LOCAL COMPUTATION IN HYPERTREES

by

Glenn Shafer and Prakash P. Shenoy

August 1988
Revised June 1991

School of Business
University of Kansas
Summerfield Hall
Lawrence, KS 66045-2003 USA

Table of Contents

Acknowledgments

Research for this monograph has been supported by NSF grants IST–8610293, IRI–8902444, and grants from the Peat Marwick Foundation's Research Opportunities in Auditing program. The authors have profited from conversations and correspondence with Chien-Chung Chan, Arthur P. Dempster, Yen-Teh Hsia, Augustine Kong, Khaled Mellouli, Judea Pearl, Debra Zarley and Lianwen Zhang.

CHAPTER ONE

Some Ideas from Graph Theory

Most of the ideas reviewed here have been studied extensively in the literature of graph theory (see Berge [1973], Golumbic [1980], and Maier [1983]). A number of the terms we use are new, however - among them, *hypertree*, *branch*, *twig, bud*, and *Markov tree*. As we have already explained in the introduction, a *hypertree* is what other authors have called an acyclic or decomposable hypergraph. A *Markov tree* is what authors in database theory have called a join tree (see Beeri et al. [1983] or Maier [1983]). We have borrowed the term *Markov tree* from probability theory, where it means a tree of variables in which separation implies probabilistic conditional independence given the separating variables. We first used the term in a non-probabilistic context in Shenoy and Shafer [1986], where we justified it in terms of a concept of qualitative independence analogous to probabilistic independence.

As we shall see, hypertrees are closely related to Markov trees. The vertices of a Markov tree are always hyperedges of a hypertree, and the hyperedges of a hypertree can always be arranged in a Markov tree. The main novelty of our exposition is its reliance on this close relationship. By exploiting it, we derive the most important properties of hypertrees from geometrically obvious properties of trees.

We limit our study of hypertrees to an investigation of the properties that we need for this monograph. For a more thorough study of hypertrees, using only slightly different definitions, see Lauritzen, Speed, and Vijayan [1984].

Section 2.1 consists mostly of definitions. We define hypergraphs, twigs and branches in hypergraphs, hypertrees, hypertree construction sequences, branchings, skeletons, and hypertree covers. Section 2.2 reviews the more familiar topic of

trees, with an emphasis on tree construction sequences rather than on the fact that a tree is connected and acyclic. Section 2.3 finally introduces Markov trees, explains how they are related to hypertrees, and exploits the relation. Section 2.4 spells out proofs for some of the displayed propositions.

2.1. Hypergraphs and Hypertrees

We call a finite nonempty set $\mathcal{H}$ of nonempty subsets of a finite set $\mathcal{X}$ a *hypergraph* on $\mathcal{X}$. We call the elements of $\mathcal{H}$ *hyperedges*. We call the elements of $\mathcal{X}$ *vertices*.

Twig and Branch. Suppose t and b are distinct hyperedges in a hypergraph $\mathcal{H}$. We say that b is a *branch* for t if the following conditions are satisfied:

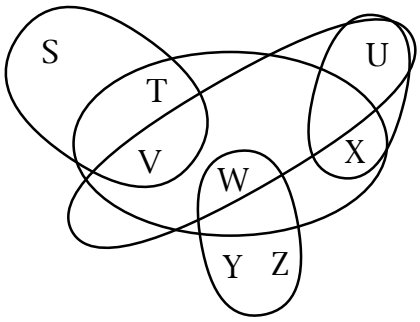
(i) The hyperedges t and b overlap. In symbols, $t \cap b \neq \emptyset$.

(ii) Every vertex in t that is also contained in another hyperedge of $\mathcal{H}$ is also contained in b. In other words, if $X \in t$ and $X \in h$, where $h \in \mathcal{H}$ and $h \neq t$, then $X \in b$.

If a hyperedge t in a hypergraph $\mathcal{H}$ has a branch in $\mathcal{H}$, we say that t is a *twig* in $\mathcal{H}$.

In a hypergraph consisting of a single hyperedge, that hyperedge is not a twig, because there is no other hyperedge that intersects it. In a hypergraph consisting of two intersecting hyperedges, both are twigs. Figure 2.1 shows some examples of twigs in a larger hypergraph. As this figure illustrates, a twig may have more than one branch.

Figure 2.1. Of the five hyperedges in this hypergraph, only two are twigs: {S, T, V} and {W, Y, Z}. The twig {S, T, V} has only one branch, {T, V, W, X}. The twig {W, Y, Z} has two branches, {T, V, W, X} and {U, V, W}.



Hypertrees. We are interested in hypergraphs that can be constructed step-by-step by adding twigs. We call such hypergraphs *hypertrees*. More precisely, we call a hypergraph a hypertree if there is an ordering of its hyperedges, say $h_1h_2...h_n$, such that h_k is a twig in the hypergraph $\{h_1, h_2, ..., h_k\}$ whenever $2 \le k \le n$. We call any such ordering of the hyperedges a *hypertree construction sequence* for the hypertree. We call the first hyperedge in a hypertree construction sequence the *root* of the hypertree construction sequence.

Notice that when we construct a hypertree following a hypertree construction sequence, we have a hypertree at every step along the way; if $h_1h_2...h_n$ is a hypertree construction sequence for $\mathcal{H}$, and $1 \le k < n$, then the subset $\{h_1, h_2, ..., h_k\}$ of $\mathcal{H}$ is also a hypertree, and $h_1h_2...h_k$ is a hypertree construction sequence for it.

A hypergraph consisting of a single hyperedge, say $\mathcal{H}=\{h\}$, qualifies as a hypertree; in this case there is only one hypertree construction sequence, the "sequence" consisting only of h. A hypergraph consisting of two hyperedges, say $\mathcal{H}=\{h, h'\}$, is a hypertree if and only if $h \cap h' \neq \varnothing$; in this case, there are two hypertree construction sequences, hh' and h'h. A hypergraph consisting of three hyperedges is a hypertree if and only if one of the hyperedges has a nonempty intersection with both the others and contains their intersection, if any. Figure 2.2

illustrates some of the possibilities in the case of three hyperedges. Figure 2.3 shows a larger hypertree.

Figure 2.2. Some hypergraphs on {W, X, Y, Z}. All these hypergraphs have exactly three hyperedges. The hypergraph $\mathcal{H}_1$ is a hypertree, and all six orderings of its three hyperedges are hypertree construction sequences. The hypergraph $\mathcal{H}_2$ is a hypertree, but it has only four hypertree construction sequences: {W, X}{X, Y}{Y, Z}, {X, Y}{W, X}{Y, Z}, {X, Y}{Y, Z}{W, X}, and {Y, Z}{X, Y}{W, X}. The hypergraph $\mathcal{H}_3$ is not a hypertree.

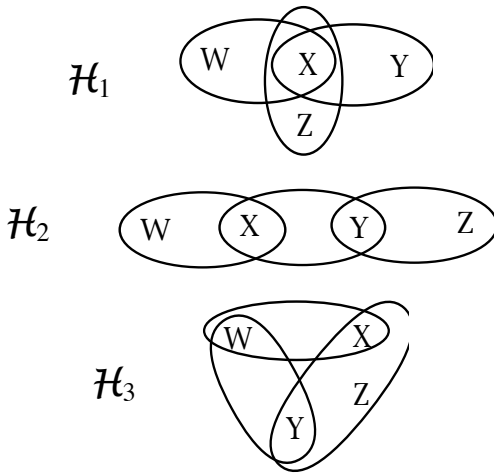
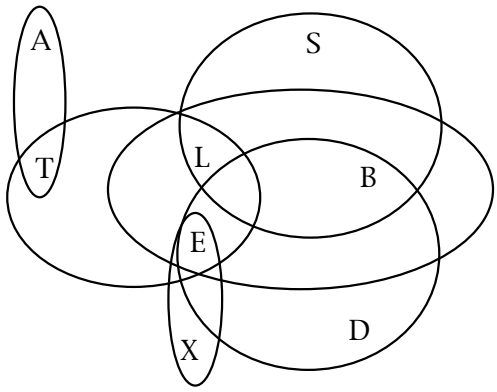
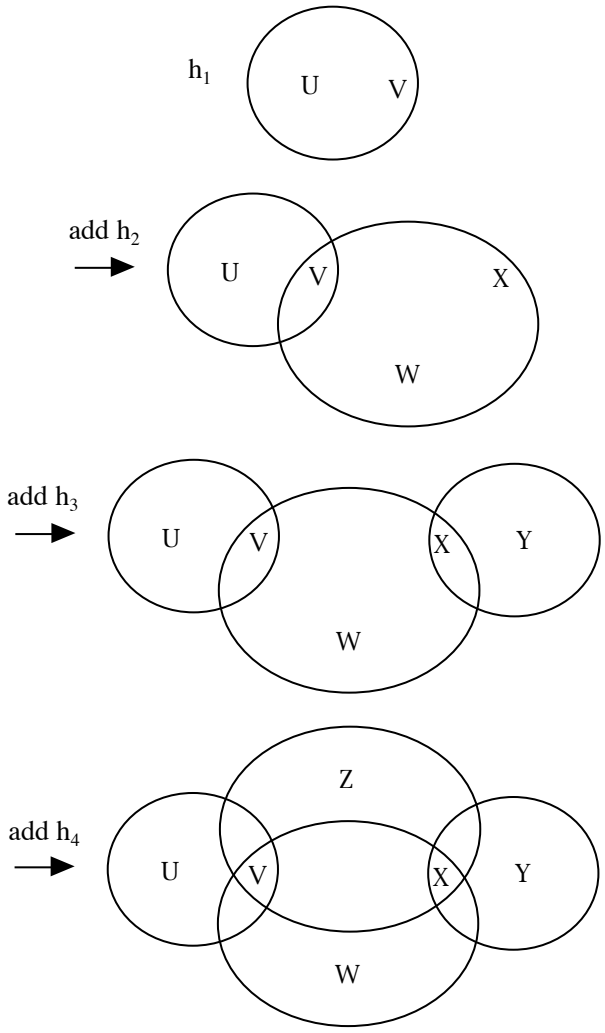


Figure 2.3. A hypertree on {A, T, S, L, B, E, D, X}. The sequence {A, T}{T, L, E}{L, B, E}{S, L, B}{B, E, D}{E, X} and the sequence {L, B, E}{T, L, E}{B, E, D}{S, L, B}{A, T}{E, X} are two of its many hypertree construction sequences. This hypertree is related to an example of local computation studied by Lauritzen and Spiegelhalter [1988].



One aspect of the potential complexity of hypertree construction sequences is the possibility that when we add a twig, we may change whether or not a hyperedge already present is a twig. Figure 2.4 shows an example. The hyperedge h_2 is a twig in $\{h_1, h_2\}$, but it is no longer a twig after the twig h_3 is added. Adding yet another twig, h_4, makes h_2 a twig again.

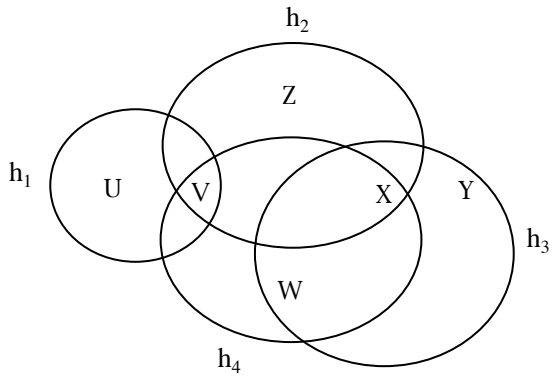
Figure 2.4. The hypergraph $\{h_1, h_2, h_3, h_4\}$ is a hypertree, and $h_1h_2h_3h_4$ is a hypertree construction sequence for it. The hyperedge h_2 is a twig in $\{h_1, h_2\}$ and in $\{h_1, h_2, h_3, h_4\}$ but not in $\{h_1, h_2, h_3\}$.



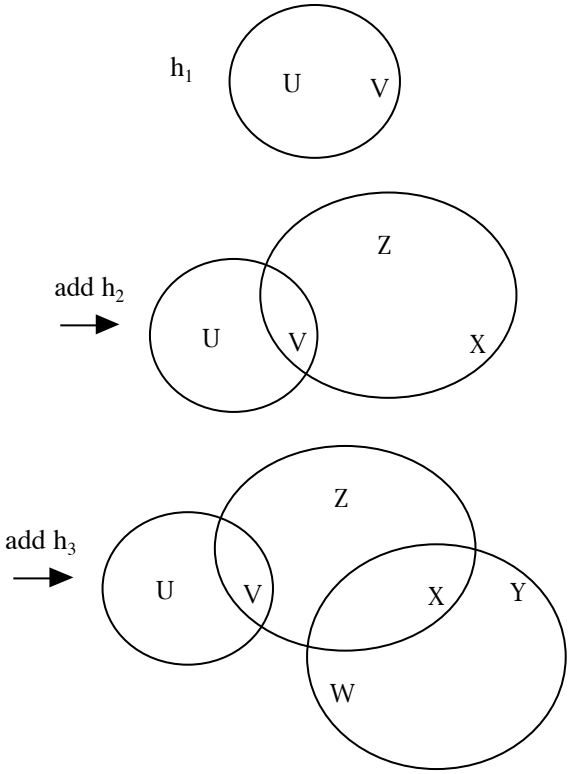
Finding Hypertree Construction Sequences. Given a hypertree, how can we find a hypertree construction sequence for it?

The examples that we have considered so far may suggest that we can always find a hypertree construction sequence simply by starting with an arbitrary hyperedge and arbitrarily adding twigs. As Figure 2.5 illustrates, this does not always work. If we add the wrong twig at one point, we may find that we cannot continue the sequence at a later point.

Figure 2.5. The hypergraph $\{h_1, h_2, h_3, h_4\}$ is a hypertree; $h_1h_2h_4h_3$ is one hypertree construction sequence for it.



On the other hand, $h_1h_2h_3h_4$ is not a hypertree construction sequence. We can get as far as h_3; $h_1h_2h_3$ is a hypertree construction sequence. But we cannot continue by adding h_4, because h_4 is not a twig in $\{h_1, h_2, h_3, h_4\}$.



Fortunately, there is a simple algorithm, called *maximum cardinality search*, which enables us to find hypertree construction sequences easily. Given a hypertree, we select an arbitrary hyperedge to begin the sequence. Then we repeatedly find and add a hyperedge that contains the largest possible number of vertices that are in hyperedges we have already selected. Such a hyperedge, as it turns out, will always be a twig when we add it. If we use maximum cardinality search in the example of Figure 2.5, we might begin with h_1 and then add h_2, but we would not then add h_3, which contains only X from among the vertices {U, V, X, Z} that are in hyperedges we have already selected. Instead, we would add h_4, which contains both X and V. Thus we would obtain the hypertree construction sequence $h_1h_2h_4h_3$. In section 2.3, we prove that maximum cardinality search always works (Proposition 2.9).

Notice that maximum cardinality search also provides a way to check whether a hypergraph is a hypertree. If we are not sure whether $\mathcal{H}$ is a hypertree when we begin, then we check whether each hyperedge the algorithm selects is a twig when we add it. If find one that is not, $\mathcal{H}$ is not a hypertree. Otherwise, $\mathcal{H}$ is a hypertree. For more information on maximum cardinality search, see Tarjan and Yannakakis [1984].

Deletion Sequences. If $h_1h_2...h_n$ is a hypertree construction sequence, then we call the reversed ordering, $h_nh_{n-1}...h_1$, a *hypertree deletion sequence*. An ordering of the hyperedges of a hypertree is a hypertree deletion sequence if and only if when we delete the hyperedges in that order, the hyperedge that we delete at each step (except the last step, when we delete the only remaining hyperedge) is a twig in the hypergraph remaining just before its deletion. This intermediate hypergraph is itself a hypertree, of course. When we tear down a hypertree following a hypertree deletion sequence, as when we construct it following a hypertree construction sequence, the hypergraphs along the way are all hypertrees.

Suppose we arbitrarily select a twig from a hypertree and delete it. Is the hypergraph that is left a hypertree? As we show in section 2.3, the answer is yes (Proposition 2.10). If $\mathcal{H}$ is a hypertree, and t is a twig in $\mathcal{H}$, then $\mathcal{H}$–{t} is a hypertree. It follows from this that we can obtain a hypertree deletion sequence for a hypertree by successively deleting twigs in an arbitrary manner. No matter which twig we delete, there will be another twig to delete next. (Since what remains is a

hypertree, it has some hypertree construction sequence, and the hyperedge that comes last in this sequence will be a twig.)

In a small example, it may be easy to identify twigs, and hence the idea of arbitrarily deleting twigs may provide a practical way to find a hypertree deletion sequence and hence a hypertree construction sequence. In a large hypergraph, however, it may be computationally expensive to find a twig. Hence it is more practical in general to find hypertree construction sequences using maximum cardinality search.

Skeletons. We call a hyperedge in a hypergraph $\mathcal{H}$ *superfluous* if it is contained in another hyperedge of $\mathcal{H}$. A superfluous hyperedge is a twig, and any hyperedge that contains it is a branch for it.

Deletion of a superfluous hyperedge will not affect whether any other hyperedge is superfluous. Indeed, no deletion of any hyperedge can make another hyperedge superfluous if it was not superfluous already. And deleting a hyperedge b that is superfluous itself cannot make a superfluous hyperedge non-superfluous. (If the superfluous hyperedge t is contained in b, but b is superfluous itself, then t will still be contained in b's branch after b is deleted.) So we can delete the superfluous hyperedges from a hypergraph in any order; each one will still be superfluous just before it is deleted, and the hypergraph that remains at the end will have no superfluous hyperedges.

We call a hypergraph that has no superfluous hyperedges *skeletal*. We call the skeletal hypergraph that remains after we delete any superfluous hyperedges from a hypergraph $\mathcal{H}$ the *skeleton* of $\mathcal{H}$. It is easy to see that a hypergraph is a hypertree if and only if its skeleton is a hypertree. (Adding twigs, superfluous or not, to a hypertree produces another hypertree; it merely extends the hypertree construction sequence. As have already mentioned, we will show in section 2.3 that removing twigs from a hypertree also produces another hypertree.)

Branchings. Since each hyperedge we add as we construct a hypertree is a twig when it is added, it has at least one branch in the hypertree at that point. Suppose we choose such a branch, say $\beta(h)$, for each hyperedge h we add. By doing so, we define a mapping β from $\mathcal{H}-\{h_1\}$ to $\mathcal{H}$, where h_1 is the root of the hypertree

construction sequence. We will call this function a *branching* for the hypertree construction sequence.

Since a twig may have more than one branch, a hypertree construction sequence may have more than one branching. In Figure 2.2, for example, the hypertree construction sequence {W, X}{X, Y}{X, Z} for $\mathcal{H}_1$ has two branchings, β_1 and β_2, which agree on {X, Y} but disagree on {X, Z}; $\beta_1(\{X, Y\}) = \beta_2(\{X, Y\}) = \{W, X\}$ but $\beta_1(\{X, Z\}) = \{W, X\}$ and $\beta_2(\{X, Z\}) = \{X, Y\}$.

This example also reveals that a given branching sometimes serves more than one hypertree construction sequence. The branching β_1 qualifies as a branching for {W, X}{X, Z}{X, Y} as well as for {W, X}{X, Y}{X, Z}. These construction sequences have, of course, the same root, {W, X}.

Two hypertree construction sequences that have the same branching always have the same root. The root is easily identifiable from the branching; it is the one hyperedge not assigned to a branch by the branching. So in addition to talking about roots of hypertree construction sequences, we may also talk about roots of branchings. The *root* of a branching β is the root of the hypertree construction sequences for which β qualifies as a branching.

Since a branching can serve more than one hypertree construction sequence for a given hypertree, we will sometimes speak of it as a *branching for the hypertree*, without specifying a particular hypertree construction sequence.

Hypertree Covers. As we explained in chapter 1, local computation requires two things. The joint probability distribution, belief function, or other object with which we are working must factor into functions each involving a small set of variables. And these sets of variables must form a hypertree.

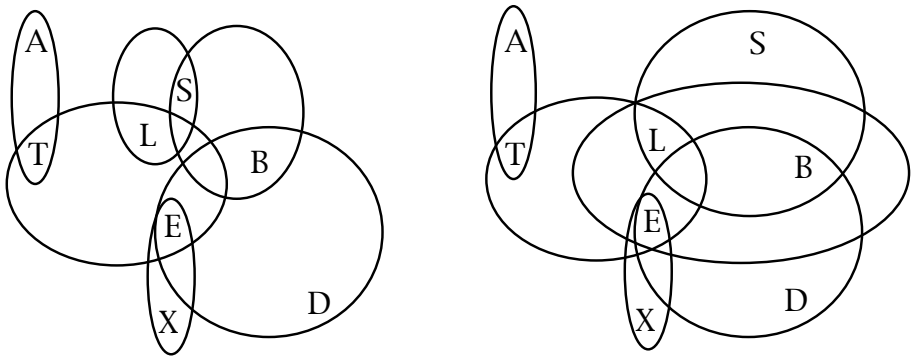
If the sets of variables form instead a hypergraph that is not a hypertree, then we must enlarge it until it is a hypertree. We can talk about this enlargement in two different ways. We can say we are adding larger hyperedges, keeping the hyperedges already there. Or, alternatively, we can say we are replacing the hyperedges already there with larger hyperedges. The choice between these two ways of talking does not matter much, because the presence of superfluous twigs (hyperedges contained in other hyperedges) does not affect whether a hypergraph is a hypertree, and because the computational cost of the procedures we will be

describing depends primarily on the size of the largest hyperedges, not on the number of the smaller hyperedges [Kong 1986].

We will say that a hypergraph $\mathcal{H}$ is *covered* by a hypergraph $\mathcal{H}^*$ if for every h in $\mathcal{H}$ there is an element h* of $\mathcal{H}^*$ such that $h \subseteq h^*$. A hypergraph is covered by any hypergraph that contains it. On the other hand, if $\mathcal{H}$ is covered by $\mathcal{H}^*$, then it is also covered by the skeleton of $\mathcal{H}^*$.

We will say that $\mathcal{H}^*$ is a *hypertree cover* for $\mathcal{H}$ if $\mathcal{H}^*$ is a hypertree and covers $\mathcal{H}$. Figure 2.6 shows a hypergraph that is not a hypertree and a hypertree cover for it.

Figure 2.6. Left: A hypergraph that is not a hypertree. Right: A hypertree cover for it. This is the same hypertree we saw in Figure 2.3.



Finding a hypertree cover is never difficult. The hypertree $\{\mathcal{X}\}$, which consists of the single hyperedge $\mathcal{X}$, is a hypertree cover for any hypergraph on $\mathcal{X}$. Finding a hypertree cover without large hyperedges, or finding a hypertree cover whose largest hyperedge is as small as possible, may be very difficult. In chapter 1, we gave references to authors who have studied heuristics that often produce reasonably good hypertree covers. Here we shall only add that these heuristics generally produce as a by-product a hypertree construction sequence and a branching for the hypertree cover.

2.2. Trees

We now review a more familiar and visually transparent topic—the theory of trees. Our understanding of trees, like our understanding of hypertrees, will be based on the idea of step-by-step construction.

The definitions of graph and tree we give here are conventional, because this suits our purposes in the next section. After reading section 2.1, the reader might expect something less conventional. In section 2.1, instead of defining a hypergraph as a pair $(\mathcal{X}, \mathcal{H})$, where $\mathcal{X}$ consists of vertices and $\mathcal{H}$ consists of subsets of $\mathcal{X}$, we defined it simply as a nonempty set $\mathcal{H}$ of nonempty sets. The reader might expect a similar emphasis on edges in our definition of a graph. We might say that a graph is a set of two-element sets, or equivalently, that it is a hypergraph in which each hyperedge has exactly two elements. But instead we will revert to the more familiar idea that a graph is a pair, a set of vertices and a set of edges.

The displayed propositions in this section are all visually obvious, and so it should not be necessary for the reader to stop to review formal proofs for them. For those readers who would like to do so, however, such proofs are provided in Section 2.4.

Graphs. Formally, then, a *graph* is a pair $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite nonempty set and $\mathcal{E}$ is a set of two-element subsets of $\mathcal{V}$. We call the elements of $\mathcal{V}$ *vertices*, and we call the elements of $\mathcal{E}$ *edges*.

This definition allows a vertex to be isolated—i.e., not contained in any edge. If $\mathcal{E}$ is empty, then all the vertices are isolated.

Suppose $(\mathcal{V}, \mathcal{E})$ is a graph. If {v, v'} is an element of $\mathcal{E}$, then we say that v and v' are *neighbors*, or that they are *connected by an edge*. If $v_1v_2...v_n$ is a sequence of distinct vertices, where n>1, and $\{v_k, v_{k+1}\}\in\mathcal{E}$ for k=1,2, ..., n–1, then we call $v_1v_2...v_n$ a *path*. If v and v' are distinct elements of $\mathcal{V}$, and there is a path $v_1v_2...v_n$ such that $v=v_1$ and $v'=v_n$, then we say that v and v' are *connected by the path*. If every two distinct elements of $\mathcal{V}$ are connected by at least one path, then we say that $(\mathcal{V}, \mathcal{E})$ is *connected*. If $v_1v_2...v_n$ is a path, n>2, and $\{v_n, v_1\}\in\mathcal{E}$, then we call $v_1v_2...v_n$ a *cycle*.

We call a vertex of a graph a *leaf* if it is contained in one and only one edge, and we call the other vertex in that edge the *bud* for the leaf.

Trees. We emphasize the step-by-step construction of trees, just as we emphasized the step-by-step construction of hypertrees. But we think of this construction in terms of vertices, not edges. To construct a hypertree, we added at each step a hyperedge that qualified as a twig. To construct a tree, we will add at each step a vertex that qualifies as a leaf. The following proposition describes this step-by-step construction in several different ways.

> *Proposition 2.1.* Suppose $(\mathcal{V}, \mathcal{E})$ is a graph and $v_1v_2...v_n$ is an ordering of its vertices. Then the following conditions are equivalent.
>
> (i) Each vertex after the first in the sequence $v_1v_2...v_n$ is connected by an edge to exactly one preceding vertex. (It may be connected by an edge to one or more or none of the following vertices.)
>
> (ii) Suppose we draw the graph $(\mathcal{V}, \mathcal{E})$ following the sequence $v_1v_2...v_n$. (This means we start with a dot for v_1, then add a dot for v_2, then add a dot for v_3, and so on; when we add a dot for v, we draw an edge between v and any v' already in the picture such that $\{v, v'\} \in \mathcal{E}$.) Then we will draw exactly one new edge for each new dot. In other words, each dot we draw after the first will be a leaf in the graph just after it is added.
>
> (iii) Suppose we start with $(\mathcal{V}, \mathcal{E})$ and successively delete v_n, v_{n-1}, and so on, until only v_1 remains. (Deleting v_k means that we remove v_k from the set of vertices and remove all edges containing v_k from the set of edges.) Then at each step the vertex we are deleting is a leaf in the graph just before it is deleted.
>
> (iv) For k=2,3, ..., n, the vertex v_k is a leaf in the graph $(\{v_1, v_2, ..., v_k\}, \mathcal{E}_k)$, where $\mathcal{E}_k$ is the subset of $\mathcal{E}$ consisting of those edges that contain only vertices in $\{v_1, v_2, ..., v_k\}$.

We call an ordering $v_1v_2...v_n$ of the elements of $\mathcal{V}$ a *tree construction sequence* for the graph $(\mathcal{V}, \mathcal{E})$ if it satisfies one (and hence all) of the conditions of

Proposition 2.1. We call a graph $(\mathcal{V}, \mathcal{E})$ a *tree* if it satisfies one (and hence all) of the conditions of the following proposition.

