lisp style syntax with several specific extensions. In particular, the basic graph theoretic operations are not standardly available in LISP but are easly implemented. The `GRAF` package provides such an implementation.

This code implements the construction procedure for building the tree of cliques described in Section 3.1.

```
(defun build-tree (elimination-order graph &key (node-size #'length))
  (declare (type List elimination-order) (type Graph graph))
  (if (endp node-list)
      ;; base case
      (make-empty-tree-of-cliques)
    (let* ((vertex (first elimination-order)) ;variable eliminated
           (N* (neighborhood vertex graph))
           (C* (closure vertex graph))
           ;; tail recursively build rest of tree
           (old-tree (build-tree (rest elimination-order)
                                 (eliminate vertex graph))))
      ;; add current closure to tree
      (if (null (nodes old-tree))
          ;;no nodes yet
          (add-node! C* old-tree)
       ;; some nodes already in tree
       (if (member N* (graph-nodes old-tree) :test #'equal-set)
           ;; 4A found neighborhood of vertex, rename it as closure
           (rename-node! N* C* old-tree)
         ;; ELSE 4B add closure
         (let ((new-tree (add-node! C* old-tree)))
           ;; and connect to a node containing neighborhood
           (add-edge! (list C* (find N* (graph-nodes old-tree)
                                        :test #'subsetp))
                      new-tree)))))))
```

The following procedure builds a junction tree.

```
(defun build-junction-tree (elimination-order graph)
  (declare (type List elimination-order) (type Graph graph))
  (if (endp node-list)
      ;; base case
      (make-empty-junction-tree)
    (let* ((vertex (first elimination-order)) ;variable eliminated
           (N* (neighborhood vertex graph))
           (C* (closure vertex graph))
           ;; tail recursively build rest of tree
           (old-tree (build-junction-tree (rest elimination-order)
                                          (eliminate vertex graph))))
      ;;; add current closure to tree
      (if (null (nodes old-tree))
          ;;no nodes yet
          (add-clique-node! C* old-tree)
       ;; some nodes already in tree
       (let ((candidate-inter-nodes
                (select N* (inter-nodes old-tree) :test #'subsetp)))
         (if candidate-inter-nodes
             ;; 4A. found in intersection nodes
             (let*  ((small-size (reduce #'min candidate-inter-nodes
                                         :key node-size))
                     (N-node (find small-size candidate-inter-nodes
                                   :key node-size)))
               (if (equal-set N-node N*)
                   ;; 4A.1 Add clique to tree attach to N-node
                   (let ((new-tree (add-clique-node C* old-tree)))
                     (add-edge (list C* N-node) new-tree))
                 ;; 4A.2 Place N* between between C* and N-node
                 (let ((affected-nodes
                        (select N* (inter-neighborhood N-node old-tree)
                                :test #'supsetp))
                       (new-tree (add-clique-node C* old-tree)))
                   (setq new-tree (add-inter-node N* new-tree))
                   (setq new-tree (add-edge (list C* N*) new-tree))
                   (move-edges N-node N* affected-nodes new-tree))))
           ;; 4B N* is the subset of a clique node
           (let ((C-node (find N* (clique-nodes old-tree)) :test #'subsetp))
             (if (equal-set C-node N*)
                 ;;4B.1
                 (rename-node C-node C* old-tree)
               ;;4B.2
                 (let ((affected-nodes
                        (select N* (inter-neighborhood C-node old-tree)
                                :test #'supsetp))
                       (new-tree (add-clique-node C* old-tree)))
                   (setq new-tree (add-inter-node N* new-tree))
                   (setq new-tree (add-edge (list C* N*) new-tree))
                   (setq new-tree (add-edge (list C-node N*) new-tree))
                   (move-edges C-node N* affected-nodes new-tree))
                 ))))))))
```