> *Proposition 2.2*. Suppose $(\mathcal{V}, \mathcal{E})$ is a graph. Then the following conditions are equivalent.
>
> (i) The elements of $\mathcal{V}$ can be ordered in a tree construction sequence.
>
> (ii) The graph $(\mathcal{V}, \mathcal{E})$ is connected and has no cycles.
>
> (iii) Whenever v and v' are distinct vertices in $(\mathcal{V}, \mathcal{E})$, there is a unique path from v to v'.

If $v_1v_2...v_n$ is a tree construction sequence for $(\mathcal{V}, \mathcal{E})$, then we call the reverse ordering, $v_nv_{n-1}...v_1$, a *tree deletion sequence* for $(\mathcal{V}, \mathcal{E})$.

Notice that a graph with a single vertex, say v, qualifies as a tree; the sequence consisting only of v is both a tree construction sequence and a tree deletion sequence for the tree $(\{v\},\varnothing)$. As we build a tree up following a tree construction sequence, or tear it down following a tree deletion sequence, the graph we have at each step in the process is itself a tree.

We call the first vertex in a tree construction sequence the *root* of the tree construction sequence.

> *Proposition 2.3*. For every vertex v in a tree, there is at least one tree construction sequence with v as its root.

This proposition is geometrically obvious; we simply add edges outwards from v.

Buddings. Since each vertex we add as we construct a tree is a leaf when it is added, it has a bud in the tree at that point. Given a tree construction sequence and a vertex v that is not the root, let $\beta(v)$ denote the bud for v as it is added. This defines a mapping β from $\mathcal{V}-\{v_1\}$ to $\mathcal{V}$, where v_1 is the root. We will call this mapping the *budding* for the tree construction sequence.

The budding for a tree construction sequence is analogous to the branching for a hypertree construction sequence, but there are significant differences. Whereas there may be many branchings for a given hypertree construction

sequence, there is only one budding for a given tree construction sequence. In fact, there is only one budding with a given root:

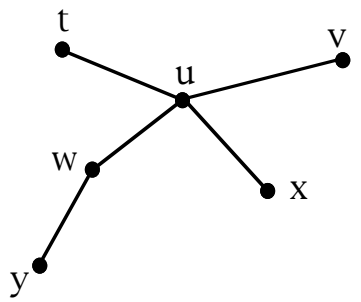
Proposition 2.4. Two tree construction sequences for a given tree have the same budding if and only if they have the same root.

Figure 2.7. If t is the root, then

—t is the bud for u,

—u is the bud for v, w, and x,

—and w is the bud for y.



To see geometrically that this proposition is true, we imagine identifying the budding by moving outwards from the root. Of the two vertices in each edge, the one closer to the root is the bud for the one farther from the root. (See Figure 2.7.)

Suppose we are given the vertices of a tree, but we have not yet decided on the edges. If we begin by selecting one of the vertices to serve as a root, then choosing the edges becomes equivalent to choosing a budding. The preceding proposition said that the edges determine the budding. The following proposition says that the budding determines the edges.

Proposition 2.5. If $(\mathcal{V}, \mathcal{E})$ is a tree, β is a budding for $(\mathcal{V}, \mathcal{E})$, and v_1 is the root for the budding, then $\mathcal{E}=\{\{v, \beta(v)\}|v\in(\mathcal{V}-\{v_1\})\}$.

Once we have chosen a root, the edges and the budding are merely two different forms of the same information.

Leaves in Trees. If a tree construction sequence consists of more than one vertex, then the final vertex in the sequence will be a leaf in the final tree. This establishes a geometrically obvious fact: a tree with more than one vertex has at least one leaf.

It is equally obvious that a tree with more than one vertex has at least two leaves. To prove this formally, consider how the number of leaves can change as we follow a tree construction sequence. When we have only two vertices, both are leaves. At each step after that, the number of leaves is unchanged (this happens if we attach the new leaf to a vertex v that was a leaf, because v will no longer be a leaf) or increases by one (this happens if we attach a new leaf to a vertex that already is not a leaf).

With a little more effort, we can also prove another geometrically obvious fact: when we delete a leaf from a tree, what remains is a tree.

> *Proposition 2.6.* If $(\mathcal{V}, \mathcal{E})$ is a tree, v is a leaf in $(\mathcal{V}, \mathcal{E})$, and v' is the bud for v, then $(\mathcal{V}–\{v\}, \mathcal{E}–\{\{v, v'\}\})$ is also a tree.

We can form a tree deletion sequence for a tree by deleting leaves in an arbitrary way. When we delete a leaf from a tree, a smaller tree will remain, and hence there will be yet another leaf we can delete, until only one vertex remains in the tree. So we can start by deleting any leaf in the initial tree, and at any point in the sequence, we can continue by deleting any leaf in the remaining tree. We need not fear that by choosing the wrong leaf to delete at some point we will land ourselves in a situation where the deletion sequence cannot be continued.

2.3. Markov Trees

As we explained in the introduction, important information about a hypertree can often be presented most clearly by means of a Markov tree. In this section, we will define Markov trees, show how they are related to hypertrees, and use the relation

to prove some of the assertions about hypertrees that we made in section 2.1. In the following chapters, we will use Markov trees to explain visually ideas about local computation in hypertrees.

Definition. We call a tree $(\mathcal{H}, \mathcal{E})$ a *Markov tree* if the following conditions are satisfied:

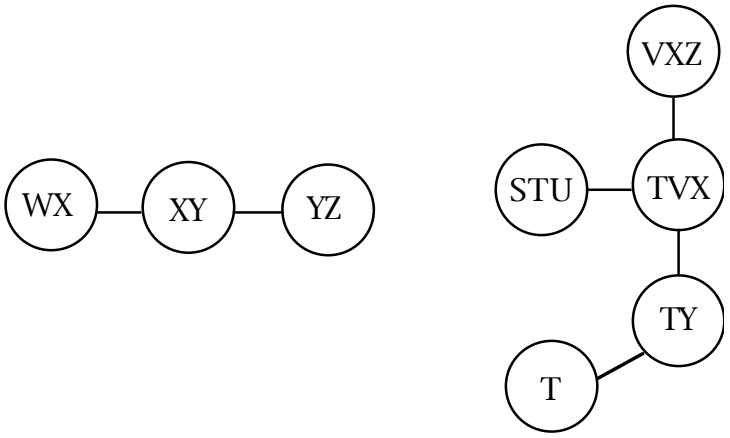
(i) $\mathcal{H}$ is a hypergraph.

(ii) If $\{h, h'\} \in \mathcal{E}$, then $h \cap h' \neq \emptyset$.

(iii) If h and h' are distinct vertices, and X is in both h and h', then X is in every vertex on the path from h to h'.

Condition (iii) can also be expressed by saying that the vertices that contain any particular element X are connected in the tree $(\mathcal{H}, \mathcal{E})$. Figure 2.8 shows two Markov trees.

Figure 2.8. Two Markov trees.



Markov Trees and Hypertrees. Our formal definition of a Markov tree does not state that the vertex set $\mathcal{H}$ is a hypertree, but it implies that it is. This is part of the following proposition.

Proposition 2.7. Suppose $(\mathcal{H}, \mathcal{E})$ is a Markov tree. Then $\mathcal{H}$ is a hypertree. Any tree construction sequence for $(\mathcal{H}, \mathcal{E})$ is a hypertree construction sequence for $\mathcal{H}$. The budding for the tree construction sequence is a branching for the hypertree construction sequence. Any leaf in $(\mathcal{H}, \mathcal{E})$ is a twig in $\mathcal{H}$.

The last statement in this proposition brings out the fact that as we delete leaves from a Markov tree (a visually transparent operation), we are deleting twigs from the hypertree.

If $(\mathcal{H}, \mathcal{E})$ is a Markov tree, then we call it a *Markov tree representative* for the hypertree $\mathcal{H}$. It follows from the following proposition that every hypertree $\mathcal{H}$ has at least one Markov tree representative.

Proposition 2.8. Suppose $\mathcal{H}$ is a hypertree, β is a branching for $\mathcal{H}$, and h_1 is the root of β. Then $(\mathcal{H}, \mathcal{E})$ is a Markov tree, where

$$\mathcal{E}=\{\{h, \beta(h)\}|h\in(\mathcal{H}-\{h_1\})\}. \tag{2.1}$$

If $h_1h_2...h_n$ is a hypertree construction sequence for $\mathcal{H}$, with β as its branching, then it is a tree construction sequence for $(\mathcal{H}, \mathcal{E})$, with β is its budding.

Propositions 2.7 and 2.8 describe the close relationship between hypertrees (with their hypertree construction sequences and branchings) and Markov trees (with their tree construction sequences and buddings). Since the relationship is somewhat complex, it may be useful to describe it in more detail. One way of doing this is to spell out what given information about a hypertree does to determine a Markov tree. We can do this by repeating Proposition 2.8 in various ways:

When we specify a hypertree, we have specified the vertices for a Markov tree.

When we specify a hypertree and a hypertree construction sequence for it, we have specified the vertices and a tree construction sequence for a Markov tree.

> When we specify a hypertree and a branching for it, we have specified a Markov tree and a budding for it (or a Markov tree and a root for it).
>
> When we specify a hypertree, a hypertree construction sequence for it, and a branching for the hypertree construction sequence, we have specified a Markov tree and a tree construction sequence for it.

Another way is to say what given information about a Markov tree tells us about the hypertree formed by its vertices. This is done by the following statements, which follow from Proposition 2.7 together with what we learned about trees in the preceding section.

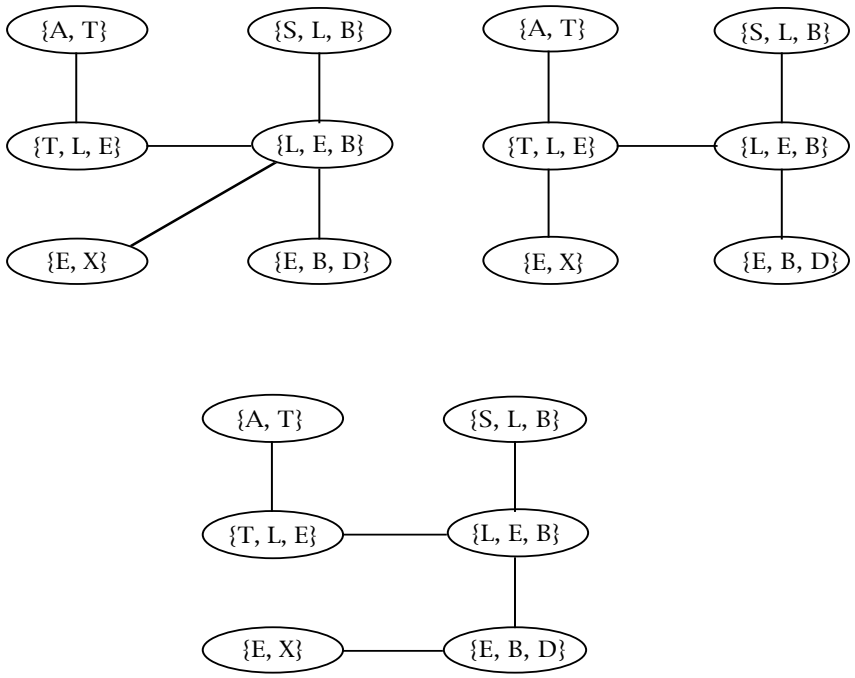
> When we specify a Markov tree, we have specified a hypertree and a branching for each possible root.
>
> When we specify a Markov tree and a root for it, we have specified a hypertree and a branching.
>
> When we specify a Markov tree and tree construction sequence for it, we have specified a hypertree, a tree construction sequence, and a branching.

Notice also that if $\mathcal{H}$ is a hypertree and h_1 is a hyperedge in $\mathcal{H}$, then equation (2.1) represents a one-to-one correspondence between the branchings for $\mathcal{H}$ that have h_1 as their root and the Markov tree representatives for $\mathcal{H}$. (Proposition 2.8 tells us that if β is a branching that has h_1 as its root, then $(\mathcal{H}, \{\{h, \beta(h)\} | h \in (\mathcal{H} - \{h_1\})\})$ is a Markov tree. On the other hand, Proposition 2.5 implies that if $(\mathcal{H}, \{\{h, \beta(h)\} | h \in (\mathcal{H} - \{h_1\})\})$ is a tree, then β is a budding for it, and hence, by Proposition 2.7, a branching for $\mathcal{H}$.)

Given a hypertree, we can choose any hyperedge as root, and then we get all the Markov tree representatives by looking at the different branchings with that

root. Figure 2.9 shows the three Markov tree representatives of the hypertree in Figure 2.3.

Figure 2.9. If we choose {L, E, B} as the root for the hypertree in Figure 2.3, then {L, E, B} must serve as the branch for {T, L, E}, {E, B, D}, and {S, L, B}, and {T, L, E} must serve as the branch for {A, T}. This leaves only {E, X}, which can use {L, E, B}, {T, L, E}, or {E, B, D} as its branch. It follows that the hypertree has exactly three Markov tree representations, which differ only in where the leaf {E, X} is attached.



Which Representation is More Useful? We have developed two different representations for the same information. On the one hand, we have hypertrees, with hypertree construction sequences and branchings. On the other hand, we have Markov trees, with tree construction sequences and buddings. What are the roles of these two representations? When should we use one, and when should we use the other?

As we will see in the following chapters, the methods of local computation that we study in this book involve moving step-by-step backwards and then sometimes forward in a hypertree construction sequence. At each step, we perform computations that involve a hyperedge and its branch. This means that we need to specify the hypertree, a construction sequence, and a branching for it. When all these things are specified, it is really a matter of taste which representation we say we are working with. We can equally well say that we are working with a hypertree construction sequence and a branching or that we are working with a tree construction sequence and a budding for a Markov tree. We can say that we are working step-by-step backwards and then forward in a hypertree construction sequence. Or we can say that we are working inward and then outward in a Markov tree, dealing with the vertices in the order specified by a particular tree deletion sequence as we move inward, and then reversing the order as we move outward.

Strictly speaking, however, it is only if we are using a serial computer that we must completely specify a construction or deletion sequence. If we are working with a parallel computer, in which different processors are made to correspond to the different vertices of the Markov tree, and in which communication links are provided between the vertices that are directly connected by edges in the Markov tree, then we can program the computer to work inwards and then outwards in the tree without specifying completely a sequence in which successive hyperedges are dealt with. In this context, the Markov tree representation will be more appropriate than the hypertree representation.

Our own work has been limited to serial computers, but we find the Markov tree representation very useful for pedagogical purposes. Trees are very accessible to the visual imagination, and we can see many things more clearly if we do not clutter the picture by specifying the particular tree construction sequence. Hence we rely heavily on the Markov tree representation in our explanations of local computation in the following chapters.

Even for the theoretical purposes of this chapter, the Markov tree representation is very useful. As we will see shortly, we can use this representation to give transparent proofs of statements we made about hypertrees in section 2.1.

The main pedagogical limitation of hypertrees is the danger that they will be over-interpreted. It must always be kept in mind that the edges that join the vertices in a

Markov tree have only a computational significance. They do not always have a meaning in terms of the practical problem that led to the computational problem. Typically, that the practical problem determines only a hypertree (or perhaps a hypergraph for which we find a hypertree cover). This hypertree may have many Markov tree representations, and a particular pair of vertices may be linked by an edge in some of these representations and not in others.

Using Markov Trees to Learn about Hypertrees. Here we will use the Markov tree representation to prove two statements about hypertrees that we made without proof in section 2.1.

First consider our assertion that maximum cardinality search always works. If we start with an arbitrary hyperedge h_1 in a hypertree, then find a hyperedge h_2 that has as many elements in common with h_1 as possible, then find a hyperedge h_3 that has as many elements in common with $h_1 \cup h_2$ as possible, and so on, then the resulting sequence $h_1 h_2 ... h_n$ will be a hypertree construction sequence. The following proposition states this more formally.

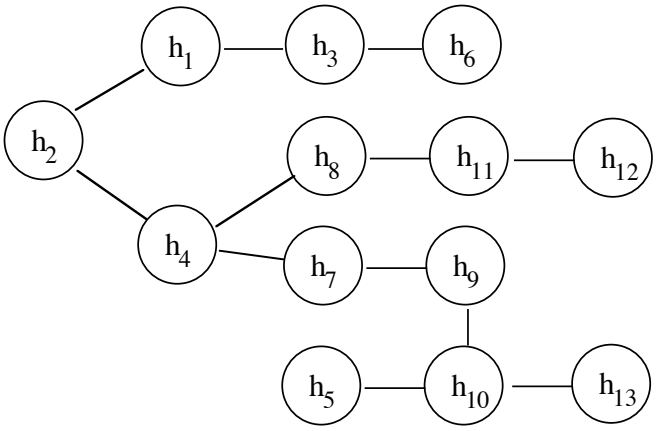
> *Proposition 2.9.* Suppose $\mathcal{H}$ is a hypertree, and $h_1 h_2 ... h_n$ is an ordering of its elements such that
>
> $$|h_i \cap (h_1 \cup h_2 \cup ... \cup h_{i-1})| = \max_{i \le j \le n} |h_j \cap (h_1 \cup h_2 \cup ... \cup h_{i-1})|$$
>
> for i=2, ..., n. Then $h_1 h_2 ... h_n$ is a hypertree construction sequence.

Proof. It suffices to show that there is a Markov tree representative with $h_1 h_2 ... h_n$ as a tree construction sequence. We can prove this by induction. We know that any Markov tree representative has a tree construction sequence that begins with h_1. So we only need to show that if there is a Markov tree representative with a tree construction sequence that begins with $h_1 h_2 ... h_{i-1}$, then there is one with a tree construction sequence that begins with $h_1 h_2 ... h_i$.

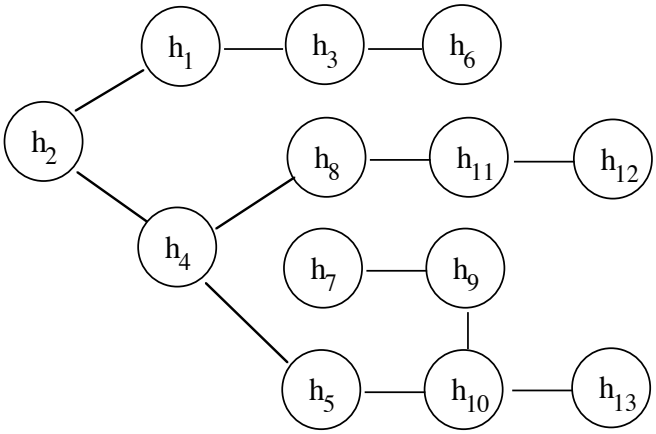
Figure 2.10 shows by example how we can go from a Markov tree with $h_1 h_2$, ..., h_{i-1} as the beginning of a tree construction sequence to one with $h_1 h_2 ... h_i$ as the beginning of a tree construction sequence. We simply replace the first edge on the path from h_j to h_i with an edge between h_j and h_i, where h_j is the vertex in $\{h_1, h_2, ..., h_{i-1}\}$ that is nearest h_i. By the Markov property, every vertex along this path from h_j to h_i contains everything that h_i has in common with $h_1 \cup h_2 \cup ... \cup h_{i-1}$. And

this is all they can have in common with $h_1 \cup h_2 \cup ... \cup h_{i-1}$, since h_i has a maximal intersection with $h_1 \cup h_2 \cup ... \cup h_{i-1}$. It follows that any element X that h_j has in common with the vertex to which it was connected by the edge now removed will also be in all the vertices along the new path to that vertex. Hence the new tree still has the Markov property. **End of Proof.**

Figure 2.10. This Markov tree has a tree construction sequence beginning with $h_1h_2h_3h_4$.



Suppose h_5 has a maximum intersection with $h_1 \cup h_2 \cup h_3 \cup h_4$—i.e., h_5 has at least as many elements in common with $h_1 \cup h_2 \cup h_3 \cup h_4$ as any of the remaining hyperedges do. By the Markov property, all the elements that h_5 has in common with $h_1 \cup h_2 \cup h_3 \cup h_4$ must be in every vertex on the path from h_4 to h_5. Moreover, h_7 cannot have anything else common with h_4; otherwise it would have more in common with $h_1 \cup h_2 \cup h_3 \cup h_4$ than h_5 does. This means that we can replace the link between h_4 and h_7 with a link between h_4 and h_5, without destroying the Markov property:



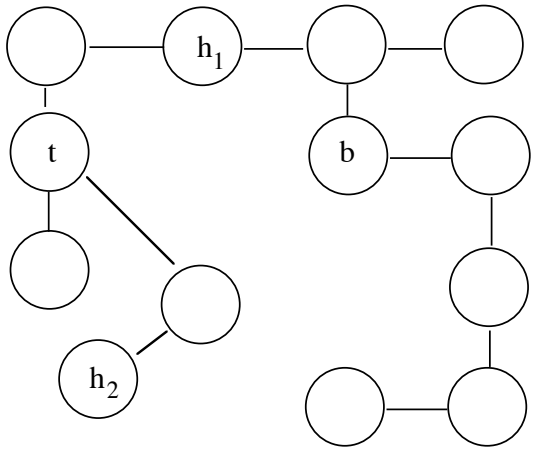
If h_4 and h_7 contains a particular element X, then X will also be in h_5, h_9 and h_{10}, and hence the set of vertices containing it will remain connected.

The preceding proposition says in particular that we can begin a hypertree construction sequence with an arbitrary hyperedge. It is also true, as we stated but did not prove in section 2.1, that we can end a hypertree construction sequence with an arbitrary twig.

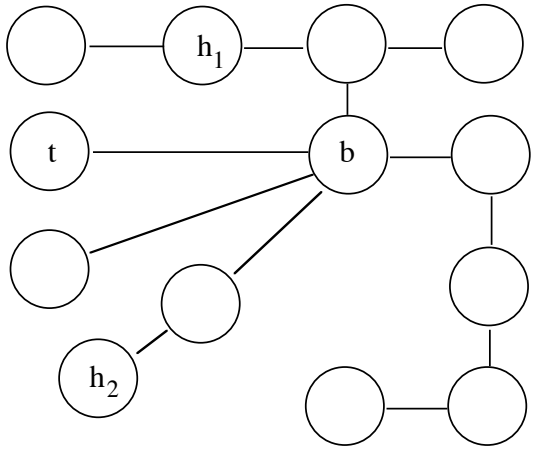
> *Proposition 2.10.* If $\mathcal{H}$ is a hypertree, and t is a twig in $\mathcal{H}$, then there is a hypertree construction sequence for $\mathcal{H}$ that ends with t.

Proof. It suffices to show that there is a Markov tree representative for $\mathcal{H}$ that has t as a leaf. To do this, we consider an arbitrary Markov tree representative for $\mathcal{H}$. Suppose t is not a twig in this Markov tree. Then we construct a new tree by deleting the edge between t and each of its neighbors, putting a new edge from t to b, and putting a new edge from each former neighbor of t to b, except for the single former neighbor that is still connected to b by a path (see Figure 2.11). This new tree has t as a leaf, and it is a Markov tree. To see that it is a Markov tree, consider any two vertices h_1 and h_2 in the tree that have an element X in common. We must show that X is also in any vertex h on the path between h_1 and h_2. This will certainly be true if this path is the same as the path between h_1 and h_2 in the old tree, because the old tree was a Markov tree. Suppose, then, that the path between h_1 and h_2 is different in the new tree, and that h is on the new path but not on the old path. The fact that the new path is different means that the old path went through t, and the fact that h is on the new path but not the old path means that h is either on the old path from h_1 to b or on the old path from h_2 to b. Since the old path went through t, X is in t and hence also in b, which is a branch for t. So X is also in the any vertex on the old path from h_1 to b or on the old path from h_2 to b. **End of Proof.**

Figure 2.11. The vertex t is not a leaf in this Markov tree, but we assume that it is a twig, with branch b, in the hypertree formed by the vertices of the Markov tree.



We remove all the edges between t and other vertices. Then we make the graph a tree again by putting an edge between b and t and an edge between b and each of t's former neighbors, except the one that was originally connected to b by a path that did not go through t.



The new tree has t as a leaf, and it is a Markov tree. To see that it is a Markov tree, suppose that the vertices h_1 and h_2 have an element X in common. They were originally connected by a path that went through t, and every vertex on this path contained X. Now they are connected by a new path. Does every vertex in the new path contain X? Yes. Since X is in t, it is in the branch b. Hence it is in the other vertex in the path from h_1 to b, and this is the only vertex in the new path that was not in the old path.

2.4. Additional Proofs

Proof of Proposition 2.1. The equivalence of the four conditions is obvious; they say the same thing in slightly different words. It is easiest to see first that (i) and (ii) are equivalent, then that (i) and (iii) are equivalent, and then that (iii) and (iv) are equivalent.

Proof of Proposition 2.2. First let us show that (i) implies (ii). Let $v_1v_2...v_n$ be a tree construction sequence for (V, E). Every vertex after v_1 in this sequence is connected by an edge to an earlier vertex, its bud. The bud, if it is not already v_1, is connected in turn to its own bud. Since there are only a finite number of vertices, this path must end eventually at v_1. Thus every vertex is connected by a path to v_1. It follows that any two vertices are connected by a path to each other. Thus (V, E) is connected. It is easy to see also that (V, E) has no cycles. Suppose C is a subset of V. In order for C to be a cycle in (V, E), every vertex in C would have to be connected to two other vertices in C. But since a vertex can only be connected to one earlier vertex in the tree construction sequence $v_1v_2...v_n$, the vertex in C that appears last in this sequence can be connected to only other vertex in C.

It is easy to see that (ii) implies (iii), for being connected means that there is path between any two vertices, and a cycle can be constructed from two distinct paths and vice versa.

We can complete the proof by showing that (iii) implies (i). Suppose there is a unique path connecting any two vertices. Start with an arbitrary vertex and call it v_1. If there is at least one more vertex, then choose one, say v_1', let v_2 be the first vertex after v_1 on the unique path from v_1 to v_1'. The sequence v_1v_2 is a tree construction sequence, and we can extend this sequence step-by-step until it includes all the vertices in V. Indeed, if $v_1v_2...v_k$ is a tree construction sequence, and there is still at least one vertex not in this sequence, then we can choose one

such vertex, say v_k', and let v_{k+1} be the first vertex not already in the sequence that we encounter on the unique path from v_1 to v_k'. It is connected by an edge to just one of the vertices already in the sequence, the one just before it on the path. (Were it also connected by an edge to another, there would be more than one path from it to v_1.) So $v_1v_2...v_kv_{k+1}$ is still a tree construction sequence.

Proof of Proposition 2.3. This is proven in the last paragraph of the proof of the preceding proposition.

Proof of Proposition 2.4. The budding explicitly identifies the root; it is the one vertex that is not assigned a bud by the budding. To see that the root determines the budding, note that for any budding β with root v_1 and any vertex v distinct from v_1, the sequence v, $\beta(v)$, $\beta(\beta(v))$, $\beta(\beta(\beta(v)))$, and so on, is a path from v to v_1. Since the path from v to v_1 is unique, $\beta(v)$ is unique.

Proof of Proposition 2.5. This is obvious if we consider the tree construction sequence for which β is the budding and recall condition (ii) of Proposition 2.1.

Proof of Proposition 2.6. By Proposition 2.3, we can choose a tree construction sequence $v_1v_2...v_n$ for $(\mathcal{V}, \mathcal{E})$ with $v \neq v_1$. Say $v=v_k$. By the definition of tree construction sequence, v is connected by an edge to a vertex earlier in the sequence, say v_j. Since v is a leaf and v' is its bud, we have $v_j=v'$. Moreover, v is not connected by an edge to any vertex later in the sequence. In other words, v_k does not serve as a bud for any vertex later in the sequence. Hence the sequence with v_k removed is still a tree construction sequence—a tree construction sequence for $(\mathcal{V}-\{v\}, \mathcal{E}-\{\{v, v'\}\})$.

Proof of Proposition 2.7. Let $h_1h_2...h_n$ be a tree construction sequence for the Markov tree $(\mathcal{H}, \mathcal{E})$, and let β be its budding. In order to show that $h_1h_2...h_n$ is a hypertree construction sequence, with β as a branching, it suffices to show that $h_i \cap \beta(h_i) \neq \varnothing$ and $(h_1 \cup h_2 \cup ... \cup h_{i-1}) \cap h_i = \beta(h_i) \cap h_i$ for i = 2, ..., n. The condition $h_i \cap \beta(h_i) \neq \varnothing$ follows from condition (i) of the definition of Markov tree. To show that $_i = \beta(h_i) \cap h_i$, suppose that X is contained in both h_i and h_j, where $j < i$. It suffices to show that X is in $\beta(h_i)$. By condition (iii) of the definition of Markov tree, X is also in every node on the path between h_i and h_j. Is $\beta(h_i)$ on this path?

Yes it is. The path h_i, $\beta(h_i)$, $\beta(\beta(h_i))$, $\beta(\beta(\beta(h_i)))$, and so on leads sooner or later back to h_1, as does the path h_j, $\beta(h_j)$, $\beta(\beta(h_j))$, $\beta(\beta(\beta(h_j)))$, and so on; so the path from h_i to h_j follows the first of these paths until it comes to an element in the second path (either h_1 or an earlier common element) and then follows the other path back out to h_j. So $\beta(h_i)$ is on the path between h_i and h_j, and hence X is in $\beta(h_i)$.

Suppose h is a leaf in $(\mathcal{H}, \mathcal{E})$. As we noted in the comments following Proposition 2.6, this implies that we can choose a tree construction sequence for $(\mathcal{H}, \mathcal{E})$ that ends with h. Since this tree construction sequence is a hypertree construction sequence for $\mathcal{H}$, h is a twig in $\mathcal{H}$.

Proof of Proposition 2.8. Since $\{h_2, \beta(h_2)\}$, $\{h_3, \beta(h_3)\}$, ..., $\{h_n, \beta(h_n)\}$ are the edges of the graph $(\mathcal{H}, \mathcal{E})$, and $\beta(h_i)$ is in $\{h_1, h_2, ..., h_{i-1}\}$ for i=2, ...,n, the sequence $h_1h_2...h_n$ is a tree construction sequence for $(\mathcal{H}, \mathcal{E})$, and hence $(\mathcal{H}, \mathcal{E})$, is a tree. Condition (ii) in the definition of Markov tree is satisfied because a twig and its branch always have nonempty intersection. To see that condition (iii) is satisfied, consider two hyperedges h_i and h_j that contain X. We may assume without loss of generality that i<j. By the definition of branch, X must be contained in $h_{\beta(j)}$, which is just before h_j on the path from h_i to h_j. If $\beta(h_j)=h_i$, we are done; otherwise we may apply the same argument to show that X is also contained in the vertex just before h_i or $\beta(h_j)$, whichever comes later in the construction sequence. We may conclude by induction that X is contained in every vertex h_k along the path between h_i and h_j.

CHAPTER THREE

Probability Propagation

In this chapter, we explain local computation for probability distributions. More precisely, we show how computation of marginal probabilities can be facilitated when a joint probability distribution is given in factored form, and the sets of variables involved in the factors form a hypertree.

We postpone for the moment consideration of how probability distributions in factored form might arise. We will return to this question in chapter 6, after we discuss conditional probability.

We begin this chapter by introducing basic definitions and notation for probability distributions, marginalization, and combination. In section 3.1, we introduce a notation for probability distributions and for the more general functions that we call potentials and arrays. In section 3.2, we study the combination of arrays, and in section 3.3, we define marginalization for arrays.

In section 3.4, we show how local computation can be used to marginalize a factorization on a hypergraph to the smaller hypergraph resulting from the deletion of a twig. Though brief and simple, this section is the heart of this chapter. Once we know how to delete a twig, we can reduce a hypertree to a single hyperedge by successively deleting twigs. When we have reduced a factorization on a hypertree to a factorization on a single hyperedge, it is no longer a factorization; it is simply the marginal for the hyperedge.

In section 3.5, we shift our attention from a hypertree to the Markov tree determined by a branching for the hypertree. Using this Markov tree, we describe

more graphically the process of marginalizing to a single hyperedge. Our description is based on the idea that each vertex in the tree is a processor, which can operate on arrays for the variables it represents and then send the result to a neighboring processor. In section 3.6, we generalize this idea to a scheme of simultaneous computation and message passing that produces marginals for all the vertices in the Markov tree. In section 3.7, we give an example of probability propagation. In section 3.8, we give proofs of the displayed propositions.

Our treatment of local computation in this chapter applies to arrays in general, not just to probability distributions. We take this approach not because the greater generality is of practical importance, but rather because it distances us from probabilistic interpretations and allows us to concentrate on purely computational aspects of our problem. In particular, it frees us from the temptation to seek a probabilistic interpretation for every step in the computation.

When considered as schemes for the computation of probabilities, the schemes presented in this chapter are similar to earlier work by Pearl [1986], Shenoy and Shafer [1986] and Lauritzen and Spiegelhalter [1988]. They represent a simplification of all this earlier work, however, for they avoid the need for divisions. In chapter 6, we will delve into the details of the probability case, and we will show how the general schemes presented here relate to the schemes studied by Pearl and by Lauritzen and Spiegelhalter.

3.1. Arrays, Potentials, and Probability Distributions

We use the symbol $\mathcal{W}_X$ for the set of possible values of a variable X, and we call $\mathcal{W}_X$ the *frame* for X. We will be concerned with a finite set $\mathcal{X}$ of variables, and we will assume that all the variables in $\mathcal{X}$ have finite frames.

Given a finite nonempty set h of variables, we let $\mathcal{W}_h$ denote the Cartesian product of $\mathcal{W}_X$ for X in h; $\mathcal{W}_h = \times\{\mathcal{W}_X \mid X \in h\}$. We call $\mathcal{W}_h$ the *frame* for h. We call elements of $\mathcal{W}_h$ *configurations of h*. We use lower-case, bold-faced letters such as **x**, **y**, etc. to denote configurations. If **x** is a configuration of g, **y** is a configuration of h, and $g \cap h = \varnothing$, then (**x**, **y**) is a configuration of $g \cup h$.

We call any real-valued function on $\mathcal{W}_h$ an *array on h*. We call the array a *potential* if the values it assigns to the configurations are non-negative and not all zero. We call a potential a *probability distribution* if the values it assigns to the configurations add to one. Given a potential that is not a probability distribution, we can construct a probability distribution by dividing all the potential's values by their sum.

It is convenient to extend this terminology to the case where the set of variables h is empty. We adopt the convention that the frame for the empty set ∅ consists of a single configuration, and we will use the symbol ♦ to name that configuration; $\mathcal{W}_\varnothing$={♦}. To be consistent with our notation above, we will adopt the convention that if **x** is a configuration for g, then (**x**, ♦) = **x.** Also note that to specify an array A on ∅, we need to specify only a single real number, the value of A(♦). If this real number is positive, the array A is a potential; if it is equal to one, the array A is a probability distribution.

When A is an array on a set of variables h, we will call h the *domain* of A, and we will write domain(A)=h.

Two arrays A and B on h are *proportional* if there is a positive number c such that A(**x**) = cB(**x**) for all **x** in $\mathcal{W}_h$. We will write A ∝ B to indicate that A and B are proportional.

3.2. Combination and Factorization of Arrays

In this section, we learn how to combine arrays, and we learn what it means for an array to factor on a hypergraph.

In order to develop a notation for the combination of arrays, we first need a notation for the projection of configurations from one frame to a smaller frame. Here projection simply means dropping extra coordinates; if (w, x, y, z) is a configuration of {W, X, Y, Z}, for example, then the projection of (w, x, y, z) to {W, X} is simply (w, x), which is a configuration of {W, X}. If g and h are sets of variables, h⊆g, and **x** is a configuration of g, then we will let $\mathbf{x}^{\downarrow h}$ denote the

projection of $\mathbf{x}$ to h. The projection $\mathbf{x}^{\downarrow h}$ is always a configuration of h. If $h=\varnothing$, then of course $\mathbf{x}^{\downarrow h}=\blacklozenge$.

Combination. We combine arrays by pointwise multiplication; if G and H are arrays on g and h respectively, then their *combination* GH is the array on $g\cup h$ given by

$$(GH)(\mathbf{x}) = G(\mathbf{x}^{\downarrow g})H(\mathbf{x}^{\downarrow h}) \tag{3.1}$$

for all $\mathbf{x}\in\mathcal{W}_{g\cup h}$.

Formula (3.1) is merely a careful way of saying that we multiply the two arrays together. It may clarify this to consider a more concrete example. Suppose g={W, X, Y} and h={Y, Z}. Then (3.1) reduces to the statement that (GH)(w, x, y, z) = G(w, x, y)H(y, z).

If G and H are potentials, their combination GH need not be a potential; it is possible that the array GH is such that $(GH)(\mathbf{x}) = 0$ for all $\mathbf{x}$. When we are propagating probabilities, we avoid situations where this happens, because we want to be able to normalize to get a probability distribution. If G and H are potentials on g and h, respectively, and there exists a configuration $\mathbf{x}$ of $g\cup h$ such that

$$G(\mathbf{x}^{\downarrow g})H(\mathbf{x}^{\downarrow h}) > 0,$$

then we will say that G and H are *combinable* and that GH is the combination of G and H. If $G(\mathbf{x}^{\downarrow g})H(\mathbf{x}^{\downarrow h}) = 0$ for all $\mathbf{x}\in\mathcal{W}_{g\cup h}$, then we will say that potentials G and H are *not combinable*.

Vacuous Extension. Suppose $h\subseteq g$, and suppose A is an array on h. Then we let $A^{\uparrow g}$ denote the array on g given by

$$A^{\uparrow g}(\mathbf{x}) = A(\mathbf{x}^{\downarrow h})$$

for all $\mathbf{x}\in\mathcal{W}_g$. We call $A^{\uparrow g}$ the *vacuous extension* of A to g. The idea of vacuous extension does not add anything essential to the ideas of this chapter, but it is useful in the exposition. We can explain formula (3.1), for example, by saying that we get GH by vacuously extending both G and H to $g\cup h$ and then multiplying them.

Suppose $h\subseteq g$, and B is an array on g. We say that B is *carried by h* if there is some array A on h such that $B = A^{\uparrow g}$. This idea will be of use to us in chapter 6.

Factorization. Suppose A is an array on a finite set of variables $\mathcal{X}$, and suppose $\mathcal{H}$ is a hypergraph on $\mathcal{X}$. If A is equal to a combination of arrays on the hyperedges of $\mathcal{H}$, say

$$A = \Pi\{A_h \mid h \in \mathcal{H}\}, \tag{3.2}$$

where A_h is an array on h, then we say that A *factors* on $\mathcal{H}$ into the arrays A_h.

If A is an array on $\mathcal{X}$, and A factors on a hypergraph $\mathcal{H}$ on $\mathcal{X}$, then $\cup\mathcal{H} = \mathcal{X}$. This is implicit in the definition, because the right-hand side of (3.2) is an array on $\cup\mathcal{H}$.

When A does factor on $\mathcal{H}$, the arrays A_h are not unique. We can multiply one of them by a non-zero constant if we compensate by dividing another by the same constant. More generally, if g and h overlap, then we can multiply A_g and divide A_h by any array on $g \cap h$ that has no zero values.

It is important, in applications, to recognize that if an array A factors on the hypergraph $\mathcal{H}$, and the hypergraph $\mathcal{H}^*$ covers $\mathcal{H}$, then A also factors on $\mathcal{H}^*$. Actually, this statement is precisely correct under our definitions only if $\cup\mathcal{H}=\cup\mathcal{H}^*$. If $\cup\mathcal{H}$ is a proper subset of $\cup\mathcal{H}^*$, then it is really only the vacuous extension $A^{\uparrow\cup\mathcal{H}^*}$ that factors on $\mathcal{H}^*$. To obtain a factorization of $A^{\uparrow\cup\mathcal{H}^*}$ on $\mathcal{H}^*$, it is necessary, in general, to assign each hyperedge h in $\mathcal{H}$ to a particular hyperedge in $\mathcal{H}^*$ that contains it. After we have done this, a particular hyperedge g in $\mathcal{H}^*$ may or may not have hyperedges from $\mathcal{H}$ assigned to it. If it has none assigned to it, we let B_g be the array I_g given by $I_g(\mathbf{x}) = 1$ for all $\mathbf{x} \in \mathcal{W}_g$. If it has one or more h from $\mathcal{H}$ assigned to it, we let B_g be I_g combined with the A_h for all the h assigned to it. Then we will have $A^{\uparrow\cup\mathcal{H}^*} = [\Pi\{B_g \mid g \in \mathcal{H}^*\}]$.

It is sufficient, in order for A to factor on $\mathcal{H}$, that A be proportional to a product of arrays on the hyperedges. Indeed, if

$$A \propto \Pi\{A_h \mid h \in \mathcal{H}\}, \tag{3.3}$$

where A_h is an array on h, then a representation of the form (3.2) can be obtained simply by incorporating the constant of proportionality into one of the A_h.

Though the theory in this chapter applies to arrays in general, we are mainly interested in factorizations of probability distributions. This means that we are concerned primarily with arrays that are potentials, for when a probability distribution P factors on a hypergraph, the arrays A_h in the factorization can be assumed to be potentials. Indeed, since P is not identically zero, none of the A_h can

be identically zero. And we can assume that $A_h(\mathbf{x}) \geq 0$ for all h and all $\mathbf{x}$. Since P does not take any negative values, the factorization would remain valid if we replaced all negative $A_h(\mathbf{x})$ by their absolute values.

In practice, relations of proportionality such as (3.3) are common when we are working with probability distributions. We will give a fuller account of the reasons for this in chapter 6; here let us say simply that factorizations of probability distributions are often reduced to proportionalities when we condition on observations.

If we are given potentials A_h and are told that the product $\Pi\{A_h \mid h \in \mathcal{H}\}$ is proportional to a probability distribution, then in principle we can find the constant of proportionality using the fact that the values of the probability distribution must add to one. In practice, this may be infeasible, since it requires a summation over the elements of the frame $\mathcal{W}_X$. As we shall see in the next section, however, this is not a serious problem when the hypergraph is a hypertree with hyperedges small enough to make local computation feasible. In that case, we can postpone finding the constant of proportionality until we have marginalized to a hyperedge using local computations.

3.3. Marginalizing Arrays

In this section, we introduce the idea of marginalizing an array from one set of variables to a smaller set of variables.

Suppose g and h are sets of variables, $h \subseteq g$, and G is an array on g. The *marginal* of G on h, denoted by $G^{\downarrow h}$, is an array on h. It is defined by

$$G^{\downarrow h}(\mathbf{x}) = \Sigma\{G(\mathbf{x}, \mathbf{y}) \mid \mathbf{y} \in \mathcal{W}_{g-h}\}$$

for all $\mathbf{x} \in \mathcal{W}_h$. For example, if G is an array on the variables {W, X, Y, Z}, then the marginal $G^{\downarrow\{W,X\}}$ is given by $G^{\downarrow\{W,X\}}(w, x) = \Sigma_{y,z} G(w, x, y, z)$, where the summation is over all possible values of Y and Z.

Proposition 3.1. If $h_1 \subseteq h_2 \subseteq g$ and G is an array on g, then $(G^{\downarrow h_2})^{\downarrow h_1} = G^{\downarrow h_1}$.

The distributive law of arithmetic says that c(a+b) = ca+cb. It follows from the distributive law that marginalization preserves proportionality; if A is an array, and c is a real number, then $(cA)^{\downarrow h} = cA^{\downarrow h}$. Here is a more general statement:

Proposition 3.2. If G and H are arrays on g and h, respectively, then $(GH)^{\downarrow g} = G(H^{\downarrow g \cap h})$.

When $h \subseteq g$ and the array P is a probability distribution on g, the marginal $P^{\downarrow h}$ is P's marginal on h in the usual probabilistic sense; $P^{\downarrow h}(\mathbf{x})$ is the probability that the variables in h take the values in **x**.

3.4. Marginalizing Factorizations

In this section, we learn how to adjust a factorization on a hypergraph to account for the deletion of a twig. This can be accomplished by local computations, computations involving only the arrays on the twig and a branch for the twig. This elimination of a twig by local computation is the key to the computation of marginals from a factorization on a hypertree, for by successively deleting twigs, we can reduce the hypertree to a single hyperedge.

Suppose $\mathcal{H}$ is a hypergraph on $\mathcal{X}$, t is a twig in $\mathcal{H}$, and b is a branch for t. The twig t may contain some vertices that are not contained in any other hyperedge in $\mathcal{H}$. These are the vertices in the set t–b. Deleting t from $\mathcal{H}$ means reducing $\mathcal{H}$ to the hypergraph $\mathcal{H}$–{t}, none of whose hyperedges contain any of the vertices in t-b.

Suppose A is an array on $\mathcal{X}$, suppose A factors on $\mathcal{H}$, and suppose we have stored A in factored form. In other words, we have stored an array A_h for each h in $\mathcal{H}$, and we know that $A = \Pi\{A_h \mid h \in \mathcal{H}\}$. Adjusting this factorization of A on $\mathcal{H}$ to account for the deletion of the twig t means reducing it to a factorization of $A^{\downarrow \mathcal{X}'}$ on

$\mathcal{H}$-{t}, where $\mathcal{X}'=\mathcal{X}-(t-b)=\cup(\mathcal{H}-\{t\})$. Can we do this? Yes. The following proposition tells us that if A factors on $\mathcal{H}$, then $A^{\downarrow\mathcal{X}'}$ factors on $\mathcal{H}-\{t\}$, and the second factorization can be obtained from the first by a local computation that involves only t and a branch.

Proposition 3.3. Under the assumptions of the preceding paragraph,

$$A^{\downarrow\mathcal{X}'} = (A_b A_t^{\downarrow t\cap b}) \Pi\{A_h \mid h\in\mathcal{H}-\{t, b\}\}. \tag{3.4}$$

Formula (3.4) says that $A^{\downarrow\mathcal{X}'}$ factors on the hypergraph $\mathcal{H}-\{t\}$. The potential on b is multiplied by $A_t^{\downarrow t\cap b}$, and the potentials on the other elements of $\mathcal{H}-\{t\}$ are unchanged.

This result is especially interesting in the case of hypertrees, because in this case repeated application of (3.4) allows us to obtain A's marginal on any particular hyperedge of $\mathcal{H}$. If we want the marginal on a hyperedge h_1, we choose a construction sequence beginning with h_1, say $h_1h_2...h_n$, and we choose a branching for this construction sequence. Let $\mathcal{X}_k$ denote $h_1\cup...\cup h_k$, and let $\mathcal{H}_k$ denote $\{h_1, h_2, ..., h_k\}$ for k=1, ..., n–1. We use (3.4) to delete the twig h_n, so that we have a factorization of $A^{\downarrow\mathcal{X}_{n-1}}$ on the hypertree $\mathcal{H}_{n-1}$. Then we use (3.4) again to delete the twig h_{n-1}, so that we have a factorization of $A^{\downarrow\mathcal{X}_{n-2}}$ on the hypertree $\mathcal{H}_{n-2}$. And so on, until we have deleted all the hyperedges except h_1, so that we have a factorization of $A^{\downarrow\mathcal{X}_1}$ on the hypertree $\mathcal{H}_1$—i.e., we have the marginal $A^{\downarrow h_1}$. At each step, the computation is local, in the sense that it involves only a twig and a branch.

We are most interested, of course, in the case where A is a probability distribution. In this case, as we mentioned in the preceding section, the factorization we wish to marginalize may be a proportionality rather than an equality. In other words, we may begin with a factorization of a potential that is only proportional to the probability distribution that interests us. Eventually, we will need to find the constant of proportionality, but since marginalization preserves proportionality, we may postpone the normalization until the final step, where we have reduced the potential to its marginal on the single hyperedge with which we are concerned, and hence normalization requires summation only over the frame for this hyperedge.

3.5. Computing Marginals in Markov Trees

As we learned in chapter 2, the choice of a branching for a hypertree determines a Markov tree for the hypertree. We now look at our scheme for computing a marginal from the viewpoint of this Markov tree. This change in viewpoint does not necessarily affect the implementation of the computation, but it gives us a richer understanding. It gives us a picture in which message passing, instead of deletion, is the dominant metaphor, and in which we have great flexibility in how the message passing is controlled.

Why did we talk about deleting the hyperedge h_k as we projected h_k's array to the branch $\beta(h_k)$? The point was simply to remove h_k from our attention. The "deletion" had no computational significance, but it helped make clear that h_k and the array on it were of no further use. What was of further use was the smaller hypertree that would remain were h_k deleted.

When we turn from the hypertree to the Markov tree, deletion of twigs translates into deletion of leaves. But a tree is easier to visualize than a hypertree. We can remove a leaf or a whole branch of a tree from our attention without leaning so heavily on metaphorical deletion. And a Markov tree also allows another, more useful, metaphor. We can imagine that each vertex of the tree is a processor, and we can imagine that the marginal is a message that one processor passes to another. Within this metaphor, vertices no longer relevant are kept out of our way by the rules guiding the message passing, not by deletion.

We cover a number of topics in this section. We begin by reviewing our marginalization scheme in the hypertree setting and seeing how its details translate into the Markov tree setting. We formulate precise descriptions of the operations that are carried out by each vertex and precise definitions of the messages that are passed from one vertex to another. Then we turn to questions of timing—whether a vertex uses a message as soon as it is received or waits for all its messages before it acts, how the order in which the vertices act is constrained, and whether the vertices act in serial or in parallel. We explain how the Markov tree can be expanded into an architecture for the parallel computation, with provisions for storing messages as well as directing them. We explain how this architecture handles updating when inputs are changed. And finally, we explain how our computation can be directed by a simple forward-chaining production system.

Translating to the Markov Tree. We now translate our marginalization scheme from the hypertree to the Markov tree.

Recall the details in the hypertree setting. We have an array A on $\mathcal{X}$, in the form of a factorization on a hypertree $\mathcal{H}$. We want the marginal for the hyperedge h_1. We choose a hypertree construction sequence with h_1 as its root, say $h_1h_2...h_n$, and we choose a branching β for $h_1h_2...h_n$. On each hyperedge h_i, we have an array A_{h_i}. We repeatedly apply the following operation:

> *Operation H*. Marginalize the array now on h_k to $\beta(h_k)$. Change the array now on $\beta(h_k)$ by combining it with this marginal.

We apply Operation H first for k=n, then for k=n–1, and so on, down to k=2. The array assigned to h_1 at the end of this process is the marginal on h_1.
We want now to re-describe Operation H, and the process of its repeated application, in terms of the actions of processors located at the vertices of the Markov tree $(\mathcal{H}, \mathcal{E})$ determined by the branching β.

The vertices of $(\mathcal{H}, \mathcal{E})$ are the hyperedges $h_1, h_2, ..., h_n$. We imagine that a processor is attached to each of the h_i. The processor attached to h_i can store an array defined on h_i, can compute the marginal of this array on $h_i \cap h_j$, where h_j is a neighboring vertex, can send the marginal to h_j as a message, can accept an array on h_i (or any smaller set of variables) as a message from a neighbor, and can change the array it has stored by combining it with such an incoming message.

The edges of $(\mathcal{H}, \mathcal{E})$ are

$$\{h_n, \beta(h_n)\}, \{h_{n-1}, \beta(h_{n-1})\}, ..., \{h_3, \beta(h_3)\}, \{h_2, h_1\}.$$

When we move from h_n to $\beta(h_n)$, then from h_{n-1} to $\beta(h_{n-1})$, and so on, we are moving inwards in this Markov tree, from the outer leaves to the root h_1. The repeated application of Operation H by the processors located at the vertices follows this path.

In order to recast Operation H in terms of these processors, we need some more notation. Let Cur_h denote the array currently stored by the processor at vertex h of $(\mathcal{H}, \mathcal{E})$. In terms of the local processors and the Cur_h, Operation H becomes the following:

Operation M_1. Vertex h computes $Cur_h^{\downarrow h\cap\beta(h)}$, the marginal of Cur_h to $\beta(h)$. It sends $Cur_h^{\downarrow h\cap\beta(h)}$ as a message to vertex $\beta(h)$. Vertex $\beta(h)$ accepts the message $Cur_h^{\downarrow h\cap\beta(h)}$ and changes $Cur_{\beta(h)}$ by multiplying it by $Cur_h^{\downarrow h\cap\beta(h)}$.

At the outset, $Cur_h = A_h$ for every vertex h. Operation M_1 is executed first for $h=h_n$, then for $h=h_{n-1}$, and so on, down to $h=h_2$. At the end of this propagation process, the array Cur_{h_1}, the array stored at h_1, is the marginal of A on h_1.

An Alternative Operation. Operation M_1 prescribes actions by two processors, h and $\beta(h)$. We now give an alternative, Operation M_2, which is executed by a single processor. Since it is executed by a single processor, Operation M_2 will be easier for us to think about when we discuss alternative control regimes for the process of propagation.

Operation M_2 differs from Operation M_1 only in that it requires a processor to combine the messages it receives all at once, rather than incorporating them into the combination one by one as they arrive. Each time Operation M_1 is executed for an h such that $\beta(h)=g$, the processor g must change the array it stores by combining it with the incoming message. But if processor g can store all its incoming messages, then it can delay the combination until it is its turn to marginalize. If we take this approach, then we can replace Operation M_1 with the following:

Operation M_{2a}. Vertex h combines the array A_h with all the messages it has received, and it calls the result Cur_h. Then it computes $Cur_h^{\downarrow h\cap\beta(h)}$, the marginal of Cur_h to $h\cap\beta(h)$. It sends $Cur_h^{\downarrow h\cap\beta(h)}$ as a message to $\beta(h)$.

Operation M_{2a} involves action by only one processor, the processor h. When Operation M_{2a} is executed by h_n, there is no combination, because h_n, being a leaf in the Markov tree, has received no messages. The same is true for the other leaves in the Markov tree. But for vertices that are not leaves in the Markov tree, the operation will involve both combination and marginalization.

After Operation M_{2a} has been executed by h_n, h_{n-1}, and so on down to h_2, the root h_1 will have received a number of messages but will not yet have acted. To

complete the process, h_1 must combine all its messages and its original array A_{h_1}, thus obtaining the marginal $A^{\downarrow h_1}$. We may call this Operation M_{2b}:

> *Operation M_{2b}.* Vertex h combines the array A_h with all the messages it has received, and it reports the result to the user of the system.

So Operation M_2 actually consists of two operations. Operation M_{2a} is executed successively by h_n, h_{n-1}, and so on down to h_2. Then Operation M_{2b} is executed by h_1.

Operation M_2 simplifies our thinking about control, or the flow of computation, because it allows us to think of control as moving with the computation in the Markov tree. In our marginalization scheme, control moves from one vertex to another, from the outer leaves inward towards the root. If we use Operation M_2, then a vertex is computing only when it has control.

Formulas for the Messages. We have described verbally how each vertex computes the message it sends to its branch. Now we will translate this verbal description into a formula that constitutes a recursive definition of the messages. The formula will not make much immediate contribution to our understanding, but it will serve as a useful reference in the next section, when we discuss how to extend our scheme for computing a single marginal to a scheme for computing all marginals.

Let $M^{h\rightarrow\beta(h)}$ denote the message sent by vertex h to its bud. Our description of Operation M_{2a} tells us that

$$M^{h\rightarrow\beta(h)} = Cur_h^{\downarrow h\cap\beta(h)},$$

where

$$Cur_h = A_h \,\Pi\{M^{g\rightarrow\beta(g)} \mid g\in\mathcal{H} \text{ and } \beta(g)=h\}.$$

Putting these two formulas together, we have

$$M^{h\rightarrow\beta(h)} = (A_h \,\Pi\{M^{g\rightarrow\beta(g)} \mid g\in\mathcal{H} \text{ and } \beta(g)=h\})^{\downarrow h\cap\beta(h)}. \tag{3.5}$$

If h is a leaf, then there is no $g\in\mathcal{H}$ such that $h=\beta(g)$, and so (3.5) reduces to

$$M^{h\rightarrow\beta(h)} = A_h^{\downarrow h\cap\beta(h)}, \tag{3.6}$$

by the convention that an empty combination is equal to one.

Formula (3.5) constitutes a recursive definition of $M^{h\to\beta(h)}$ for all h, excepting only the root h_1 of the budding β. The special case (3.6) defines $M^{h\to\beta(h)}$ for the leaves; a further application of (3.5) defines $M^{h\to\beta(h)}$ for vertices one step in towards the root from the leaves; a third application defines $M^{h\to\beta(h)}$ for vertices two steps in towards the root from the leaves; and so on.

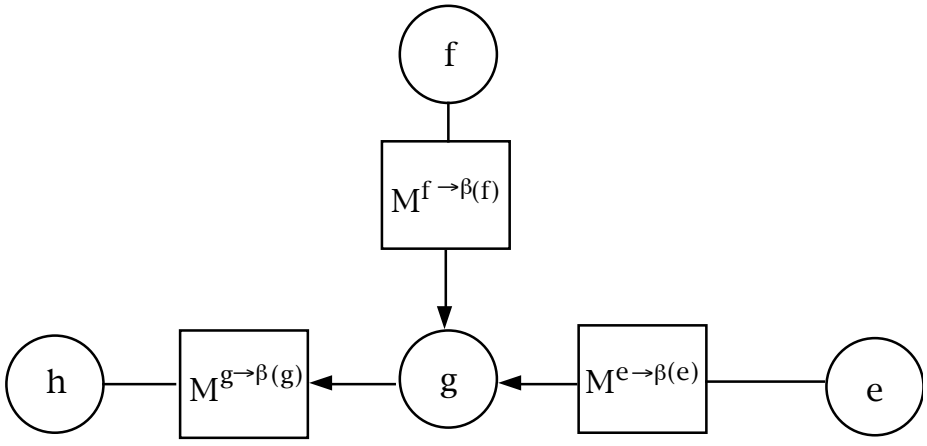
We can also represent Operation M_{2b} by a formula:

$$A^{\downarrow h} = A_h \, \Pi\{M^{g\to\beta(g)} \mid g\in\mathcal{H} \text{ and } \beta(g)=h\}. \qquad (3.7)$$

Storing the Messages. If we want to think in terms of Operation M_2, then we must imagine that our processors have a way to store incoming messages.

Figure 3.1 depicts an architecture that provides for such storage. The figure shows a storage register at vertex g for each of g's neighbors. The registers for neighbors on the side of g away from the goal vertex are used to store incoming messages. The register for the neighbor in the direction of the goal vertex is used to store the vertex's outgoing message. The registers serve as communication links between neighbors; the outgoing register for one vertex being the incoming register for its neighbor in the direction of the goal vertex.

Figure 3.1. A typical vertex processor g, with incoming messages from vertices f and e and outgoing message to h; here $g=\beta(f)=\beta(e)$ and $h=\beta(g)$.

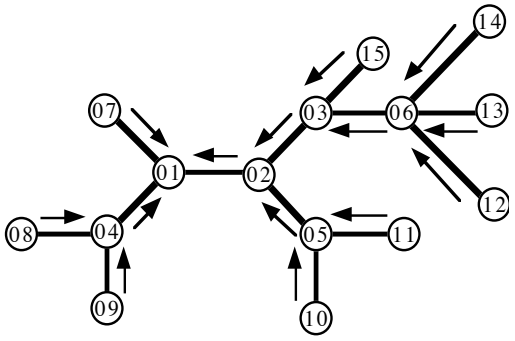


The message $M^{g \to \beta(g)}$, which vertex g stores in the register linking g to its bud, is an array on $g \cap \beta(g)$. It is the marginal for the bud of an array on g.

Flexibility of Control. Whether we use operation M_1 or M_2, it is not necessary to follow exactly the order h_n, h_{n-1}, and so on. The final result will be the same provided only that a processor never sends a message until after it has received and absorbed all the messages it is supposed to receive.

This point is obvious when we look at a picture of the Markov tree. Consider, for example a Markov tree with 15 vertices, as in Figure 3.2. The vertices are numbered from 1 to 15 in this picture, indicating a construction sequence $h_1 h_2 ... h_{15}$. Since we want to find the marginal for vertex 1, all our messages will be sent towards vertex 1, in the directions indicated by the arrows. Our scheme calls for a message from vertex 15 to vertex 3, then a message from vertex 14 to vertex 6, and so on. But we could just as well begin with messages from 10 and 11 to 5, follow with a message from 5 to 2, then messages from 12, 13, and 14 to 6, from 6 and 15 to 3, and so on.

Figure 3.2. A tree with 15 vertices.

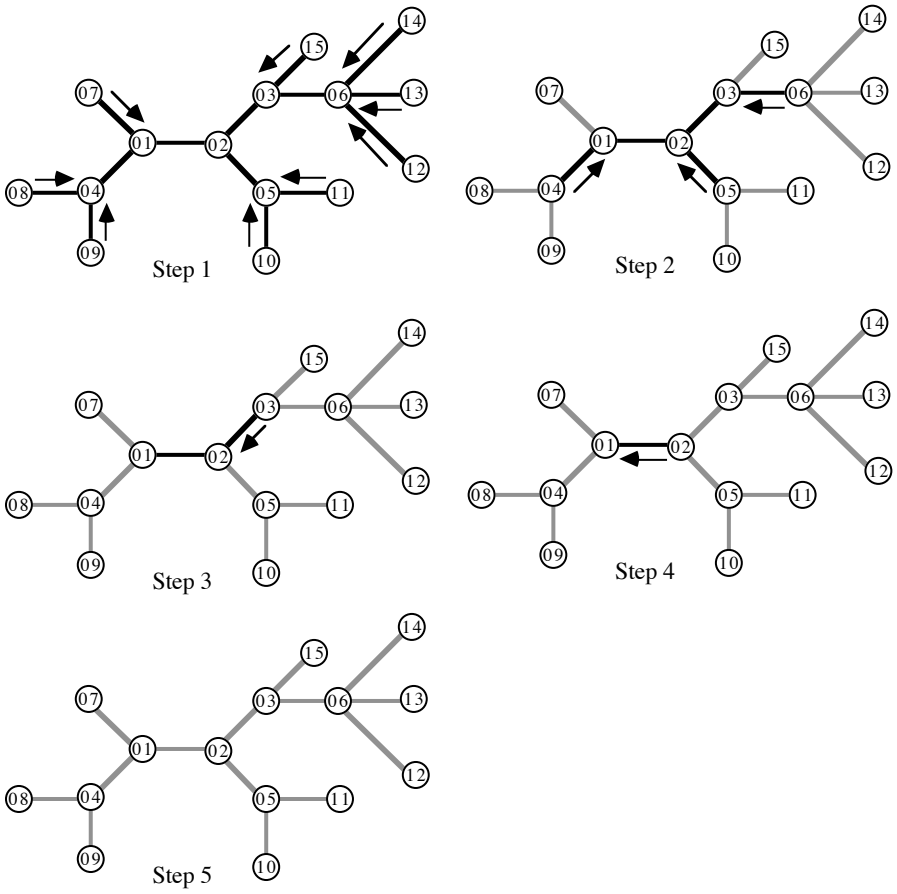


Returning to the metaphor of deletion, where each vertex is deleted when it sends its message, we can say that the only constraint on the order in which the vertices act is that each vertex must be a leaf when it acts; all the vertices that used it as a branch must have sent their messages to it and then been deleted, leaving it a leaf.

The different orders of marginalization that obey this constraint correspond, of course, to the different tree construction sequences for ($\mathcal{H}$, $\mathcal{E}$) that use the branching β.

So far, we have been thinking about different sequences in which the vertices might act. This is most appropriate if we are really implementing the scheme on a serial computer. But if the different vertices really did have independent processors that could operate in parallel, then some of the vertices could act simultaneously. Figure 3.3 illustrates one way this might go for the Markov tree of Figure 3.2. In step 1, all the leaf processors project to their branches. In step 2, vertices 4, 5, and 6 (which would be leaves were the original leaves deleted) project. And so on.

Figure 3.3. An example of the message-passing scheme for computation of the marginal of vertex 1.



If the different processors take different amounts of time to perform Operation M_2 on their inputs, then the lock-step timing of Figure 3.3 may not provide the quickest way to find the marginal for h_1. It may be quicker to allow a processor to act as soon as it receives messages from its leaves, whether or not all the other processors that started along with these leaves have finished.

In general, the only constraint, in the parallel as in the serial case, is that action move inwards towards h_1. Each vertex must receive and absorb all its messages from vertices farther away from h_1 before sending its own message on towards h_1. (In terms of Figure 3.1, each processor must wait until all its incoming registers are filled before it can compute a message to put in its outgoing register.) If we want to get the job done as quickly as possible, we will demand that each processor go to work as quickly as possible subject to this constraint. But the job will get done eventually provided only that all the processors act eventually. It will get done, for example, if each processor checks on its inputs periodically or at random times and acts if it has those inputs [Pearl 1986].

If we tell each processor who its neighbors are and which one of these neighbors lies on the path towards the goal, then no further global control or synchronization is needed. Each processor knows that it should send its outgoing message as soon as it can after receiving all its incoming messages. The leaf processors, which have no incoming messages, can act immediately. The others must wait their turn.

Updating Messages. Suppose we have completed the computation of $A^{\downarrow h_1}$, the marginal for our goal vertex. And suppose we now find reason to change A by changing one or more of our inputs, the A_h. If we have implemented the architecture just described, with storage registers between each of the vertices, then we may be able to update the marginal $A^{\downarrow h_1}$ without discarding all the work we have already done. If we leave some of the inputs unchanged, then some of the computations may not need to be repeated.

Unnecessary computation can be avoided without global control. We simply need a way of marking arrays, to indicate that they have received any needed updating. Suppose the processor at each vertex h can recognize the mark on any of its inputs (on A_h, our direct input, or on any message $M^{g\to\beta(g)}$ from a vertex g that has h as its bud), and can write the mark on its own output, the message $M^{h\to\beta(h)}$. When we wish to update the computation of $A^{\downarrow h_1}$, we put in the new values for

those A_h we wish to change, and we mark all the A_h, both the ones we have changed, and the others, which we do not want to change. Then we run the system as before, except that a processor, instead of waiting for its incoming registers to be full before it acts, waits until all its inputs are marked. The processor can recognize when an input is marked without being changed, and in this case it simply marks its output instead of re-computing it.

Of course, updating can also be achieved with much less control. As Pearl [1986] has emphasized, hardly any control at all is needed if we are indifferent to the possibility of wasted effort. If we do not care whether a processor repeats the same computations, we can forget about marking arrays and simply allow each processor to re-compute its output from its inputs periodically or at random times. Under these circumstances, any change in one of the A_g will eventually be propagated through the system to change $A^{\downarrow h_1}$.

The idea of updating is theoretically important in probability theory because of the example of conditioning. We often want to condition a probability distribution on the observed values of one or more variables. Conditioning on a variable X can be achieved by multiplying a factorization of the probability distribution by a potential on X. Since this new potential on X can be incorporated in the potential on any hyperedge containing X, conditioning on X can be achieved by changing the input potential in just one of the hyperedges in the hypertree. We will give an example of this in section 3.7, in this chapter. We will develop the general theory in chapter 6.

A Simple Production System. In reality, we will never have a parallel computer organized precisely to fit our problem. Our story about passing messages between independent processors should be thought of as metaphor, not as a guide to implementation. Implementations can take advantage, however, of the modularity the metaphor reveals.

One way to take advantage of this modularity, even on a serial computer, is to implement the computational scheme in a simple forward-chaining production system. A forward-chaining production system consists of a working memory and a rule-base, a set of rules for changing the contents of the memory. (See Brownston et al. [1985] or Davis and King [1984].)

A very simple production system is adequate for our problem. We need a working memory that initially contains A_h for each vertex h of $(\mathcal{H}, \mathcal{E})$, and a rule-base consisting of just two rules, corresponding to Operations M_{2a} and M_{2b}.

> **Rule 1**: If A_h is in working memory and $M^{g \to \beta(g)}$ is in working memory for every g such that $\beta(g)=h$, then use (3.6) to compute $M^{h \to \beta(h)}$, and place it in working memory.

> **Rule 2**: If A_{h_1} is in working memory and $M^{g \to \beta(g)}$ is in working memory for every g such that $\beta(g)=h_1$, then use (3.7) to compute $A^{\downarrow h_1}$, and print the result.

Initially, there will be no $M^{g \to \beta(g)}$ at all in working memory, so Rule 1 can fire only for h such that there is no g with $\beta(g)=h$—i.e., only for h that are leaves. But eventually Rule 1 will fire for every vertex except the root h_1. Then Rule 2 will fire, completing the computation. Altogether, there will be n firings, one for each vertex in the Markov tree.

Production systems are usually implemented so that a rule will fire only once for a given instantiation of its antecedent; this is called *refraction* [Brownston et al. 1985, pp. 62–63]. If our simple production system is implemented with refraction, there will be no unnecessary firings of rules; only the n firings that are needed will occur. Even without refraction, however, the computation will eventually be completed.

Since refraction allows a rule to fire again for a given instantiation when the inputs for that instantiation are changed, this simple production system will also handle updating efficiently, performing only those re-computations that are necessary.

3.6. Simultaneous Propagation in Markov Trees

In the preceding section, we were concerned with the computation of the marginal on a single vertex of the Markov tree. In this section, we will be concerned with

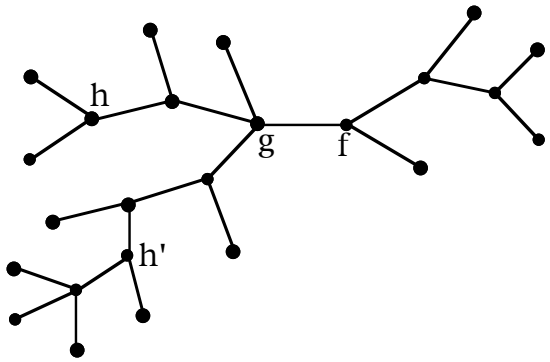
how to compute the marginals on all vertices simultaneously. As we will see, this can be done efficiently with only slight changes in architecture or rules.

Computing all the Marginals. If we can compute the marginal of A on one hyperedge in $\mathcal{H}$, then we can compute the marginals on all the hyperedges in $\mathcal{H}$. We simply compute them one after the other. It is obvious, however, that this will involve much duplication of effort. How can we avoid the duplication?

The first point to notice in answering this question is that we only need one Markov tree. Though there may be many Markov tree representatives for $\mathcal{H}$, any one of them can serve for the computation of all the marginals. Once we have chosen a Markov tree representative $(\mathcal{H}, \mathcal{E})$, then no matter which element h of $\mathcal{H}$ interests us, we can choose a tree construction sequence for $(\mathcal{H}, \mathcal{E})$ that begins with h, and since this sequence is also a hypertree construction sequence for $\mathcal{H}$, we can apply the method of section 3.4 to it to compute $A^{\downarrow h}$.

The second point to notice is that the message passed from one vertex to another, say from f to g, will be the same no matter what marginal we are computing. If β is the budding that we use to compute $A^{\downarrow h}$, the marginal on h, and β' is the budding we use to compute $A^{\downarrow h'}$, and if $\beta(f)=\beta'(f)=g$, then the message $M^{f\to\beta(f)}$ that we send from f to g when computing $A^{\downarrow h}$ is the same as the message $M^{f\to\beta'(f)}$ that we send from f to g when computing $A^{\downarrow h'}$. To see that this is true, think about Figure 3.4. Since g is the branch for f in both computations, h and h' are both on the g side of the edge {f, g}. All the computation leading up to the message passed from f to g takes place on the other side, the f side, and is not influenced by whether the ultimate goal, after the passage through f and g, is h or h'.

Figure 3.4.



Since the value of $M^{f\to\beta(f)}$ does not depend on the budding β, we may write $M^{f\to g}$ instead of $M^{f\to\beta(f)}$ when $\beta(f)=g$.

If we compute marginals for all the vertices, then we will eventually compute both $M^{f\to g}$ and $M^{g\to f}$ for every edge $\{f, g\}$. We will compute $M^{f\to g}$ when we compute the marginal on g or on any other vertex on the g side of the edge, and we will compute $M^{g\to f}$ when we compute the marginal on g or on any other vertex on the g side of the edge.

We can easily generalize the recursive definition of $M^{g\to\beta(g)}$ that we gave in section 3.5 to a recursive definition of $M^{g\to h}$ for all neighbors g and h. To do so, we merely restate (3.5) in a way that replaces references to the budding β by references to neighbors and the direction of the message. We obtain

$$M^{g\to h} = (A_g\, \Pi\{M^{f\to g} \mid f\in(\mathcal{N}_g - \{h\})\})^{\downarrow g\cap h}, \tag{3.8}$$

where $\mathcal{N}_g$ is the set of all g's neighbors in $(\mathcal{H}, \mathcal{E})$. If g is a leaf vertex, then (3.8) reduces to $M^{g\to h} = A_g{}^{\downarrow g\cap h}$.

After we carry out the recursion to compute $M^{g\to h}$ for all pairs of neighbors g and h, we can compute the marginal of A on each h by

$$A^{\downarrow h} = A_h\, \Pi\{M^{g\to h} \mid g\in\mathcal{N}_h\}. \tag{3.9}$$

The General Architecture. A slight modification of the architecture shown in Figure 3.1 will allow us to implement the simultaneous computation of the marginals on all the hyperedges. We simply put both two storage registers between

every pair of neighbors f and g, as in Figure 3.5. One register stores the message from f to g; the other stores the message from g to f.

Figure 3.6 shows a more elaborate architecture for the simultaneous computation. In addition to the storage registers that communicate between vertices, this figure shows registers where the original arrays, the A_h, are put into the system and the marginals, the $A^{\downarrow h}$, are read out.

Figure 3.5. The two storage registers between f and g.

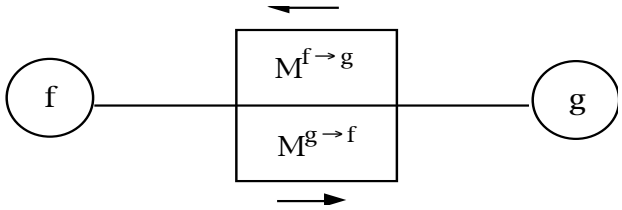
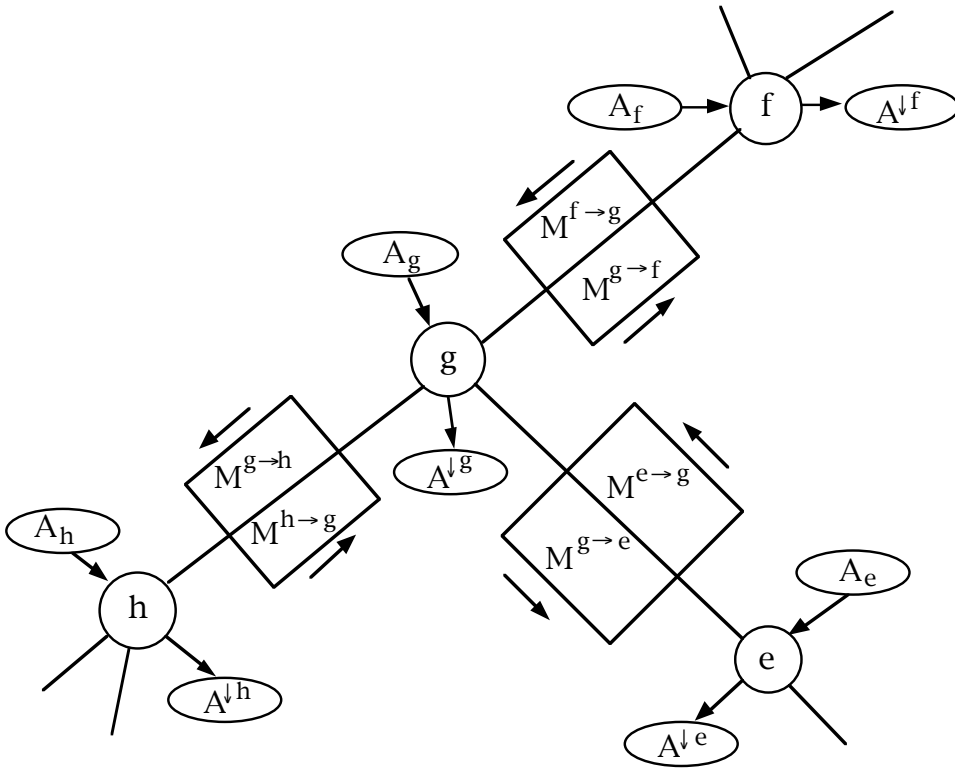


Figure 3.6. Several vertices, with storage registers for communication between themselves and with the user.



In the architecture of Figure 3.1, computation is controlled by the simple requirement that a vertex g must have messages in all its incoming registers before it can compute a message to place in its outgoing register. In the architecture of Figure 3.6, computation is controlled by the requirement that a vertex g must have messages in all its incoming registers except the one from h before it can compute a message to send to h.

This basic requirement leaves room for a variety of control regimes. Most of the comments we made about the flexibility of control for Figure 3.1 carry over to Figure 3.6.

In particular, updating can be handled efficiently if a method is provided for marking updated inputs and messages. If we change just one of the input, then efficient updating will save about half the work involved in simply re-performing the entire computation. To see that this is so, consider the effect of changing the input A_h in Figure 3.4. This will change the message $M^{g\to f}$, but not the message

$M^{f \to g}$. The same will be true for every edge; one of the two messages will have to be recomputed, but not the other.

It may be enlightening to look at how the lock-step control we illustrated with Figure 3.3 might generalize to simultaneous computation of the marginals for all vertices. Consider a lock-step regime where at each step, each vertex looks and sees what messages it has the information to compute, computes these messages, and sends them. After all the vertices working are done, they look again, see what other messages they now have the information to compute, compute these messages, and send them. And so on. Figure 3.7 gives an example. At the first step, the only messages that can be computed are the messages from the leaves to their branches. At the second step, the computation moves inward. Finally, at step 3, it reaches vertex 2, which then has the information needed to compute its own marginal and messages for all its neighbors. Then the messages move back out towards the leaves, with each vertex along the way being able to compute its own marginal and messages for all its other neighbors as soon as it receives the message from its neighbor nearest vertex 2.

In the first phase, the inward phase, a vertex sends a message to only one of its neighbors, the neighbor towards the center. In the second phase, the outward phase, a vertex sends k–1 messages, where k is the number of its neighbors. Yet the number of messages sent in the two phases is roughly the same, because the leaf vertices participate in the first phase and not in the second.

There are seven vertices in the longest path in the tree of Figure 3.7. Whenever the number of vertices in the longest path is odd, the lock-step control regime will result in computation proceeding inwards to a central vertex and then proceeding back outwards to the leaves. Whenever this number is even, there will instead be two central vertices that send each other messages simultaneously, after which they both send messages back outwards towards the leaves.

If we really do have independent processors for each vertex, then we do not have to wait for all the computations that start together to finish before taking advantage of the ones that are finished to start new ones. We can allow a new computation to start whenever a processor is free and it has the information needed. On the other hand, we need not require that the work be done so promptly. We can assume that processors look for work to do only at random times. But no matter how we handle these issues, the computation will converge to some

particular vertex or pair of neighboring vertices and then move back out from that vertex or pair of vertices.

There is exactly twice as much message passing in our scheme for simultaneous computation as there was in our scheme for computing a single marginal. Here every pair of neighbors exchange messages; there only one message was sent between every pair of neighbors. Notice also that we can make the computation of any given marginal the beginning of the simultaneous computation. We can single out any hyperedge h (even a leaf), and forbid it to send a message to any neighbor until it has received messages from all its neighbors. If we then let the system of Figure 3.7 run, it will behave just like the system of Figure 3.3 with h as the root, until h has received messages from all its neighbors. At that point, h can compute its marginal and can also send messages to all its neighbors; the second half of the message passing then proceeds, with messages moving back in the other direction.

The Corresponding Production System. Implementing simultaneous computation in a production system requires only slight changes in our two rules. The following will work:

> **Rule 1'**: If A_g is in working memory, and $M^{f\to g}$ is in working memory for every f in $\mathcal{N}_g - \{h\}$, then use (3.8) to compute $M^{g\to h}$, and place it in working memory.

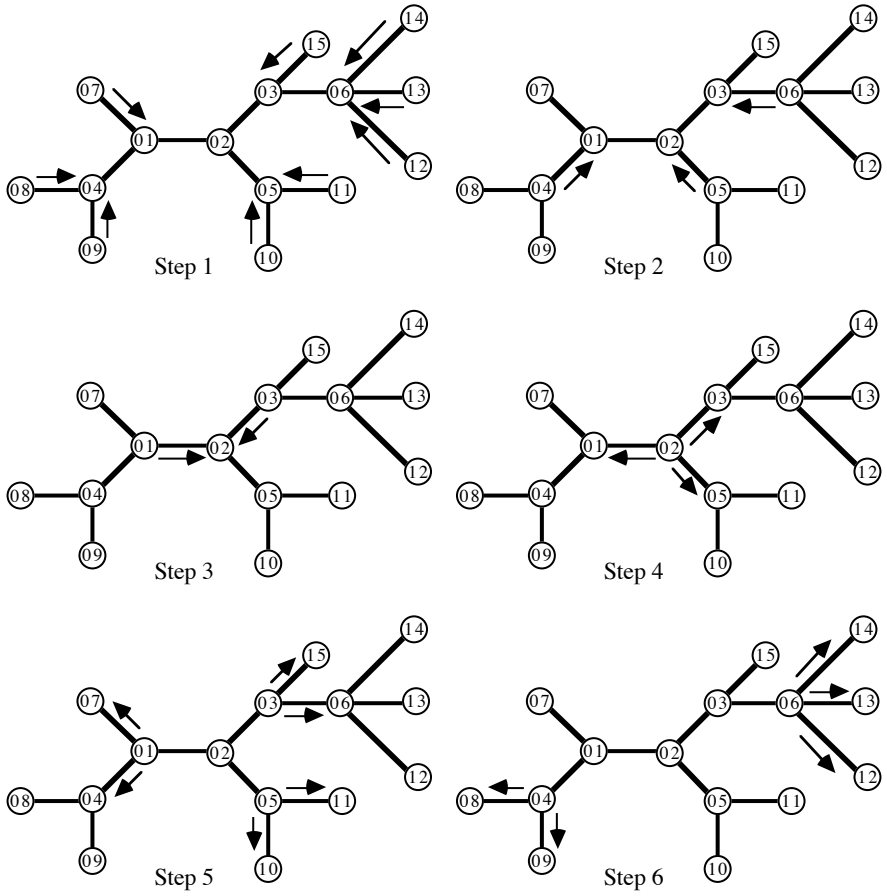
> **Rule 2'**: If A_h is in working memory, and $M^{g\to h}$ is in working memory for every g in $\mathcal{N}_h$, then use (3.9) to compute $A^{\downarrow h}$, and print the result.

Initially, there will be no $M^{f\to g}$ at all in working memory, so Rule 1' can fire only for g and h such that $\mathcal{N}_g - \{h\}$ is empty—i.e., only when g is a leaf and h is its bud. But eventually Rule 1' will fire in both directions for every edge {g, h}. Once Rule 1' has fired for all the neighbors g of h, in the direction of h, Rule 2' will fire for h. Altogether, there will be 3n–2 firings, two firings of Rule 1' for each of the n–1 edges, and one firing of Rule 2' for each of the n vertices.

As the count of firings indicates, our scheme for simultaneous computation finds marginals for all the vertices with roughly the same effort that would be

required to find marginals for three vertices if this were done by running the scheme of section 3.5 three times.

Figure 3.7. An example of the message-passing scheme for simultaneous computation of all marginals.



3.7 An Example

This example is adapted from Shachter and Heckerman [1987]. Consider three variables D, B and G representing diabetes, blue toe, and glucose in urine,

respectively. The frame for each variable has two configurations. D=d will represent the proposition *diabetes is present* (in some patient) and D=~d will represent the proposition *diabetes is not present*. Similarly for B and G. Let P denote the joint probability distribution for {D, B, G}. We will assume that B and G are conditionally independent (with respect to P) given D. As we will explain in chapter 6, this means that we can factor P as follows.

$$P = P^{D}\, P^{B|D}\, P^{G|D} \tag{3.10}$$

where P^{D} is the marginal on {D} ($P^{D}=P^{\downarrow\{D\}}$), $P^{B|D}$ is a potential on {D, B} called the conditional of B given D, and $P^{G|D}$ is a potential on {D, G} called the conditional of G given D. (This means that $P^{D}(d)$ is the probability that D=d, $P^{B|D}(d, b)$ is the conditional probability that B=b given that D=d, and so on.)

Suppose the potentials P^{D}, $P^{B|D}$, and $P^{G|D}$ have the values shown in Figure 3.8.

Figure 3.8. The potentials P^{D}, $P^{B|D}$, and $P^{G|D}$.

P^{D}	
d	.1
~d	.9

$P^{B|D}$	
d,b	.014
d,~b	.986
~d,b	.006
~d,~b	.994

$P^{G|D}$	
d,g	.9
d,~g	.1
~d,g	.01
~d,~g	.99

Formula (3.10) tells us that P factors on the hypertree {{D}, {D, B}, {D, G}}. Since we would like to compute the marginals for B and G, we will expand the hypertree to include the hyperedges {B} and {G}. This does not take us outside of our theory, because we can replace (3.10) with the equivalent formula

$$P = P^{D}\, P^{B|D}\, P^{G|D}\, I_{\{B\}}\, I_{\{G\}},$$

where $I_{\{B\}}(\mathbf{x}) = 1$ for all $\mathbf{x}\in\mathcal{W}_{\{B\}}$, and $I_{\{G\}}(\mathbf{x}) = 1$ for all $\mathbf{x}\in\mathcal{W}_{\{G\}}$. Figure 3.9 show the expanded hypertree and a Markov tree representative.

Figure 3.9. The hypertree and a Markov tree representative.

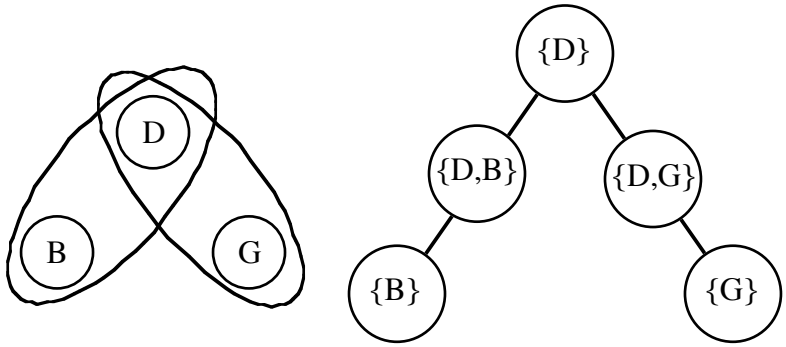
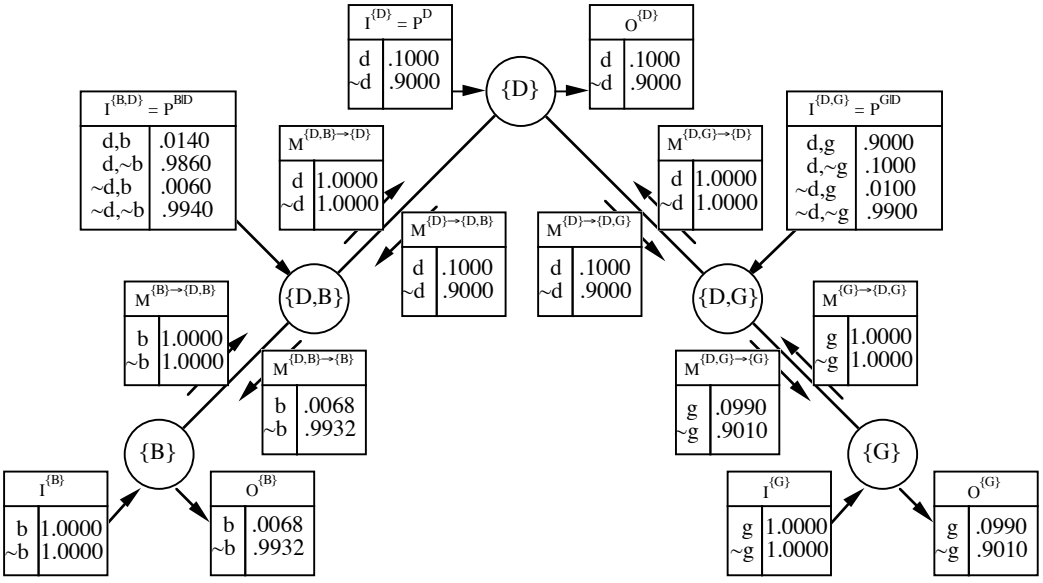


Figure 3.10 shows the results of propagating the potentials following the scheme described in section 2. For each vertex h, the input potentials are shown as I^h and the output potentials are shown as O^h. All the messages are also shown.

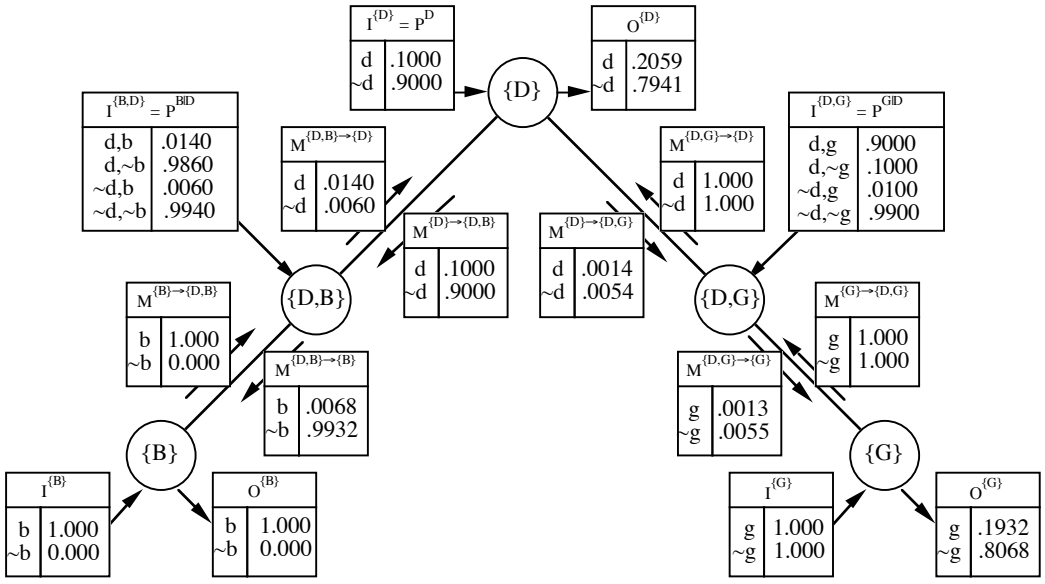
Figure 3.10. The initial propagation of potentials.



Now suppose we observe that the patient has blue toe. As we will explain in chapter 6, this can be taken into account by changing the input for variable B to the

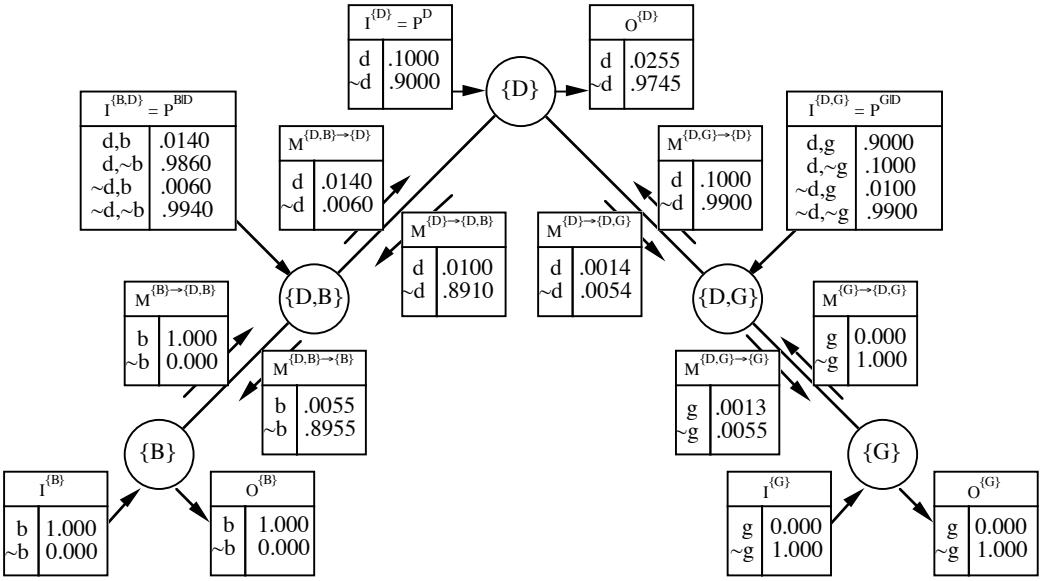
array that assigns the value 1 to b and the value 0 to ~b. If we change this one input and leave the other inputs unchanged, the product of the inputs will be proportional to the posterior probability distribution given the observation. Thus we can find the posterior marginal probabilities by propagating and then normalizing the output potentials. This is done in Figure 3.11.

Figure 3.11. The results of propagation after the presence of blue toe is observed.



The probability for diabetes has increased from .1 to .2059 and consequently the probability for glucose in urine has also increased from .0990 to .1932. Now suppose the patient is tested for glucose in urine, the results indicate that there is none. This information is represented by a potential that assigns the value 0 to g and the value 0 to ~g. The other potentials remain the same as before. Figure 3.12 shows the results of propagating now. The probability of diabetes has decreased from .2059 to .0255.

Figure 3.12. The results of propagation after absence of glucose in urine is observed.



3.8. Proofs

Proof of Proposition 3.1. If $h_1 \subseteq h_2 \subseteq g$ and G is an array on g, then $(G^{\downarrow h_2})^{\downarrow h_1} = G^{\downarrow h_1}$.

$$\begin{aligned}(G^{\downarrow h_2})^{\downarrow h_1}(\mathbf{x}) &= \Sigma\{(G^{\downarrow h_2})(\mathbf{x}, \mathbf{y}) \mid \mathbf{y} \in \mathcal{W}_{h_2 - h_1}\} \\ &= \Sigma\{\Sigma\{G(\mathbf{x}, \mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{g - h_2}\} \mid \mathbf{y} \in \mathcal{W}_{h_2 - h_1}\} \\ &= \Sigma\{G(\mathbf{x}, \mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{g - h_2}, \mathbf{y} \in \mathcal{W}_{h_2 - h_1}\} \\ &= \Sigma\{G(\mathbf{x}, \mathbf{w}) \mid \mathbf{w} \in \mathcal{W}_{g - h_1}\} \\ &= G^{\downarrow h_1}(\mathbf{x}).\end{aligned}$$

Proof of Proposition 3.2. Suppose G and H are arrays on g and h respectively. Then

$$(GH)^{\downarrow g}(\mathbf{x}) = \Sigma\{(GH)(\mathbf{x}, \mathbf{y}) \mid \mathbf{y} \in \mathcal{W}_{h - g}\}$$

$$= \Sigma\{G(\mathbf{x})H(\mathbf{x}^{\downarrow g\cap h}, \mathbf{y}) \mid \mathbf{y}\in\mathcal{W}_{h-g}\}$$
$$= G(\mathbf{x})\,\Sigma\{H(\mathbf{x}^{\downarrow g\cap h}, \mathbf{y}) \mid \mathbf{y}\in\mathcal{W}_{h-g}\}$$
$$= G(\mathbf{x})H^{\downarrow g\cap h}(\mathbf{x}^{\downarrow g\cap h})$$
$$= (GH^{\downarrow g\cap h})(\mathbf{x}).$$

Proof of Proposition 3.3. We have

$$A = A_t\,\Pi\{A_h \mid h\in(\mathcal{H}-\{t\})\},$$

and $\Pi\{A_h \mid h\in(\mathcal{H}-\{t\})\}$ is an array on $\mathcal{X}'$. So by Proposition 3.2,

$$A^{\downarrow\mathcal{X}'} = A_t^{\downarrow t\cap\mathcal{X}'}\,\Pi\{A_h \mid h\in\mathcal{H}-\{t\}\}.$$

This can also be written

$$A^{\downarrow\mathcal{X}'} = A_t^{\downarrow t\cap b}\,\Pi\{A_h \mid h\in\mathcal{H}-\{t\}\}. \tag{3.11}$$

Finally, we can rewrite (3.11) as (3.4).

CHAPTER FOUR

Axioms for Local Computation of Marginals

In the preceding chapter, we learned how to find marginal probabilities using local computations for joint probability distributions that factor on hypertrees. Now we will distill the essential features that make this local computation possible into a small set of axioms.

The present chapter is short because the work was really all done in the preceding chapter. The theory we developed there was based on Propositions 3.1 and 3.2, together with the associativity and commutativity of multiplication. So we can simply think of combination and marginalization as primitive operations and adopt Propositions 3.1 and 3.2, along with the associativity and commutativity of combination, as axioms.

The important point is that these axioms are satisfied in many other computational problems in addition to the problem of marginalizing probabilities. After stating the axioms and sketching how they imply the computational theory of the preceding chapter, we list some of these other problems. We will examine some of them in detail in later chapters.

4.1. The Axiomatic Framework

Variables and Valuations. We begin with a finite nonempty set $\mathcal{X}$ and a nonempty set $\mathcal{V}_h$ for each subset h of $\mathcal{X}$. We call elements of $\mathcal{X}$ *variables*, and we call subsets of $\mathcal{X}$ *domains*. We call elements of $\mathcal{V}_h$ *valuations on h*. We write $\mathcal{V}$ for $\cup\{\mathcal{V}_h | h \subseteq \mathcal{X}\}$, the set of all valuations.

Combination. We assume that $\otimes$ is a binary operation on the set of valuations. In other words, we assume that for any two valuations G and H there is a third valuation, denoted by $G \otimes H$. We call $G \otimes H$ the *combination* of G and H. If G and H are valuations on g and h respectively, then $G \otimes H$ is a valuation on $g \cup h$.

Marginalization. We assume also that for every pair h and g of domains such that $h \subseteq g$, there is a mapping that maps every valuation on g to a valuation on h. We write $G^{\downarrow h}$ for the valuation to which the valuation G on g is mapped. We call $G^{\downarrow h}$ the *marginal of G on h*.

Axioms. We assume that combination and marginalization satisfy these three axioms.

> **Axiom A1** (*Commutativity and associativity of combination*): If G, H, and K are valuations, then $G \otimes H = H \otimes G$ and $G \otimes (H \otimes K) = (G \otimes H) \otimes K$.
>
> **Axiom A2** (*Consonance of marginalization*): If $h_1 \subseteq h_2 \subseteq g$ and G is a valuation on g, then $(G^{\downarrow h_2})^{\downarrow h_1} = G^{\downarrow h_1}$.
>
> **Axiom A3** (*Modularity*): If G and H are valuations on g and h, respectively, then $(G \otimes H)^{\downarrow g} = G \otimes (H^{\downarrow g \cap h})$.

Axiom A1 can also be expressed by saying that the binary operation $\otimes$ is commutative and associative, or by saying that the pair $(\mathcal{V}, \otimes)$ form a commutative semigroup. It follows from Axiom A1 that the result of combining two or more valuations does not depend on the order of combination. So instead of writing an

expression such as $(...((A_{h_1}\otimes A_{h_2})\otimes A_{h_3})\otimes...\otimes A_{h_n})$, which indicates the order of combination, we can write simply $A_{h_1}\otimes A_{h_2}\otimes A_{h_3}\otimes\cdots\otimes A_h$ or $\otimes\{A_{h_i} \mid i=1, ..., n\}$.

Axioms A2 and A3 are essentially identical to Propositions 3.1 and 3.2, respectively, of the preceding chapter.

Factorization. Suppose $\mathcal{H}$ is a hypergraph on $\mathcal{X}$, and suppose

$$A = \otimes\{A_h|h\in\mathcal{H}\},$$

where for each domain h in $\mathcal{H}$, A_h is a valuation on h. Then we say that A, which is a valuation on $\cup\mathcal{H}$, *factors on* $\mathcal{H}$.

4.2. Computational Theory

Now we formulate and prove two propositions that reproduce the basic computational theory of Chapter 3.

The first proposition is essentially identical to Proposition 3.3 of the preceding chapter. It concerns a hypergraph $\mathcal{H}$ on $\mathcal{X}$, a twig t in $\mathcal{H}$, and a branch b for t. We write $\mathcal{H}'$ for $\mathcal{H}$–{t} and $\mathcal{X}'$ for $\cup\mathcal{H}' = \mathcal{X}$–(t–b).

Proposition 4.1. Suppose the valuation A factors on the hypergraph $\mathcal{H}$;

$$A = \otimes\{A_h|h\in\mathcal{H}\},$$

where A_h is a valuation on h. Then

$$A^{\downarrow\mathcal{X}'} = A_b\otimes A_t^{\downarrow t\cap b}\otimes(\otimes\{A_h \mid h\in\mathcal{H}\text{–}\{t, b\}\}). \tag{4.1}$$

Formula (4.1) says that the marginal $A^{\downarrow\mathcal{X}'}$ factors on $\mathcal{H}'$. On b we have the factor $A_b\otimes A_t^{\downarrow t\cap b}$, and on each other h in $\mathcal{H}'$, we have the original factor A_h.

The computational significance of Proposition 4.1 is the same as the computational significance of Proposition 3.3. Whenever a hypergraph as a twig, we can reduce a factorization on that hypergraph to the smaller hypergraph without

the twig, using only computations involving the twig and its branch. We do not need to work with all the variables in the hypergraph.

The proof of Proposition 4.1 is the same as the proof of Proposition 3.3, except that this time the steps are justified by our axioms. Notice that we use only Axioms A1 and A3.

Proof of Proposition 4.1. By Axiom A1, we can write

$$A = (\otimes\{A_h \mid h\in\mathcal{H}'\}) \otimes A_t.$$

Applying Axiom A3 to this expression, we obtain

$$A^{\downarrow\mathcal{X}'} = (\otimes\{A_h \mid h\in\mathcal{H}'\}) \otimes A_t^{\downarrow t\cap\mathcal{X}'}.$$

Since b is a branch for t, $t\cap\mathcal{X}' = t\cap b$. So

$$A^{\downarrow\mathcal{X}'} = (\otimes\{A_h \mid h\in\mathcal{H}'\}) \otimes A_t^{\downarrow t\cap b}.$$

By Axiom A1, we can rearrange the order of the factors in this expression to obtain (4.1). **End of Proof.**

Axiom A2 becomes relevant when $\mathcal{H}$ is a hypertree, and we want to apply Proposition (4.1) repeatedly in order to find A's marginal on a hyperedge. We dealt with this case informally in Chapter 3. This time around, we will state the result more formally. This requires considerable notation.

First, suppose $\mathcal{H}$ is a hypertree, suppose $h_1h_2...h_n$ is a hypertree construction sequence for $\mathcal{H}$, and suppose β is a branching for $h_1h_2...h_n$. Let $\mathcal{X}_k$ denote $h_1\cup...\cup h_k$, and let $\mathcal{H}_k$ denote $\{h_1, h_2, ..., h_k\}$, for k=1, ..., n.

Next, suppose $\{A_h|h\in\mathcal{H}\}$ is a collection of valuations; A_h is a valuation on h for each h. For each k between 1 and n, inclusive, we define a collection of valuations $\{A_{h,k}|h\in\mathcal{H}_k\}$, where $A_{h,k}$ is a valuation on h. We do so recursively, working backwards from n. We set $A_{h,n} = A_h$, so that $\{A_{h,n}|h\in\mathcal{H}_n\}$ is the same as $\{A_h|h\in\mathcal{H}\}$. We then define $\{A_{h,n-1}|h\in\mathcal{H}_{n-1}\}$ by setting

$$A_{b(h_n),n-1} = A_{b(h_n),n}\otimes A_{h_n,n}^{\downarrow h_n\cap b(h_n)}$$

and $A_{h,n-1} = A_{h,n}$ for all other h in $\mathcal{H}_{n-1}$. And for k from n-1 down to 2, we similarly and successively define $\{A_{h,k-1}|h\in\mathcal{H}_{k-1}\}$ in terms of $\{A_{h,k}|h\in\mathcal{H}_k\}$ by setting

$$A_{b(h_k),k-1} = A_{b(h_k),k}\otimes A_{h_k,k}^{\downarrow h_k\cap b(h_k)}$$

and $A_{h,k-1} = A_{h,k}$ for all other h in $\mathcal{H}_{k-1}$.

Notice that we can compute the collections $\{A_{h,k}|h\in\mathcal{H}_k\}$ step by step, using only computations involving the twigs and their branches. The step from $\{A_{h,k}|h\in\mathcal{H}_k\}$ to $\{A_{h,k-1}|h\in\mathcal{H}_{k-1}\}$ involves only a marginalization from h_k to $h_k\cap b(h_k)$ and then a combination on $b(h_k)$.

Now we can state our conclusion formally:

Proposition 4.2. If $A = \otimes\{A_h|h\in\mathcal{H}\}$, then

$$A^{\downarrow\chi_k} = \otimes\{A_{h,k}|h\in\mathcal{H}_k\} \tag{4.2}$$

for k=1, ...,n-1.

Since $\mathcal{H}_k=\{h_1\}$, $\chi_1=h_1$, and $\{A_{h_1,1}\}$, (4.2) reduces to

$$A^{\downarrow h_1} = A_{h_1,1} \tag{4.3}$$

when k=1. Thus the step-by-step computations on the twigs and their branches enable us to find A's marginal on h_1. Since we can find a hypertree construction sequence beginning with an arbitrary hyperedge, this means we can find the marginal on an arbitrary hyperedge using local computations.

Proof of Proposition 4.2. By Proposition 4.1,

$$\otimes\{A_{h,r}|h\in\mathcal{H}_r\} = (\otimes\{A_{h,r+1}|h\in\mathcal{H}_{r+1}\})^{\downarrow\chi_r}$$

for r=k, ...,n-1. So

$$\begin{aligned}\otimes\{A_{h,k}|h\in\mathcal{H}_k\} &= (\otimes\{A_{h,k+1}|h\in\mathcal{H}_{k+1}\})^{\downarrow\chi_k}\\ &= ((\otimes\{A_{h,k+2}|h\in\mathcal{H}_{k+2}\})^{\downarrow\chi_{k+1}})^{\downarrow\chi_k}\\ &\;\;\vdots\\ &= ((\ldots((\otimes\{A_{h,n}|h\in\mathcal{H}_n\})^{\downarrow\chi_{n-1}})^{\downarrow\chi_{n-2}}\ldots)^{\downarrow\chi_{k+1}})^{\downarrow\chi_k}.\\ &= ((\ldots(A^{\downarrow\chi_{n-1}})^{\downarrow\chi_{n-2}}\ldots)^{\downarrow\chi_{k+1}})^{\downarrow\chi_k}.\end{aligned}$$

By Axiom A2, the last expression is equal to $A^{\downarrow\chi_k}$. **End of Proof.**

Propositions 4.1 and 4.2 simply repeat, in the general setting of our axioms, what we learned about probability in section 3.4 of the preceding chapter. Everything that we said about computation in Markov trees in sections 3.5 and 3.6 also carries over to this general setting.

4.3. Instances of the Axioms

In chapter 3, we satisfied Axioms A1, A2, and A3 by assigning a finite frame to each variable, taking valuations to be real-valued functions on the frames of for sets of variables, taking combination to be multiplication, and taking marginalization to be marginalization in the usual probability sense. Here is list of other examples. We will study some of these examples in greater detail in later chapters.

Finding marginals is of some interest in all these examples, but in most of the examples the main goal is to find what we call, in general, solution configurations. They, too, can be found by local computation when a valuation factors on a hypertree. In chapter 7, we will show how the axiomatic framework of this chapter can be extended to an axiomatic framework that accounts for such local computation of solution configurations.

Belief Functions. In the next chapter, we will show Axioms A1, A2, and A3 are satisfied if we take valuations to be belief functions, combine them by Dempster's rule of combination, and marginalize them in the standard way.

We would like to remark here that we first understood local computation, and first isolated the axioms, in our study of the belief function case. This work is reported in Shenoy and Shafer [1986], Shenoy, Shafer and Mellouli [1986], and Shafer, Shenoy and Mellouli [1987]. We have presented the probability case first in this book only because we expect it to be more familiar and transparent to most of our readers.

Constraint Satisfaction.

Discrete Optimization.

Sparse Linear Equations.

CHAPTER FIVE

Belief-Function Propagation

In this chapter, we study the problem of propagating belief functions using local computations. This problem has been examined previously by Shafer and Logan [1987], Shenoy and Shafer [1986], Shenoy, Shafer and Mellouli [1986], Kong [1986], Dempster and Kong [1986], Shafer, Shenoy and Mellouli [1987], and Mellouli [1987].

In the case of belief functions, the idea of factoring a single probability distribution into potentials is replaced by the idea of decomposing evidence into independent items of evidence—items that involve independent uncertainties. Each item of evidence is represented by a belief function that bears on a few variables, and the belief functions are combined by Dempster's rule. The result is a belief function representing the total evidence on all the variables.

We begin by defining belief functions, basic probability assignment functions, plausibility functions, and commonality functions. All of these functions contain the same information and they can all be defined mathematically in terms of a random non-empty subset.

Next, we introduce the ideas of projecting a subset from one frame to a subset of a smaller frame and vacuously extending a subset from one frame to a subset of a larger frame.

Using projection of subsets, we define marginals of belief functions.

The combination operator for belief functions, the operator that plays the role of multiplication for potentials, is Dempster's rule of combination.

Mathematically, the rule corresponds to finding the distribution of the intersection of independent random non-empty subsets, conditional on this intersection being non-empty. Intuitively, combination of belief functions by this rule corresponds to pooling the evidence on which the belief functions is based. The combination results in a belief function that is supposed to represent the pooled evidence. We give two descriptions of Dempster's rule, one in terms of basic probability assignments, and one in terms of commonality functions.

We use the axiomatic framework of chapter 4 to demonstrate that if the belief functions being combined by Dempster's rule bear on separate hyperedges of a hypertree of variables, marginals of the belief function resulting from the combination can be obtained using local computation.

We conclude by discussing implementation of the belief-function propagation algorithm.

5.1. Basic Definitions

The definitions we give here are purely mathematical; we define belief functions in terms of random non-empty subsets. We should caution the reader, however, that the idea of a random subset does not provide an appropriate intuitive basis for the interpretation of belief functions as assessments of evidence. For information on the interpretation of belief functions, see Shafer [1976, 1987].

Random Non-Empty Subset. Suppose $\mathcal{W}_X$ is the frame for a variable X, its set of possible values. A *random subset* $\mathcal{S}$ *of* $\mathcal{W}_X$ is defined by giving a probability measure on the set of all subsets of $\mathcal{W}_X$. In other words, we assign to the subsets of $\mathcal{W}_X$ non-negative numbers adding to one. We write $\Pr[\mathcal{S}=A]$ for the non-negative number assigned to the subset A of $\mathcal{W}_X$, and we call $\Pr[\mathcal{S}=A]$ the probability that $\mathcal{S}$ is equal to A. If $\Pr[\mathcal{S}=\varnothing] = 0$, then we say that the random subset $\mathcal{S}$ is *non-empty*.

Belief Function. A function Bel that assigns a degree of belief Bel(A) to every subset A of $\mathcal{W}_X$ is called a *belief function* on X if there exists a random non-empty subset $\mathcal{S}$ of $\mathcal{W}_X$ such that Bel is given by

$$\text{Bel}(A) = \Pr[\mathcal{S} \subseteq A]$$

for every subset A of $\mathcal{W}_X$. Intuitively, the number Bel(A) is the degree to which we judge given evidence to support the proposition that the true value of X is in A, or the degree to which we think it reasonable to believe this proposition on the basis of that evidence alone.

A subset A of $\mathcal{W}_X$ is called a *focal element* of Bel if $\Pr[\mathcal{S}=A]$ is positive.

The simplest belief function on X is the one corresponding to the random subset that is equal to the whole set $\mathcal{W}_X$ with probability one. We call this belief function *the vacuous belief function on X*. The set $\mathcal{W}_X$ itself is its only focal element**.** The vacuous belief function on X is appropriate for representing the opinion that given evidence is irrelevant to X.

Basic Probability Assignment Function. The information contained in a belief function can be expressed in several different ways. One way is in terms of the *basic probability assignment function* m, defined by

$$m(A) = \Pr[\mathcal{S}=A]$$

for every subset A of $\mathcal{W}_X$. Since $\mathcal{S}$ is non-empty, $m(\varnothing)=0$, and since $\mathcal{W}_X$ is finite,

$$\sum\{m(A) \mid A \subseteq \mathcal{W}_X\} = 1.$$

Intuitively, m(A) measures the belief that is committed exactly to A (and to nothing smaller or larger). We can express Bel in terms of m as follows:

$$\begin{aligned} \text{Bel}(A) &= \Pr[\mathcal{S} \subseteq A] \\ &= \sum\{\Pr[\mathcal{S}=B] \mid B \subseteq A\} \\ &= \sum\{m(B) \mid B \subseteq A\}. \end{aligned}$$

It is shown in Shafer [1976, Ch. 2] that we can also obtain m from Bel:

$$m(A) = \sum\{(-1)^{|A-B|}\,\text{Bel}(B) \mid B \subseteq A\},$$

where $|A-B|$ denotes the number of elements in the set $A-B$.

Plausibility Function. Another way of expressing the information contained in a belief function Bel is in terms of the *plausibility function* Pl, which is given by

$$\mathrm{Pl}(A) = 1 - \mathrm{Bel}(\mathcal{W}_X - A) = \Pr[\mathcal{S} \cap A \neq \varnothing]$$

for every subset A of $\mathcal{W}_X$. Intuitively, Pl(A) measures the extent to which given evidence fails to refute A. To recover Bel from Pl, we use the relation

$$\mathrm{Bel}(A) = 1 - \mathrm{Pl}(\mathcal{W}_X - A).$$

Notice that $\mathrm{Bel}(A) \leq \mathrm{Pl}(A)$ for every subset A of $\mathcal{W}_X$. Both Bel and Pl are monotone: $\mathrm{Bel}(A) \leq \mathrm{Bel}(B)$ and $\mathrm{Pl}(A) \leq \mathrm{Pl}(B)$ whenever $A \subseteq B$.

Commonality Function. Finally, the information in Bel or m or Pl is also contained in the *commonality function* Q, defined by

$$Q(A) = \Pr[\mathcal{S} \supseteq A] = \sum\{m(B) \mid B \supseteq A\}$$

for every subset A of $\mathcal{W}_X$. The following proposition tells us that we can recover $\mathcal{S}$ or m from Q and also states another property of commonality functions.

Proposition 5.1. Let Q and m be commonality function and basic probability assignment function corresponding to $\mathcal{S}$. Then

$$m(A) = \Pr[\mathcal{S} = A] = \Sigma\{(-1)^{|B-A|}\, Q(B) \mid B \supseteq A\}$$

for all subsets A of $\mathcal{W}_X$, and

$$\Sigma\{(-1)^{|A|+1}\, Q(A) \mid \varnothing \neq A \subseteq \mathcal{W}_X\} = 1. \tag{5.1}$$

It is shown in Shafer [1976, Ch.2] that

$$Q(A) = \sum\{(-1)^{|B|+1}\, \mathrm{Pl}(B) \mid \varnothing \neq B \subseteq A\}, \tag{5.2}$$

and

$$\mathrm{Pl}(A) = \sum\{(-1)^{|B|+1} Q(B) \mid \varnothing \neq B \subseteq A\}$$

for every non-empty subset A of $\mathcal{W}_X$. We do not need formulas for the empty set, since $Q(\varnothing) = 1$ and $\mathrm{Pl}(\varnothing) = 0$ for any belief function. Notice also that if the set A contains only a single element, then (5.2) reduces to $Q(A) = \mathrm{Pl}(A)$.

Our definitions generalize straightforwardly to sets of variables. A *random subset $\mathcal{S}$ of $\mathcal{W}_h$*, where h is a finite set of variables, is defined by giving a probability measure on the set of all subsets of $\mathcal{W}_h$. We call Bel a *belief function* on h if there exists a random non-empty subset $\mathcal{S}$ of $\mathcal{W}_h$ such that $\mathrm{Bel}(A) = \Pr[\mathcal{S} \subseteq A]$

for every subset A of $\mathcal{W}_h$. We call m a *basic probability assignment function* for h if there exists a random non-empty subset S of $\mathcal{W}_h$ such that $m(A) = Pr[S=A]$ for every subset A of $\mathcal{W}_h$. We call Pl a *plausibility function* for h if there exists a random non-empty subset S of $\mathcal{W}_h$ such that $Pl(A) = Pr[S \cap A \neq \varnothing]$ for every subset A of $\mathcal{W}_h$. And finally, we call Q a *commonality function* for h if there exists a random non-empty subset S of $\mathcal{W}_h$ such that $Q(A) = Pr[S \supseteq A]$ for every subset A of $\mathcal{W}_h$.

In chapter 3, we established the convention that $\mathcal{W}_\varnothing$, the frame for the empty set $\varnothing$, consists of a single element, ♦; $\mathcal{W}_\varnothing = \{♦\}$. This implies that there is only one random non-empty subset of $\mathcal{W}_\varnothing$; it is equal to the whole frame $\mathcal{W}_\varnothing$ with probability one. And hence there is only one belief function on $\varnothing$; it has the values $Bel(\varnothing)=0$, and $Bel(\mathcal{W}_\varnothing)=1$. The corresponding values for m, Pl, and Q are $m(\varnothing)=0$ and $m(\mathcal{W}_\varnothing)=1$, $Pl(\varnothing)=0$ and $Pl(\mathcal{W}_\varnothing)=1$, and $Q(\varnothing)=1$ and $Q(\mathcal{W}_\varnothing)=1$.

5.2. Projection and Vacuous Extension of Subsets

In this section, we define projection of a subset of one frame to a subset of a smaller frame and vacuous extension of a subset of a frame to a subset of a larger frame.

Projection. If A is subset of $\mathcal{W}_{\{W,X,Y,Z\}}$, for example, then the marginal of A to a subset of $\mathcal{W}_{\{W,X\}}$ consists of the elements of $\mathcal{W}_{\{W,X\}}$ which can be obtained by projecting elements of A to $\mathcal{W}_{\{W,X\}}$.

If g and h are sets of variables, $h \subseteq g$, and A_g is a non-empty subset of $\mathcal{W}_g$, then the *projection* of A_g to $\mathcal{W}_h$, denoted by $A_g^{\downarrow h}$, is given by

$$A_g^{\downarrow h} = \{\mathbf{x}^{\downarrow h} \mid \mathbf{x} \in A_g\}$$

We will adopt the convention that the projection of the empty subset is the empty subset.

The projection of a non-empty subset is always non-empty. If $h=\varnothing$ and $A_g \neq \varnothing$, then $A_g^{\downarrow h} = \{♦\}$. Note that the definition implies that $A_g^{\downarrow g} = A_g$.

Vacuous Extension. By vacuous extension of a subset of a frame to a subset of a larger frame, we mean a cylinder set extension. If A is a subset of $\mathcal{W}_{\{W,X\}}$, for example, then the vacuous extension of A to $\mathcal{W}_{\{W,X,Y,Z\}}$ is $A \times \mathcal{W}_{\{Y,Z\}}$.

If g and h are sets of variables, $g \subseteq h$, $g \neq h$, and A_g is a subset of $\mathcal{W}_g$, then the *vacuous extension* of A_g to $\mathcal{W}_h$ is $A_g \times \mathcal{W}_{h-g}$. If A_g is a subset of $\mathcal{W}_g$, then the vacuous extension of A_g to $\mathcal{W}_g$ is defined to be A_g. We will let $A_g^{\uparrow h}$ denote the vacuous extension of A_g to $\mathcal{W}_h$.

We shall now state some results regarding this operation on subsets.

Lemma 5.1. Suppose A_g is a subset of $\mathcal{W}_g$ and suppose $h_1 \subseteq h_2 \subseteq g$. Then

$$A_g^{\downarrow h_1} = (A_g^{\downarrow h_2})^{\downarrow h_1}.$$

Lemma 5.2. Suppose A_g and B_h are subsets of $\mathcal{W}_g$ and $\mathcal{W}_h$ respectively. Then

$$(A_g^{\uparrow g \cup h} \cap B_h^{\uparrow g \cup h})^{\downarrow g} = A_g \cap (B_h^{\downarrow h \cap g}).$$

5.3. Dempster's Rule of Combination

Dempster's rule of combination is a rule for forming a new belief function from two or more belief functions. Consider two random non-empty subsets $\mathcal{S}_g$ and $\mathcal{S}_h$ of $\mathcal{W}_g$ and $\mathcal{W}_h$ respectively. Suppose $\mathcal{S}_g$ and $\mathcal{S}_h$ are probabilistically independent, i.e.,

$$\Pr[\mathcal{S}_g = A_g \text{ and } \mathcal{S}_h = A_h] = \Pr[\mathcal{S}_g = A_g] \Pr[\mathcal{S}_h = A_h]$$

for all subsets A_g of $\mathcal{W}_g$ and A_h of $\mathcal{W}_h$. Suppose also that $\Pr[\mathcal{S}_g^{\uparrow(g \cup h)} \cap \mathcal{S}_h^{\uparrow(g \cup h)} \neq \varnothing] > 0$. Let $\mathcal{S}$ be the random non-empty subset that has the probability distribution of $\mathcal{S}_g^{\uparrow(g \cup h)} \cap \mathcal{S}_h^{\uparrow(g \cup h)}$ conditional on $\mathcal{S}_g^{\uparrow(g \cup h)} \cap \mathcal{S}_h^{\uparrow(g \cup h)} \neq \varnothing$, i.e.,

$$\Pr[\mathcal{S} = A] = \Pr[\mathcal{S}_g^{\uparrow(g \cup h)} \cap \mathcal{S}_h^{\uparrow(g \cup h)} = A] \,/\, \Pr[\mathcal{S}_g^{\uparrow(g \cup h)} \cap \mathcal{S}_h^{\uparrow(g \cup h)} \neq \varnothing]$$

for every non-empty subset A of $\mathcal{W}_{g \cup h}$. If Bel_1 and Bel_2 are belief functions for g and h corresponding to $\mathcal{S}_g$ and $\mathcal{S}_h$ respectively, then we call the belief function for

$g \cup h$ corresponding to S the *orthogonal sum* of Bel_1 and Bel_2. The orthogonal sum of Bel_1 and Bel_2 is denoted by $Bel_1 \oplus Bel_2$. The rule for forming $Bel_1 \oplus Bel_2$ is called *Dempster's rule of combination*. If the bodies of evidence on which Bel_1 and Bel_2 are based are independent, then $Bel_1 \oplus Bel_2$ is supposed to represent the result of pooling these two bodies of evidence.

It is obvious from the definitions that the operation $\oplus$ has the following properties:

Existence: $Bel_1 \oplus Bel_2$ exists unless there is a subset A of W_h such that $Bel_1(A)=1$ and $Bel_2(W_h - A)=1$.

Commutativity: $Bel_1 \oplus Bel_2 = Bel_2 \oplus Bel_1$.

Associativity: $(Bel_1 \oplus Bel_2) \oplus Bel_3 = Bel_1 \oplus (Bel_2 \oplus Bel_3)$.

In general, $Bel \oplus Bel \neq Bel$. The belief function $Bel \oplus Bel$ will favor the same subsets as Bel, but it will do so with twice the weight of evidence, as it were.

Vacuousness: If Bel_1 is vacuous, then $Bel_1 \oplus Bel_2 = Bel_2$.

Note that Dempster's rule of combination as defined above involves vacuously extending the random non-empty subsets S_g and S_h to subsets of a common frame ($W_{g \cup h}$) and then intersecting the two random non-empty subsets.

Dempster's rule can be expressed in terms of the probability mass assignment function. We will do so in two stages. First we will describe the vacuous extension of a basic probability assignment function. Next we will define the combination of two basic probability assignment functions on a common set of variables.

Proposition 5.2. Suppose that m is a basic probability assignment function for g. Suppose that $h \supseteq g$. Then the vacuous extension of m to h is given as follows:

$$m^{\uparrow h}(A) = \begin{cases} m(B) & \text{if } A = B^{\uparrow h} \text{ for some } B \subseteq \mathcal{W}_g \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Suppose that Bel_1 and Bel_2 are two belief functions on g corresponding to random non-empty subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. Let the basic probability assignment functions for Bel_1, Bel_2 and $Bel_1 \oplus Bel_2$ be denoted by m_1, m_2 and m, respectively. Then for any non-empty subset A of $\mathcal{W}_g$, we have

$$\begin{aligned} m(A) &= \Pr[\mathcal{S}=A] = \Pr[\mathcal{S}_1 \cap \mathcal{S}_2 = A] \,/\, \Pr[\mathcal{S}_1 \cap \mathcal{S}_2 \neq \varnothing] \\ &= \sum\{m_1(B)\, m_2(C) \mid B \cap C = A\} \,/\, \sum\{m_1(B) m_2(C) \mid B \cap C \neq \varnothing\} \\ &= K^{-1} \sum\{m_1(B_1)\, m_2(B_2) \mid B_1 \cap B_2 = A\} \end{aligned} \tag{5.4}$$

where K^{-1} is a normalizing constant given by

$$\begin{aligned} K &= \sum\{m_1(B_1) m_2(B_2) \mid B_1 \cap B_2 \neq \varnothing\} \\ &= \sum\Big\{\sum\{m_1(B_1) m_2(B_2) \mid B_1 \cap B_2 = A\} \;\Big|\; \varnothing \neq A \subseteq \mathcal{W}_h\Big\} \\ &= \sum\{m(A) \mid \varnothing \neq A \subseteq \mathcal{W}_h\} \end{aligned} \tag{5.5}$$

where K does not depend on A.

Dempster's rule can also be described in terms of commonality functions. First let us describe vacuous extension of commonality functions.

Proposition 5.4. Suppose Q is a commonality function for g. Suppose that $h \supseteq g$. Then the vacuous extension of Q to h is given as follows:

$$Q^{\uparrow h}(A) = Q(A^{\downarrow g}) \tag{5.6}$$

for all subsets A of $\mathcal{W}_h$.

Suppose that Bel_1 and Bel_2 are two belief functions on g corresponding to random non-empty subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. If the commonality functions for Bel_1, Bel_2, and $Bel_1 \oplus Bel_2$ are denoted by Q_1, Q_2, and Q, respectively, then

$$\begin{aligned} Q(A) &= \Pr[\mathcal{S} \supseteq A] \\ &= K^{-1} \Pr[\mathcal{S}_1 \cap \mathcal{S}_2 \supseteq A] \\ &= K^{-1} \Pr[\mathcal{S}_1 \supseteq A \text{ and } \mathcal{S}_2 \supseteq A] \\ &= K^{-1} \Pr[\mathcal{S}_1 \supseteq A] \Pr[\mathcal{S}_2 \supseteq A] \\ &= K^{-1} Q_1(A) Q_2(A), \end{aligned} \tag{5.7}$$

where $K = \Pr[\mathcal{S}_1 \cap \mathcal{S}_2 \neq \varnothing]$ does not depend on A.

Substituting (5.5) in (5.1) results in an expression for K:

$$K = \sum\{(-1)^{|A|+1} Q_1(A) Q_2(A) \mid \varnothing \neq A \subseteq \mathcal{W}_h\}. \tag{5.8}$$

Implementing Dempster's rule for belief functions on h is computationally expensive when $\mathcal{W}_h$ is large. Whether the rule is implemented using basic probability assignment functions or commonality functions, the number of terms in (5.5) or (5.8) involves a term for every non-empty subset A of $\mathcal{W}_h$, and the number of these subsets increases exponentially with the size of $\mathcal{W}_h$. This means that we face a computation of exponential complexity even if we are trying to find the value of the orthogonal sum $Bel_1 \oplus Bel_2$ only for a single subset A of $\mathcal{W}_h$.

This computational complexity seems to be intrinsic to Dempster's rule. It is possible in some cases to exploit special structure in the belief functions being combined in order to reduce the complexity [Barnett, 1981]. But there does not seem to be any general way of implementing the rule that will always involve fewer computations than are involved in (5.4) and (5.5), or (5.7) and (5.8).

5.4. Marginalization for Belief Functions

In this section, we first introduce the idea of marginalizing a belief function from one set of variables to a smaller set of variables.

Suppose that Bel is a belief function for g corresponding to random non-empty subset $\mathcal{S}_g$ of $\mathcal{W}_g$ and suppose $h \subseteq g$. The *marginal* of Bel to h, denoted by $Bel^{\downarrow h}$, is the belief function corresponding to the random non-empty subset $\mathcal{S}_g^{\downarrow h}$.

We are using standard probability notation here. The random non-empty subset $\mathcal{S}_g^{\downarrow h}$ is a "function" of the random non-empty subset $\mathcal{S}_g$ in the sense that

whenever S_g=A, $S_g^{\downarrow h}$=$A^{\downarrow h}$. Thus $S_g^{\downarrow h}$ is a well-defined random non-empty subset of W_h. The following proposition gives an explicit description of $S_g^{\downarrow h}$ in terms of S_g.

Lemma 5.1. Suppose that S_g is a random non-empty subset of W_g. Then $S_g^{\downarrow h}$ is the random non-empty subset of W_h given by

$$\Pr[S_g^{\downarrow h}=A] = \Sigma\{\Pr[S_g=B] \mid B\subseteq W_g \text{ such that } B^{\downarrow h}=A\}$$

for all subsets A of W_h.

The marginalization of basic probability assignment function, plausibility function and commonality function are defined likewise. Suppose that m, Pl and Q represent basic probability assignment function, plausibility function and commonality function, respectively, for g with random non-empty subset S_g. The *marginal* of m, Pl, and Q to h, denoted by $m^{\downarrow h}$, $Pl^{\downarrow h}$, and $Q^{\downarrow h}$, respectively, is the basic probability assignment function, plausibility function and commonality function, respectively, corresponding to random non-empty subset $S_g^{\downarrow h}$.

Proposition 5.2. Suppose that m and Q are basic probability assignment function and commonality function, respectively, for g. Suppose that h⊆g. Then the marginal of m and Q for h are given as follows:

$$m^{\downarrow h}(A) = \Sigma\{m(B) \mid B\subseteq W_g \text{ such that } B^{\downarrow h}=A\} \tag{5.9}$$

and

$$Q^{\downarrow h}(A) = \Sigma\{(-1)^{|B|-|A|} Q(B) \mid B\subseteq W_g \text{ such that } B^{\downarrow h}=A\} \tag{5.10}$$

for all subsets A of W_h.

5.5. Local Computation for Belief Functions

In chapter 4 we saw that in order to compute the marginal of a valuation (that factors on a hypertree) using local computations, it is necessary for the projection

and combination operation to satisfy certain axioms. In this chapter, valuations on h correspond to belief functions on h, and combination and marginalization operators correspond to Dempster's rule of combination and marginalization of belief functions. We have already shown that Dempster's rule of combination satisfies Axiom A1. The following two theorems assert that Axioms A2 and A3 are also satisfied by these two operations.

Theorem 5.1. Suppose Bel_g is a belief function for g and suppose $h_1 \subseteq h_2 \subseteq g$. Then

$$Bel_g^{\downarrow h_1} = (Bel_g^{\downarrow h_2})^{\downarrow h_1}$$

Theorem 5.2. Suppose Bel_g and Bel_h are belief function for g and h respectively. Then

$$(Bel_g^{\uparrow g\cup h} \oplus Bel_h^{\uparrow g\cup h})^{\downarrow g} = Bel_g \oplus (Bel_h^{\downarrow h\cap g})$$

5.6. Implementation Issues

Since the combination and marginalization operations for belief functions satisfy Axioms A1 to A3, we can compute the marginals using a scheme similar to that presented chapter 3 in great detail for probabilities and repeated in chapter 4. Here we will not repeat the scheme for belief functions. Instead we will make a few observations regarding implementation of the scheme.

The most natural implementation of the belief function propagation scheme is using basic probability assignment functions. Belief functions are most easily assessed in terms of basic probability assignment functions. The vacuous extension of basic probability assignment functions, given in (5.3), is a simple operation involving only a change of the focal elements to their cylinder extension. The marginalization of basic probability assignment functions, given in (5.5), while involving more computations than vacuous extension, is also fairly inexpensive. Combining basic probability assignment functions using Dempster's rule, given in (5.4) and (5.5), involves the most computational expense of the three operations. It

should be noted here that similar to potentials, we can avoid the renormalization of basic probability assignment functions until the very end.

Another implementation of the belief function propagation scheme is using commonality functions. Commonality functions have no intuitive interpretations. Hence, we may have to translate assessed belief functions from basic probability assignment functions to commonality functions. The vacuous extension of commonality functions, given in (5.4), involves a little more than changing the names of the focal elements. Marginalization of commonality functions, given in (5.6), is computationally expensive. However, this is offset by the fact that combining commonality functions using Dempster's rule is simply pointwise multiplication (where the points correspond to subsets). Again, we can avoid renormalization, given in (5.10), until the very end. However, to report the results, we will have to translate the commonality functions back to basic probability assignment functions.

5.7. Proofs

CHAPTER SIX

Conditional Probability

One of the themes of this book is that conditionals need not be the primitives of a language for the assessment of evidence. In chapters 2 and 3, we showed that the language of conditionals is not necessary to the local computation of probabilities. In chapter 4, we presented belief functions as one mode of evidential assessment that does not rely on conditionals. Having made these negative points, we now need to round out the picture by looking at conditional probability and exploring the roles it can play.

As we mentioned in the introduction, conditional probability plays an important role in probabilistic modeling, especially when causal ideas are involved. We will not explore this point in detail here, but we will discuss how conditional independence is involved in probabilistic modeling, and how such modeling yields factorizations that can be exploited by the methods of chapter 3.

Conditional probability also plays a role in the algorithmsss for local computation studied by Kelly and Barclay [1973], Pearl [1986], and Lauritzen and Spiegelhalter [1988]. Here we will explain how conditional probability is involved in the motivation for these authors' algorithms, and how their algorithms are related to the algorithms we learned in chapter 3. As we will see, Pearl's algorithm is essentially the algorithm of section 3.6 applied to a factorization of a special form, while Lauritzen and Spiegelhalter's algorithm differs only slightly from the algorithm of section 3.6.

We begin, in section 6.1, with a theoretical study of conditional probability. Here we translate standard ideas about conditional probability into a terminology

and a notation compatible with the terminology and notation introduced in chapter 3. In section 6.2, we apply what we have learned to the problem of conditioning a factored probability distribution on new evidence. In section 6.3, we take up the role of conditional independence in causal modeling, and we define probability trees. In section 6.4, we review the computational role of Bayes's theorem in statistical inference, and we show that the algorithm of section 3.6, applied to probability trees, results in the generalization of Bayes's theorem developed by Kelly and Barclay and Pearl. In section 6.5, we take up the algorithm for local computation studied by Lauritzen and Spiegelhalter. Finally, in section 6.6, we give proofs of the displayed propositions.

6.1. The Theory of Conditional Probability

In this section, we study conditional probability, conditional independence, and Markov probability distributions. We develop for these topics a notation and terminology consistent with the notation and terminology we have already developed for unconditional probability distributions on sets of variables.

A conditional probability is a ratio of probabilities, and it may be considered ill-defined if the probability in its denominator is zero. Hence the possibility of zero probabilities can make discussions of conditional probability awkward. We will seek to minimize this awkwardness.

We do want to allow zero probabilities. At first glance, it might seem reasonable to prohibit them in a theoretical discussion. Perhaps every event should be allowed at least some tiny probability, on the grounds of our own fallibility [Pearl 1986]. But relationships among variables often make certain combinations of values impossible, and clarity of thought requires that we be able to represent this impossibility by giving the combinations probability zero [Lauritzen and Spiegelhalter 1988].

We do two things to deal with the problem of zero probabilities. First, we adopt the convention that division by zero yields zero. Second, we emphasize ratios of entire probability distributions (or, more generally, potentials) rather than

individual probabilities. It turns out that the probability distributions in whose ratios we are interested are usually of a special type; the denominator is a marginal of the numerator. This, as we will see, implies that the ratios can be handled, in many respects, as if the denominators were never zero.

Zero probabilities also complicate the idea of probabilistic independence. When zero probabilities are not allowed, independence and conditional independence can be defined simply in terms of factorization. But when zero probabilities are allowed, factorization is not enough to imply conditional independence. As it turns out, this is not a problem for us; the conditional independence relations with which we are concerned are relations among hyperedges in a hypertree, and factorization on the hypertree is enough to guarantee these particular conditional independence relations. But for the sake of completeness, we investigate what conditional independence adds, in general, to factorization.

We base our formal definition of conditional probability on two general ideas, the idea of a ratio potential and the idea of an indicator potential. A ratio potential is the ratio of a potential to one of its marginals. An indicator potential is a potential that takes only the values zero and one.

We begin with some general comments about division for potentials. Then we introduce ratio potentials and indicator potentials, and we define conditional probability. Then we study independence and the implications for probability distributions of factorization on hypertrees.

Mathematically, we are working in the framework established in chapter 3. Our potentials are real-valued functions on the Cartesian products of the finite frames of a finite set of variables $\mathcal{X}$.

The Division of Potentials. Division, like multiplication, will be pointwise; if A and B are potentials on h, then the quotient A/B is the array on h given by $(A/B)(\mathbf{x})=A(\mathbf{x})/B(\mathbf{x})$. By our convention that division by zero yields zero, $(A/B)(\mathbf{x})=0$ whenever $B(\mathbf{x})=0$. If there is a configuration $\mathbf{x}$ of h such that both $A(\mathbf{x})$ and $B(\mathbf{x})$ are non-zero, then A/B will be a potential. Otherwise it will be an array identically equal to zero.

If G is a potential on g, and H is a potential on h, then G/H is a potential on $g \cup h$ defined by $(G/H)(\mathbf{x}) = G(\mathbf{x}^{\downarrow g})/H(\mathbf{x}^{\downarrow h})$.

The following proposition sets out what we need to know about the division of potentials when the denominator is a marginal of the numerator.

Proposition 6.1. If G is a potential on g, and $h \subseteq g$, then the following statements are all true.

(i) $G(\mathbf{x}) \leq G^{\downarrow h}(\mathbf{x}^{\downarrow h})$ for all $\mathbf{x} \in \mathcal{W}_g$.

(ii) If $\mathbf{x} \in \mathcal{W}_g$ and $G^{\downarrow h}(\mathbf{x}^{\downarrow h})=0$, then $G(\mathbf{x})=0$.

(iii) $(G/G^{\downarrow h})(\mathbf{x})=0$ if and only if $G(\mathbf{x})=0$.

(iv) The array $G/G^{\downarrow h}$ is a potential.

(v) If $G/G^{\downarrow h} = B$, then $G = BG^{\downarrow h}$.

(vi) If $GG^{\downarrow h} = B$, then $G = B/G^{\downarrow h}$.

Because of the consonance of marginalization, the fact that $A^{\downarrow h_1}=(A^{\downarrow h_2})^{\downarrow h_1}$ if $h_1 \subseteq h_2$, all the statements in Proposition 6.1 generalize to the case where $G^{\downarrow h}$ is compared not to G but to some lesser marginalization of G. Statement (ii), for example, generalizes to "If $G^{\downarrow h}(\mathbf{x}^{\downarrow h})=0$ and $h \subseteq f$, then $G^{\downarrow f}(\mathbf{x}^{\downarrow f})=0$."

The point of statement (iv) is that $G/G^{\downarrow h}$ cannot be identically zero; since G is a potential, $G(\mathbf{x})>0$ for at least one $\mathbf{x}$, and by (iii), $(G/G^{\downarrow h})(\mathbf{x}) > 0$ for this $\mathbf{x}$.

The last two statements, (v) and (vi), are the ones of most direct importance. They tell us that if $G^{\downarrow h}$ is multiplying or dividing G on one side of an equation, we can eliminate it from that side of the equation without regard to the fact that it might take the value zero.

Ratio Potentials. Suppose A is a potential, and g and h are contained in A's domain. We set

$$A^{h|g} = A^{\downarrow g \cup h}/A^{\downarrow g}, \tag{6.1}$$

and we refer to $A^{h|g}$ as A's *ratio potential for h given g*.

When A is a potential on $\mathcal{X}$, we will sometimes abbreviate $A^{\mathcal{X}|g}$ to $A^{|g}$. We will also simplify the notation when we are dealing with single variables; when X and Y are variables in the domain of A, we will write $A^{Y|g}$ instead of $A^{\{Y\}|g}$, $A^{h|X}$ instead of $A^{h|\{X\}}$, and $A^{Y|X}$ instead of $A^{\{Y\}|\{X\}}$.

Notice that $A^{|\varnothing} = A/A^{\downarrow\varnothing}$, which is simply A divided by the sum of all A's values. In other words, $A^{|\varnothing}$ is the probability distribution proportional to A.

Applying part (v) of Proposition 6.1 to (6.1), we see that

$$A^{\downarrow g\cup h} = A^{\downarrow g} A^{h|g} \tag{6.2}$$

always holds.

The potentials $A^{h|g}$ and $A^{\downarrow g\cup h}$ have the same domain, g∪h, and by Proposition 6.1, they are non-zero on the same elements of $\mathcal{W}_{g\cup h}$. If g∪h=g∪f, then $A^{h|g} = A^{f|g}$.

If B is proportional to A, then $B^{h|g}=A^{h|g}$. In particular, $P^{h|g}=A^{h|g}$, where P is the probability distribution proportional to A.

Our next two propositions explore factorization implies for marginals and ratio potentials. Proposition 6.3 lays the groundwork for our study of probabilistic independence. Proposition 6.2 is a simple extension of Proposition 3.2; its main use is in the proof of Proposition 6.3.

Proposition 6.2. If G is an array on g and H is an array on h, then $(GH)^{\downarrow g\cap h} = G^{\downarrow g\cap h} H^{\downarrow g\cap h}$.

Proposition 6.3. Suppose A is a potential on g∪h. Then the following statements are all equivalent.

(i) $A\, A^{\downarrow g\cap h} = A^{\downarrow g}\, A^{\downarrow h}$.

(ii) $A = A^{\downarrow h}\, A^{g|g\cap h}$.

(iii) $A = A^{\downarrow g}\, A^{h|g\cap h}$.

(iv) $A^{g\cup h|g\cap h} = A^{g|g\cap h}\, A^{h|g\cap h}$.

(v) A factors on {g,h}.

????

(vi) $(A^{g|h})^{\downarrow g} = A^{g|g\cap h}$.

(vii) $(A^{h|g})^{\downarrow h} = A^{h|g\cap h}$.

(viii) $A^{g|h}$ is carried by g.

(ix) $A^{h|g}$ is carried by h.

Indicator Potentials. A potential I on a set of variables h is called an *indicator potential* if I(**x**) is equal to zero or one for all $\mathbf{x}\in\mathcal{W}_h$. For each $\mathbf{y}\in\mathcal{W}_h$, we define an indicator potential $I^{h=\mathbf{y}}$ on h by

$$I^{h=\mathbf{y}}(\mathbf{z}) = \begin{cases} 0 & \text{if } \mathbf{z} \neq \mathbf{y} \\ 1 & \text{if } \mathbf{z} = \mathbf{y}, \end{cases}$$

and we call $I^{h=\mathbf{y}}$ the *indicator potential for h=**y***.

This definition specializes, of course, to the case of a single variable. If Y is a variable and $y\in\mathcal{W}_Y$, then the indicator potential $I^{Y=y}$ on Y given by

$$I^{Y=y}(z) = \begin{cases} 0 & \text{if } z \neq y \\ 1 & \text{if } z = y \end{cases}$$

is called the *indicator potential for Y=y*.

Notice that if $h=\{Y_1, ..., Y_n\}$ and $\mathbf{y}=(y_1, ..., y_n)$, then

$$I^{h=\mathbf{y}} = I^{Y_1=y_1} \cdots I^{Y_n=y_n}. \tag{6.3}$$

Conditional Probability Distributions. Suppose P is probability distribution on $\mathcal{X}$, suppose h is a subset of $\mathcal{X}$, and suppose $\mathbf{y}\in\mathcal{W}_h$. Then we let $P^{|h=\mathbf{y}}$ denote the array on $\mathcal{X}$ given by

$$P^{|h=\mathbf{y}} = P^{|h}\, I^{h=\mathbf{y}}. \tag{6.4}$$

If $P^{\downarrow h}(\mathbf{y})=0$, then $P^{|h=\mathbf{y}}$ is identically equal to zero. We are more interested in the case $P^{\downarrow h}(\mathbf{y}) > 0$.

Proposition 6.4. If $P^{\downarrow h}(\mathbf{y}) > 0$, then $P^{|h=\mathbf{y}}$ is a probability distribution.

When $P^{\downarrow h}(\mathbf{y}) > 0$, we call $P^{|h=\mathbf{y}}$ the *conditional distribution given h=**y***.

The values of $P^{|h=\mathbf{y}}$ are indeed conditional probabilities, as this term is usually understood. To see that this is so, we rewrite (6.4) as

$$P^{|h=\mathbf{y}}(\mathbf{x}) = I^{h=\mathbf{y}}(\mathbf{x}^{\downarrow h})\, P^{|h}(\mathbf{x}) = I^{h=\mathbf{y}}(\mathbf{x}^{\downarrow h})\, P(\mathbf{x}) / P^{\downarrow h}(\mathbf{x}^{\downarrow h})$$

$$= \begin{cases} 0 & \text{if } \mathbf{x}^{\downarrow h} \neq \mathbf{y}, \\ P(\mathbf{x})/P^{\downarrow h}(\mathbf{y}) & \text{if } \mathbf{x}^{\downarrow h} = \mathbf{y}, \end{cases}$$

or, in a more colloquial notation,

$$P^{|h=\mathbf{y}}(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \text{ and } \mathbf{y} \text{ disagree}, \\ \Pr(\mathcal{X}=\mathbf{x})/\Pr(h=\mathbf{y}) & \text{if } \mathbf{x} \text{ and } \mathbf{y} \text{ agree}, \end{cases}$$

or

$$P^{|h=\mathbf{y}}(\mathbf{x}) = \Pr(\mathcal{X}=\mathbf{x} \;\&\; h=\mathbf{y})/\Pr(h=\mathbf{y}),$$

and this is "the conditional probability that $\mathcal{X}=\mathbf{x}$ given that $h=\mathbf{y}$," as it is usually defined.

Formula (6.4) defines the conditional distributions given h in terms of the ratio potential $P^{|h}$. We can also go the other way: $P^{|h}(\mathbf{x}) = P^{|h=\mathbf{x}^{\downarrow h}}(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{W}_{\mathcal{X}}$.

Independence. The independence of events or variables with respect to a probability distribution is usually defined in terms of the multiplication of their probabilities. Two events are independent if the probability of both happening is the product of their separate probabilities. The variables are independent if the probability of their jointly taking a pair of values is always equal to the product of the probabilities of their separately taking these values.

This approach to independence is perfectly adequate, and it generalizes readily from a pair of variables to disjoint sets of variables. The formulations that work for disjoint sets of variables also work for overlapping sets of variables if the probability distribution is strictly positive. But some complications arise if zero probabilities are allowed.

Fortunately, in the context of a hypertree, the complications that arise from zero probabilities can be dealt with straightforwardly. In order to deal with them, we need the concept of a variable or a set of variables being determined by a probability distribution.

Proposition 6.5. Suppose P is a probability distribution on $\mathcal{X}$, and suppose f is a subset of $\mathcal{X}$. Then the following statements are all equivalent.

(i) There exists $\mathbf{x} \in \mathcal{W}_f$ such that $P^{\downarrow f}(\mathbf{x})=1$.

(ii) $P = P^{|f}$.

(iii) $P = P\, P^{\downarrow f}$.

If the statements in Proposition 6.5 are satisfied, then we say that f is *determined by P*, or that the marginal $P^{\downarrow f}$ is *categorical*. Notice that the empty set is determined by any probability distribution; we always have $P^{\downarrow \varnothing}(\blacklozenge)=1$.

Now we state general conditions for g and h to be independent with respect to a probability distribution P, conditions that apply whether or not g and h are disjoint.

Proposition 6.6. Suppose P is a probability distribution on $g \cup h$. Then the following statements are all equivalent.

(i) $P = P^{\downarrow g}\, P^{\downarrow h}$.

(ii) P factors on {g, h}, and $P^{\downarrow g \cap h}$ is categorical.

(iii) $(P^{g|h})^{\downarrow g} = P^{\downarrow g}$.

(iv) $(P^{h|g})^{\downarrow g} = P^{\downarrow h}$.

(v) $P^{g|h}$ is carried by g.

(vi) $P^{h|g}$ is carried by h.

If the statements in Proposition 6.6 are satisfied, then we say that g and h are *independent with respect to P*. If P is a probability distribution on a set of variables larger than $g \cup h$, then we say g and h are independent with respect to P if they are independent with respect to $P^{\downarrow g \cup h}$.

We will write $\perp_P [g, h]$ to indicate that g and h are independent with respect to P.

If $g \cap h = \varnothing$, then $P^{\downarrow g \cap h}$ is necessarily categorical. So disjoint sets of variables g and h are independent with respect to a probability distribution P if and only if P factors on {g, h}.

Conditional Independence. Suppose P is a probability distribution, and suppose f, g, and h are contained in P's domain. We say that g and h are *conditionally independent given f with respect to P* if g and h are independent with respect to $P^{|f=\mathbf{z}}$ for every $\mathbf{z} \in \mathcal{W}_f$ such that $P^{\downarrow f}(\mathbf{z}) > 0$.

We will write $f \rightarrow \perp_P [g, h]$ to indicate that g and h are conditionally independent given f with respect to P.

Notice that independence given the empty set is the same as unconditional independence; $\varnothing \rightarrow \perp_P [g, h]$ means $\perp_P [g, h]$.

Before stating equivalent conditions for conditional independence, we must extend further our vocabulary for dealing with zero probabilities. Suppose P is a probability distribution, and suppose g and h are contained in P's domain. We say that h is *determined by P and g* if h is determined by $P^{|g=\mathbf{x}}$ for every $\mathbf{x} \in \mathcal{W}_g$ such that $P^{\downarrow g}(\mathbf{x}) > 0$.

Proposition 6.7. Suppose P is probability distribution, and suppose g and h are contained in P's domain. Then the following conditions are equivalent.

(i) h is determined by P and g.

(ii) There exists a mapping $\delta: \mathcal{W}_g \rightarrow \mathcal{W}_{g \cup h}$ such that $\delta(\mathbf{x})^{\downarrow g} = \mathbf{x}$ and $P^{g \cup h | g=\mathbf{x}}(\delta(\mathbf{x})) = 1$ whenever $\mathbf{x} \in \mathcal{W}_g$ and $P^{\downarrow g}(\mathbf{x}) > 0$.

(iii) There exists a mapping $\delta: \mathcal{W}_g \rightarrow \mathcal{W}_{g \cup h}$ such that $\delta(\mathbf{x})^{\downarrow g} = \mathbf{x}$ and $P^{\downarrow g \cup h}(\delta(\mathbf{x})) = P^{\downarrow g}(\mathbf{x})$ whenever $\mathbf{x} \in \mathcal{W}_g$ and $P^{\downarrow g}(\mathbf{x}) > 0$.

(iv) For every $\mathbf{x} \in \mathcal{W}_g$ such that $P^{\downarrow g}(\mathbf{x}) > 0$, there is only one $\mathbf{z} \in \mathcal{W}_{g \cup h}$ such $\mathbf{z}^{\downarrow g} = \mathbf{x}$ and $P^{\downarrow g \cup h}(\mathbf{z}) > 0$.

(v) For every $\mathbf{z} \in \mathcal{W}_{g \cup h}$, either $P^{\downarrow g \cup h}(\mathbf{z}) = 0$ or $P^{\downarrow g \cup h}(\mathbf{z}) = P^{\downarrow g}(\mathbf{z})$.

Condition (ii) of Proposition 6.7 can be paraphrased by saying that if g=**x**, then with probability one h=δ(**x**). The variables in h are a function of the variables in g with probability one, and δ is the function.

If h⊆g, then P and g determine h no matter what P looks like. If P is strictly positive, then P and g determine h if and only if h⊆g. If P and one set of variables determine h, then P and any larger set also determine h. A probability distribution P and the empty set determine a set of variables h if and only $P^{\downarrow h}$ is categorical.

Proposition 6.8. Suppose P is probability distribution, and suppose f, g, and h are contained in P's domain. If P and g determine f, then $P^{h \cup f | g} = P^{h | g \cup f}$.

Proposition 6.9. Suppose P is a probability distribution, and suppose f, g, and h are contained in P's domain. Then the following conditions are all equivalent:

(i) $f \rightarrow \perp_P [g, h]$.

(ii) $P^{\downarrow f \cup g \cup h}$ factors on {f∪g, f∪h}, and P and f determine g∩h.

(iii) $P^{\downarrow f \cup g \cup h} P^{\downarrow f} = P^{\downarrow f \cup g} P^{\downarrow f \cup h}$.

(iv) $P^{\downarrow f \cup g \cup h} = P^{\downarrow f \cup g} P^{h | f}$.

(v) $P^{\downarrow f \cup g \cup h} = P^{\downarrow f \cup h} P^{g | f}$.

(vi) $P^{g \cup h | f} = P^{g | f} P^{h | f}$.

(vii) $(P^{g | f \cup h})^{\downarrow g \cup f} = P^{g | f}$.

(viii) $(P^{h | f \cup g})^{\downarrow h \cup f} = P^{h | f}$.

In the case where P is strictly positive (i.e., P(**x**) > 0 for every $\mathbf{x} \in \mathcal{W}_\mathcal{X}$), the condition that P and f determine g∩h simplifies to the condition that g∩h⊆f. In any case, g∩h⊆f is always a sufficient condition for P and f to determine g∩h.

The following proposition shows us how to generalize the concept of independence from two sets of variables to more than two.

Proposition 6.10. Suppose P is a probability distribution, and f and h_1, ..., h_n are in P's domain. Then the following conditions are all equivalent.

(i) $P^{h_1 \cup \cdots \cup h_n | f} = P^{h_1|f} P^{h_2|f} \ldots P^{h_n|f}$.

(ii) $P^{\downarrow f \cup h_1 \cup \cdots \cup h_n}$ factors on $\{f \cup h_1, \ldots, f \cup h_n\}$, and P and f determine $h_i \cap h_j$ for all i and j, $1 \leq i < j \leq n$.

(iii) $f \rightarrow \perp_P [h_1 \cup \cdots \cup h_{k-1}, h_k]$ for k=2, ..., n.

(iv) $f \rightarrow \perp_P [g_1, g_2]$ whenever g_1 is the union of one subset of the h_1, ..., h_n and g_2 is the union of another subset, disjoint from the first.

When the conditions of Proposition 6.10 are met, we say that h_1, ...,h_n are *conditionally independent given f with respect to P*, and we write $f \rightarrow \perp_P [h_1, \ldots, h_n]$.

Markov Probability Distributions. Suppose P is a probability distribution on $\mathcal{X}$, and suppose $\mathcal{H}$ is a hypertree on $\mathcal{X}$. If P factors on $\mathcal{H}$, then we say that P is *Markov with respect to* $\mathcal{H}$.

Proposition 6.11. Suppose P is a joint probability distribution for variables in $\mathcal{X}$, and suppose $\mathcal{H}$ is a hypertree on $\mathcal{X}$. Then the following statements are all equivalent.

(i) P factors on $\mathcal{H}$.

(ii) There is a hypertree construction sequence $h_1 \ldots h_n$ for $\mathcal{H}$ and a branching β for $h_1 \ldots h_n$ such that $\beta(h_k) \rightarrow \perp_P [h_1 \cup \cdots \cup h_{k-1}, h_k]$ for k=2, ..., n.

(iii) There is a hypertree construction sequence $h_1 \ldots h_n$ for $\mathcal{H}$ and a branching β for $h_1 \ldots h_n$ such that $P = P^{\downarrow h_1} P^{h_2|\beta(h_2)} \ldots P^{h_n|\beta(h_n)}$.

(iv) For any hypertree construction sequence $h_1 \ldots h_n$ for $\mathcal{H}$ and any branching β for $h_1 \ldots h_n$, $\beta(h_k) \rightarrow \perp_P [h_1 \cup \cdots \cup h_{k-1}, h_k]$ for k=2, ..., n.

(v) For any hypertree construction sequence $h_1 \ldots h_n$ for $\mathcal{H}$ and any branching β for $h_1 \ldots h_n$, $P = P^{\downarrow h_1} P^{h_2|\beta(h_2)} \ldots P^{h_n|\beta(h_n)}$.

(vi) Suppose β is a branching for $\mathcal{H}$, and suppose $(\mathcal{H}, \mathcal{E})$ is the Markov tree determined by β. Suppose $\mathcal{H}$ is a vertex of this tree. Then

the trees in the forest resulting from the removal of $\mathcal{H}$ are independent given $\mathcal{H}$ with respect to P.

The conditional independence relations mentioned in parts (ii), (iv), and (vi) of this proposition are all represented graphically by separation. This is what is signaled in general by the name *Markov*. The Markov property is that separation implies independence given the separator.

The main point of Proposition 6.11 is that factorization is equivalent the Markov property in a hypertree, even when zero probabilities are allowed. This equivalence can be generalized to hypergraphs that are not hypertrees if zero probabilities are prohibited; this is known as the Gibbs-Markov equivalence [Speed 1979]. The hypergraph case is more complicated, however, if zero probabilities are allowed; see Moussouris [1974].

6.2. Conditioning Factorizations

Conditioning plays an important role in the assessment of evidence. If the probability distribution P represents our assessment of a given body of evidence, and we add to that evidence the observation that f=**y**, then we may want to change our assessment to $P^{|f=\mathbf{y}}$. (See Shafer [1985] for a discussion of when this is appropriate.)

Suppose the probability distribution P represents our assessment of a given body of evidence, and we have been computing marginals for P from the factorization

$$P = \Pi\{R_h \mid h \in \mathcal{H}\}, \tag{6.5}$$

where $\mathcal{H}$ is a hypertree on $\mathcal{X}$. Suppose we observe the values of some of the variables in $\mathcal{X}$; say we observe $Y_1=y_1$, $Y_2=y_2$, and so on, up to $Y_n=y_n$. We change our assessment from P to $P^{|f=\mathbf{y}}$, where $f=\{Y_1, ..., Y_n\}$ and $\mathbf{y}=(y_1, ..., y_n)$. We now want to compute marginals for $P^{|f=\mathbf{y}}$, and this would be facilitated by a factorization of $P^{|f=\mathbf{y}}$. Can we adapt (6.5) to a factorization of $P^{|f=\mathbf{y}}$?

Yes, we can. More precisely, we can adapt (6.5) to a factorization of a potential proportional to $P^{|f=\mathbf{y}}$, and this, as we noted in sections 3.3 and 3.4, is good enough.

The adaptation is simple. We know from (6.3) and (6.4) that

$$\begin{aligned} P^{|f=\mathbf{y}} &= I^{h=\mathbf{y}}\, P^{|f} \\ &= I^{Y_1=y_1} \cdots I^{Y_n=y_n}\, P \,/\, P^{\downarrow f}(\mathbf{y}) \\ &= I^{Y_1=y_1} \cdots I^{Y_n=y_n}\, \Pi\{R_h \mid h\in\mathcal{H}\} \,/\, P^{\downarrow f}(\mathbf{y}), \end{aligned}$$

or

$$P^{|f=\mathbf{y}} \propto I^{Y_1=y_1} \cdots I^{Y_n=y_n}\, \Pi\{R_h \mid h\in\mathcal{H}\}, \tag{6.6}$$

where the constant of proportionality is $1/P^{\downarrow h}(\mathbf{y})$. And we can make the right-hand side of (6.6) into a factorization on $\mathcal{H}$ by absorbing each of the $I^{Y_i=y_i}$ into one of the R_h. Indeed, each Y_i is contained in at least one hyperedge in $\mathcal{H}$. Choose one and call it h(i). Set $S_h = R_h\, \Pi\{I^{Y_i=y_i} \mid h=h(i)\}$. Then (6.6) becomes

$$P^{|f=\mathbf{y}} \propto \Pi\{S_h \mid h\in\mathcal{H}\}, \tag{6.7}$$

where the constant of proportionality is still $1/P^{\downarrow h}(\mathbf{y})$.

Aside from its usefulness for computing $P^{|f=\mathbf{y}}$, the possibility of adapting (6.5) to (6.7) is also sometimes useful as a way of computing $P^{\downarrow f}(\mathbf{y})$ for an individual $\mathbf{y}$ in $\mathcal{W}_f$. We run the algorithm of section 3.5, say, with a factorization (6.5), obtaining P's marginal for a particular hyperedge h_1. Then we run the algorithm again, with the factorization (6.7). The result will be a potential on h_1 differing from the first by the factor $1/P^{\downarrow f}(\mathbf{y})$. We can find $P^{\downarrow f}(\mathbf{y})$ by comparing the two results.

This device for computing $P^{\downarrow f}(\mathbf{y})$ will not be needed if f is one of the hyperedges in the hypertree $\mathcal{H}$. For then one run of the algorithm of section 3.5 will give us the whole marginal $P^{\downarrow f}$. But we may sometimes want to compute a probability $P^{\downarrow f}(\mathbf{y})$ for an f that is not in $\mathcal{H}$ and cannot be added without forcing us to a hypertree cover with hyperedges too large for the algorithm to be practical.

6.3. Conditional Independence in Modeling

In this section, we point to two related sources of factored probability distributions: causal reasoning and log-linear statistical modeling. When causal reasoning is put into a probabilistic setting, restrictions on paths of causation can be used to justify assumptions of conditional independence that lead to factorized probabilities (Blalock [1971], Pearl [1986], Wold [1954], Wright [1934]). When log-linear models are fit to discrete statistical data, the simplest models are those in which interactions are of low order, and hence Occam's razor leads to factorized probabilities (Besag [1972, 1974], Darroch, Lauritzen and Speed [1980], Edwards and Kreiner [1983], Lauritzen [1982], Wermuth and Lauritzen [1983]).

Causal Models. In this monograph, we are primarily concerned with probabilities that result from deliberate assessment of evidence and that represent a judgment about the degree to which the evidence supports a proposition. Such probabilities are called *judgmental* or *subjective*, and they are often contrasted with *objective* probabilities, which are supposed to be properties of the world. The objective probability of an event is the propensity of this event to occur, and this propensity is manifested by the frequency with which the event does occur in repeated trials. This objective interpretation, though it is not overtly concerned with the assessment of evidence, often underlies practical probability assessments. We often assess the probability of an event by thinking about how often the event would occur under the circumstances.

If an event has different probabilities under different circumstances, then it is natural to say that the circumstances are helping cause the event. Thus a set of probability assessments based on the objective interpretation can be thought of as a causal model.

Too simple-minded an identification of statistical frequency with causation is dangerous. We are often warned that correlation does not imply causation. It is sometimes reasonable, however, to suppose that causation implies correlation. If two situations differ only in the presence of one cause, then the probability in the situation where the cause is present should be higher.

The probabilistic interpretation of causation for one cause involves the assumption of a probability distribution for other causes. If a given cause A

influences an event B, but does not completely determine whether B will happen, then there must be other causes of B, and these causes must vary from case to case in their presence or strength. If we say that B has a definite objective probability or frequency when A is present, then we must be assuming some stability for these other causes, some stability in their distribution. In other words, we must be assuming an objective, frequentist, probability distribution for these causes.

If we identify "probabilities in different situations" with conditional probabilities, then lack of causal influence should mean probabilistic independence. Since causal influence can be indirect, it is difficult to formulate this thought precisely. Reasonable formulations are possible, however, when we can appeal to a temporal ordering or some other device that restricts the directions of possible influence.

Pearl [1986] has emphasized the simple case where variables $X_1, X_2, ..., X_n$ are ordered so that X_i can influence X_j only if i<j. In this case, we can say precisely how lack of direct causal influence should imply independence or conditional independence. Since X_2 follows X_1, it may be partially caused, or influenced, by X_1. If we believe it is not, then we should expect X_1 and X_2 to be independent; the probability of a given value of X_2 should be the same no matter what the value of X_1. Since X_3 follows both X_1 and X_n, it can be influenced by neither or one or both. If it is influenced by neither, then it should be independent of both. If it is influenced by X_1 but not by X_2, say, then it should be independent of X_2 given X_1;

$$X_1 \twoheadrightarrow\!\perp_P [X_2, X_3].$$

We can continue this way through the X_k; for each X_k after the first, we choose a subset g_k of $\{X_1, X_2, ..., X_{k-1}\}$ such that X_k is influenced directly by g_k but not by $\{X_1, X_2, ..., X_{k-1}\}-g_k$;

$$g_k \twoheadrightarrow\!\perp_P [\{X_1, X_2, ..., X_{k-1}\}-g_k, X_k]. \qquad (6.8)$$

These conditional independence relations constitute the foundation for a causal model.

Completing the causal model means constructing a probability distribution P on $\{X_1, X_2, ..., X_n\}$ that satisfies (6.8) for k=2, ..., n. To do this, we need to supply the conditional probabilities. We need to specify $P^{\downarrow X_1}$, and we need to specify $P^{X_k|g_k}$ for k=2, ..., n. Doing so gives us a probability distribution P in factored form;

$$P = P^{\downarrow X_1} P^{X_2|g_2} \ldots P^{X_n|g_n}. \qquad (6.9)$$

This is a factorization of P on the hypergraph $\mathcal{H}$, where

$$\mathcal{H} = \{\{X_1\}, \{X_2\}\cup g_2, ..., \{X_n\}\cup g_n\}. \tag{6.10}$$

If this is hypergraph has a hypertree cover with reasonably small hyperedges, then we can apply the methods of chapter 3 to (6.9) to obtain marginal probabilities for P.

We would like, of course, for the hypergraph $\mathcal{H}$ in (6.10) to be a hypertree itself, so we would not have to search for a hypertree cover. One interesting case in which it is hypertree is the case where each g_k has exactly one element. In this case, if we write $b(X_k)$ for the unique element of g_k, then (6.9) becomes

$$P = P^{\downarrow X_1} P^{X_2|b(X_2)} ... P^{X_n|b(X_n)}, \tag{6.11}$$

and (6.10) becomes

$$\mathcal{H} = \{\{X_1\}, \{X_2, b(X_2)\}, ..., \{X_n, b(X_n)\}\}. \tag{6.12}$$

In the next section, we will apply the algorithm of section 3.6 to the factorization (6.11). We will find it convenient, when we do this, to enlarge the hypertree given by (6.12) by adding all the singletons $\{X_2\}$, $\{X_3\}$, ..., $\{X_n\}$. This gives the hypertree

$$\mathcal{H}' = \{\{X_1\}, \{X_2, b(X_2)\}, \{X_2\}, \{X_3, b(X_3)\}, \{X_3\}, ..., \{X_n, b(X_n)\}, \{X_n\}\}. \tag{6.13}$$

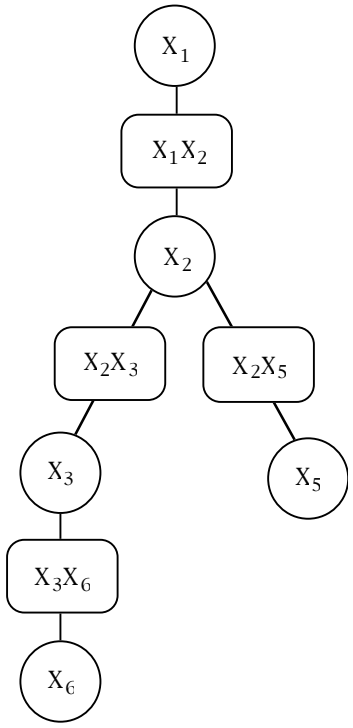
It is also convenient to use the order in which the hyperedges appear in (6.13) as the construction sequence for $\mathcal{H}'$, and to use for this construction sequence the branching β given by

$$\beta(\{X_i, b(X_i)\}) = \{b(X_i)\} \text{ and } \beta(\{X_i\}) = \{X_i, b(X_i)\}, \tag{6.14}$$

for i=2, ..., n.

Figure 6.1 shows the Markov tree determined by (6.13) and (6.14) for a simple example with n=6.

Figure 6.1. The Markov tree determined by (6.13) and (6.14) when n=6, $b(X_2)=X_1$, $b(X_3)=X_2$, $b(X_4)=X_3$, $b(X_5)=X_2$, and $b(X_6)=X_3$. We use circles and rounded rectangles to distinguish graphically between individual and joint variables.



As Figure 6.1 illustrates, the Markov tree determined by (6.13) and (6.14) actually has a very simple structure. It can be obtained by taking the tree $\mathcal{T}=(\mathcal{V}, \mathcal{E})$, where

$$\mathcal{V}=\{X_1, X_2, ..., X_n\} \text{ and } \mathcal{E}=\{\{X_2, b(X_2)\}, ...,\{X_n, b(X_n)\}\}, \qquad (6.15)$$

and turning each edge e in $\mathcal{E}$ into a vertex connected by edges to the vertices in e.

Formally, let us call a pair $(\mathcal{T}, P)$ a *probability tree* if $\mathcal{T}$ is a tree of the form (6.15), P is a probability distribution on the vertices, and P factors as in (6.11). (By Proposition 6.11, P factors as in (6.11) if and only if P factors on the hypertree $\mathcal{E}$.) We will refer to the Markov tree determined by (6.13) and (6.14) as the Markov tree obtained by interpolating vertices on the edges of the probability tree $(\mathcal{T}, P)$.

Log-Linear Statistical Models. Explain briefly the idea of low and high order interactions.

6.4. Local Computation in Probability Trees.

In this section, we review the basic ideas of Bayesian statistical inference, and we see how Bayes's theorem generalizes to the case of probability trees.

The generalization of Bayes's theorem that we study here falls out when we apply the algorithm of section 3.6 to the factorization (6.11). Our way of explaining the generalization is inspired by Pearl [1986], but the details of our approach differ significantly from the details of Pearl's approach. We first explained our approach in Shenoy and Shafer [1986].

Bayesian Statistical Inference. Suppose we are concerned with two variables, X and Y, and we believe that X causes Y. We construct a probability distribution P on {X, Y} by assessing the marginal $P^{\downarrow X}$, assessing the conditional $P^{Y|X}$, and writing

$$P = P^{\downarrow X} P^{Y|X}. \tag{6.16}$$

This formula, usually called *the rule of total probability*, is a special case of (6.2).

Now we observe that Y=y, and we want to compute new probabilities for X taking this observation into account. In other words, we want to compute the conditional distribution $P^{X|Y=y}$. By (6.6),

$$P^{|Y=y} \propto I^{Y=y} P^{\downarrow X} P^{Y|X}.$$

So

$$P^{X|Y=y} \propto (I^{Y=y} P^{\downarrow X} P^{Y|X})^{\downarrow X} = P^{\downarrow X} (I^{Y=y} P^{Y|X})^{\downarrow X}.$$

Since $(I^{Y=y} P^{Y|X})^{\downarrow X}(x) = P^{Y|X}(y,x)$, this can be written

$$P^{X|Y=y} \propto P^{\downarrow X} L^{X|Y=y}, \tag{6.17}$$

where $L^{X|Y=y}$ is the potential on X given by

$$L^{X|Y=y}(x) = P^{Y|X}(y,x). \tag{6.18}$$

Formula (6.17) is often called *Bayes's theorem*.

In the terminology of Bayesian statistics, the probability distribution $P^{X|Y=y}$ is the *posterior* distribution for X, the probability distribution $P^{\downarrow X}$ is the *prior* distribution for X, and the potential $L^{X|Y=y}$ is the *likelihood*. Thus Bayes's theorem can be written more verbally as

$$\text{posterior}(x) \propto \text{prior}(x)\ \text{likelihood}(x). \tag{6.19}$$

The utility of Bayes's theorem depends, of course, on whether we have in hand the factorization (6.16). If our joint probability distribution P has been constructed following (6.16), then the likelihood is easily found by (6.18), and Bayes's theorem provides an efficient way of computing the conditional distribution $P^{X|Y=y}$. If, on the other hand, P has not been constructed in this way, then we will likely find $P^{X|Y=y}$ in some other way, and Bayes's theorem will have no role to play.

Bayes's theorem is particularly interesting when the ratio potential $P^{Y|X}$ can be given a relatively objective or frequentist interpretation. As we argued in the preceding section, this requires a certain stability in the distribution in the other causes of Y. If we are to talk about the probability that Y=y when X=x, without reference to which situation with X=x we have in mind, there must be some stability in how the other causes of Y operate whenever X=x. But an objective interpretation of $P^{Y|X}$ does not imply an objective interpretation of the marginal $P^{\downarrow X}$. The distribution of X may still be unstable, so that any probabilities for X must have a more subjective character. So even if the likelihood in Bayes's theorem has an objective interpretation, and we hold $P^{Y|X}$ constant as we consider the evidence about the variables X and Y for different individuals, the prior can have a subjective interpretation, and we may to assess it differently for each individual.

Since the time of Laplace, mathematical statistics has been particularly concerned with the situation just described, and controversy has persisted between those who, like Laplace, are willing to make the subjective judgments required to construct the prior distribution $P^{\downarrow X}$ and those who are not willing to do so. Those willing to construct $P^{\downarrow X}$ are now often called *Bayesians*, since they are able to carry through the use of Bayes's theorem, while those not willing to construct $P^{\downarrow X}$ are often called *objectivists*, since they want to find modes of judgment that use only

objective probabilities. Both Bayesians and objectivists have tended to take it for granted that the ratio potential $P^{Y|X}$ is objective, stable, and known.

Unfortunately, the stylized nature of the this hoary controversy gives undue weight to Bayes's theorem. As we have already argued, if P is not constructed following (6.16), and we want to compute $P^{X|Y=y}$ after observing Y=y, we are likely to do so in some way other than Bayes's theorem. Bayes's theorem should be seen as a computational device for computing conditional probabilities in certain circumstances, not as a philosophical foundation for probability judgment in general, as the term *Bayesian* suggests.

Moreover, the situation where we have an objective likelihood and a subjective prior is characteristic merely of a set of problems that have historically been of interest to statisticians. We cannot assume that is characteristic of all problems. It may be uncharacteristic, for example, of problems dealt with by modern expert systems.

For further discussion of the distinction between foundational and computation concerns in the construction of probability arguments, see Shafer and Tversky [1985].

Inference in Probability Trees. The ideas underlying Bayes's theorem can be generalized from the simple case of two variables to the case of a probability tree of variables. This generalization has been studied by Kelly and Barclay [1973] and Pearl [1986]. As we will now see, it can be seen as a special case of the algorithm we studied in section 3.6 above.

Recall that $(\mathcal{T}, P)$ is a probability tree when $\mathcal{T}=(\mathcal{V}, \mathcal{E})$, $\mathcal{V}=\{X_1, X_2, ..., X_n\}$, $\mathcal{E}=\{\{X_2, b(X_2)\}, ...,\{X_n, b(X_n)\}\}$, and P is a probability distribution on $\mathcal{V}$ satisfying

$$P = P^{\downarrow X_1} P^{X_2|b(X_2)} ... P^{X_n|b(X_n)}. \tag{6.11}$$

Given the probability tree $(\mathcal{T},P)$, we form the hypertree

$$\mathcal{H}' = \{\{X_1\}, \{X_2, b(X_2)\}, \{X_2\}, \{X_3, b(X_3)\}, \{X_3\}, ..., \{X_n, b(X_n)\}, \{X_n\}\}, \tag{6.13}$$

with the branching β given by

$$\beta(\{X_i, b(X_i)\}) = \{b(X_i)\} \text{ and } \beta(\{X_i\}) = \{X_i, b(X_i)\} \tag{6.14}$$

for i=2, ..., n.

Consider how we can implement the algorithm of section 3.6 for the factorization (6.11) in the Markov tree determined by (6.13) and (6.14).

We must first assign a potential to each vertex of the Markov tree. The natural way of doing this is to assign $P^{\downarrow X_1}$ to $\{X_1\}$ and to assign $P^{X_i|b(X_i)}$ to $\{X_i, b(X_i)\}$. This leaves the $\{X_i\}$, $i\neq 1$, without potentials; we assign each of them a vector of ones.

Figure 6.2 shows this assignment of potentials for the example of Figure 6.1. Here we let 1^X denote the vector of ones on the variable X; $1^X(x)=1$ for every $x\in\mathcal{W}_X$.

Figure 6.2. The assignment of potentials to Markov tree of Figure 6.1. In this example, $P = P^{\downarrow X_1} P^{X_2|X_1} P^{X_3|X_2} P^{X_4|X_3} P^{X_5|X_2} P^{X_6|X_3}$.

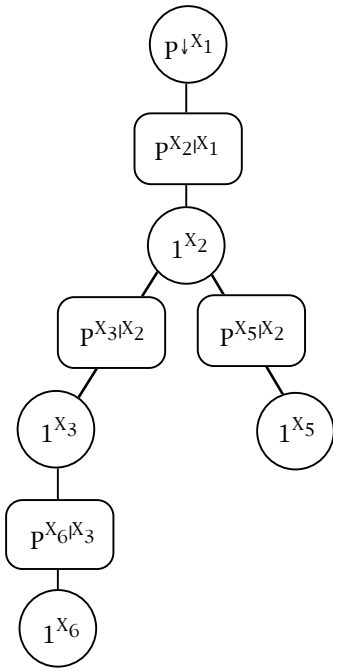
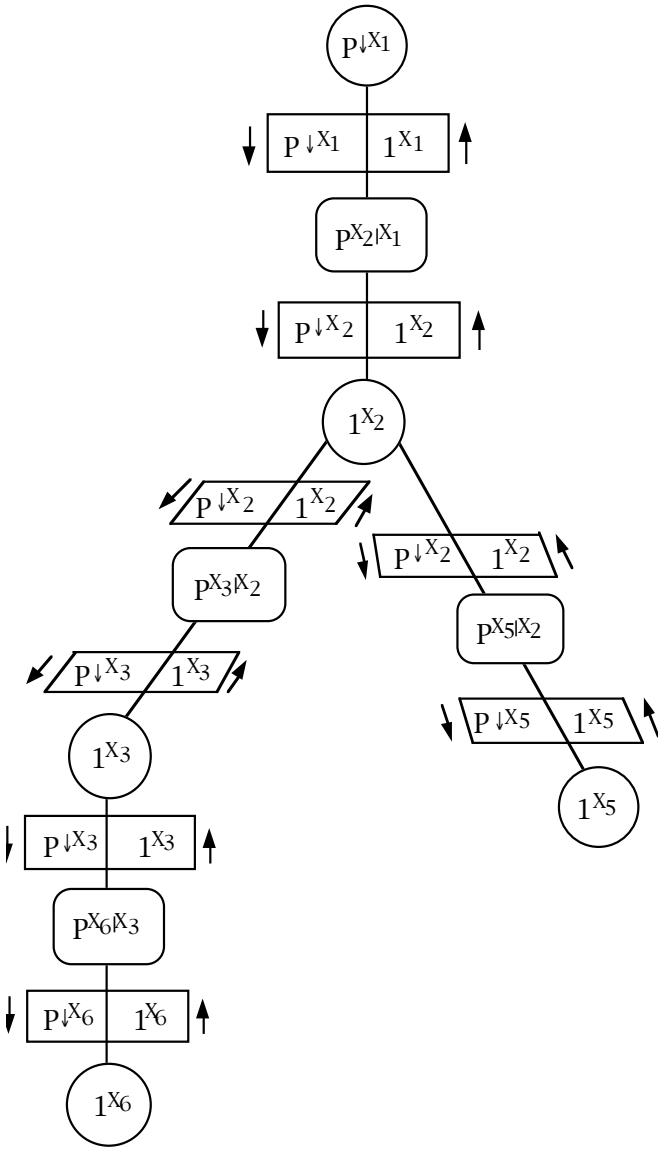


Figure 6.3 shows the messages that will be passed if we run the algorithm of section 3.6 with this assignment of potentials. To verify that the messages given in this figure are correct, begin at the leaves of the Markov tree and work inward. Initially, the three leaves, $\{X_1\}$, $\{X_5\}$, and $\{X_6\}$ send inward their own potentials, $P^{\downarrow X_1}$, 1^{X_5}, and 1^{X_6}. This allows the vertices $\{X_1, X_2\}$, $\{X_2, X_5\}$, and $\{X_3, X_6\}$ to act.

The vertex $\{X_2, X_1\}$ multiplies the message $P^{\downarrow X_1}$ and its own potential, $P^{X_2|X_1}$, and sends to $\{X_2, X_3\}$ the projection $(P^{\downarrow X_1} P^{X_2|X_1})^{\downarrow X_2} = (P^{\downarrow\{X_1,X_2\}})^{\downarrow X_2} = P^{\downarrow X_2}$. And so on.

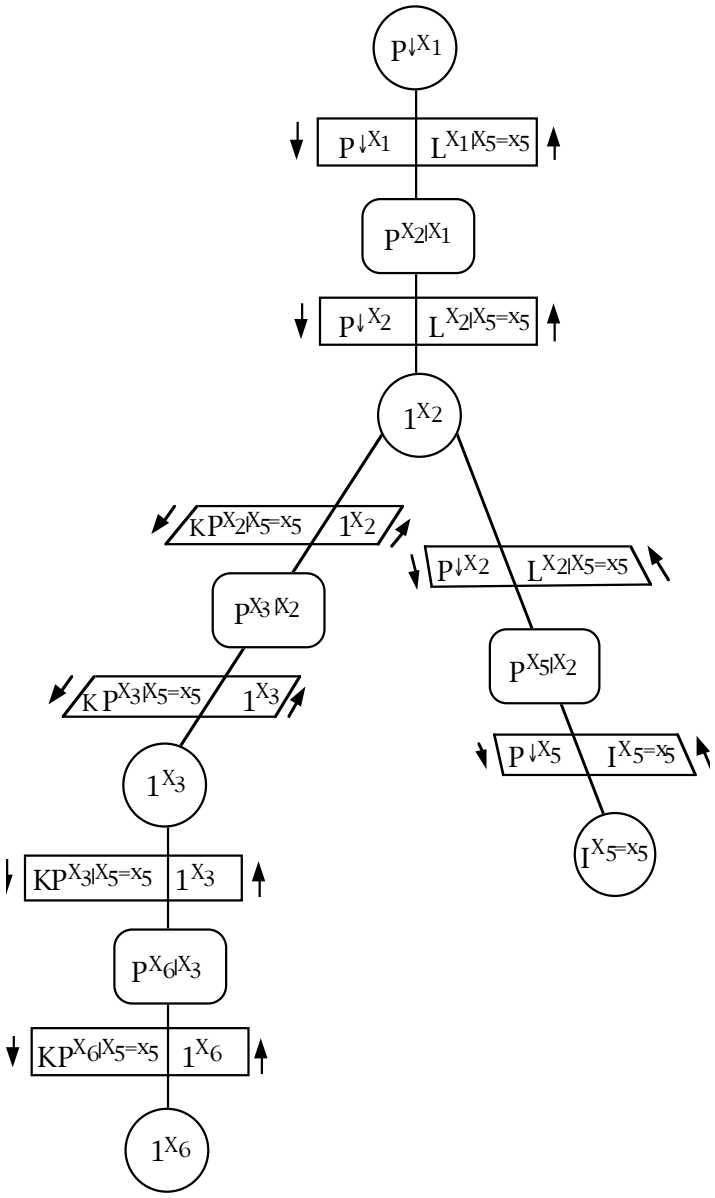
Figure 6.3. The messages passed when the potentials shown in Figure 6.2 are used.



If we observe the value of a variable, we will replace the vector of ones on that variable with an indicator vector, and then we will allow the algorithm to update. Figure 6.4 shows the result when we observe that $X_5=x_5$. The messages

shown in this figure can also be verified by working inwards. For example, vertex $\{X_2, X_5\}$ multiplies its potential $P^{X_5|X_2}$ by the message $I^{X_5=x_5}$ before projecting to $\{X_2\}$. The result is $(I^{X_5=x_5}P^{X_5|X_2})^{\downarrow X_2} = L^{X_2|X_5=x_5}$. Vertex $\{X_2\}$ will send to $\{X_2, X_3\}$ the product $L^{X_2|X_5=x_5}\, P^{X_2}$, which is proportional to $P^{X_2|X_5=x_5}$. At each step, the messages from the direction of X_1 are probability distributions, either prior distributions or distributions posterior to those observations that lie in the direction from which the message is being sent, while the messages towards X_1 are either vectors of ones or likelihoods.

Figure 6.4. The messages passed when the potential on $\{X_5\}$ is changed to indicate the observation $X_5=x_5$.



The computations in Figures 6.3 and 6.4 can be described quite simply in the language of matrices. If we think of the potential $P^{X_i|X_j}$ as a matrix with $P^{X_i|X_j}(x_i, x_j)$ in row x_i and column x_j, then the operation of $\{X_i, b(X_i)\}$ can be described as matrix multiplication; probability vectors on X_j transform to probability vectors on X_i by post-multiplying the matrix as column vectors, and likelihood vectors on X_i

transform to likelihood vectors on X_j by pre-multiplying the matrix as row vectors. The vertices representing individual variables simply multiply the vectors they receive pointwise. For more details, see Shenoy and Shafer [1986].

Though our algorithm, as presented in Figures 6.3 and 6.4, was inspired by Pearl [1986], it differs in detail from the algorithm Pearl himself presents. Pearl bases his algorithm on the probability tree rather than the Markov tree. In the probability tree, we have vertices only for variables, no vertices for the pairs $\{X_i, b(X_i)\}$. So Pearl draws a picture in which all computation and all storage is performed by individual variables.

Pearl's algorithm also differs from ours in other ways. Each individual variable X has access to less storage. Rather than storing all the messages it receives and multiplying only those required when it sends a message, X multiplies all its messages and sends the projection of this overall product as its message to a neighbor Y. The neighbor Y, when it receives this message, re-computes any message it had sent X earlier and divides this message out of the message just received from X. We will not delve further into details here, but we will study the strategy of substituting division for storage in the next section.

The following proposition describes what happens in general when we apply the algorithm of section 3.6 to a probability tree. In stating the proposition, we have assumed that the Markov tree is drawn downward from the root X_1, as in Figures 6.1–6.4, so that we can talk about downward messages and upward messages.

Proposition 6.12. Suppose we apply the algorithm of section 3.6 to the factorization (6.11), in the Markov tree determined by (6.13) and (6.14). We assign $P^{\downarrow X_1}$ to $\{X_1\}$ and $P^{X_i|b(X_i)}$ to $\{X_i, b(X_i)\}$. Consider two cases for the assignment to the other vertices.

(i) We assign 1^{X_i} to $\{X_i\}$, $i>1$. In this case, each downward message is a prior marginal probability distribution, and each upward message is a vector of ones. More precisely,

$$M^{\{b(X_i)\}\to\{X_i,b(X_i)\}} = P^{\downarrow b(X_i)},$$

$$M^{\{X_i,b(X_i)\}\to\{b(X_i)\}} = 1^{b(X_i)},$$

$$M^{\{X_i,b(X_i)\}\to\{X_i\}} = P^{\downarrow X_i},$$

and

$$M^{\{X_i\}\to\{X_i,b(X_i)\}} = 1^{X_i}.$$

See Figure 6.5.

(ii) We choose a subset $g=\{Y_1, ..., Y_k\}$ of $\{X_2, ..., X_n\}$, and we choose an element $\mathbf{y}=(y_1, y_2, ..., y_k)$ of $\mathcal{W}_g$. We assign $I^{Y_i=y_i}$ to $\{Y_i\}$, and we assign 1^{X_i} to $\{X_i\}$ for X_i in $\{X_2, ..., X_n\}$–g. In this case, each downward message is proportional to the probability distribution posterior to all the observations except those below it, and each upward message is the likelihood potential given the observations below it.

In order to say this more precisely, we need more notation. For each variable X in $\{X_1, ..., X_n\}$, we let $\bullet X$ denote the subset of $\{X_1, ..., X_n\}$ consisting X together with all the variables that lie below X in the Markov tree, and we let $\circ X$ denote the others; $\circ X=\{X_1, ..., X_n\}-(\bullet X)$. We let $g\bullet X$ denote the intersection $g\cap(\bullet X)$, and we let $\mathbf{y}\bullet X$ denote the projection $\mathbf{y}^{\downarrow g\bullet X}$. Similarly for $g\circ X$ and $\mathbf{y}\circ X$.

With this notation, we can write

$$M^{\{b(X_i)\}\to\{X_i,b(X_i)\}} = P^{b(X_i)|g\circ X_i=\mathbf{y}\circ X_i},$$

$$M^{\{X_i,b(X_i)\}\to\{b(X_i)\}} = L^{b(X_i)|g\bullet X_i=\mathbf{y}\bullet X_i},$$

$$M^{\{X_i,b(X_i)\}\to\{X_i\}} = P^{X_i|g\circ X_i=\mathbf{y}\circ X_i},$$

and

$$M^{\{X_i\}\to\{X_i,b(X_i)\}} = L^{X_i|g\bullet X_i=\mathbf{y}\bullet X_i}.$$

See Figure 6.6.

Figure 6.5. Probabilities down; vectors of ones up.

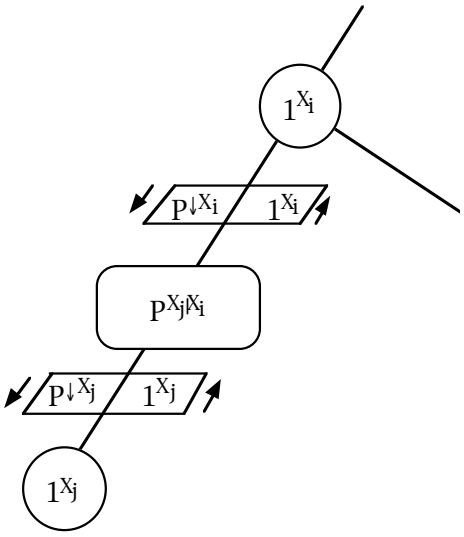
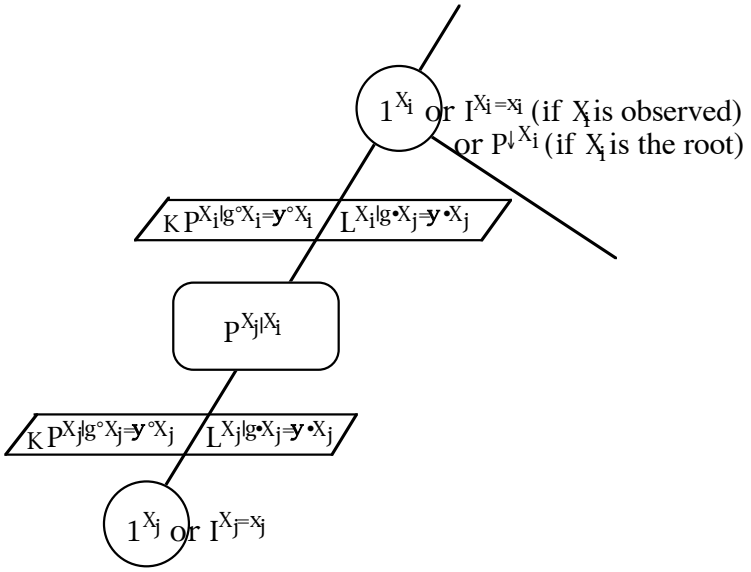


Figure 6.6. Probabilities down, posterior to any observations that do not lie below. Likelihoods up, given any observations that do lie below. If g°X is empty, the corresponding conditional distribution reduces to the unconditional distribution. If g•X is empty, the corresponding likelihood reduces to a vector of ones.

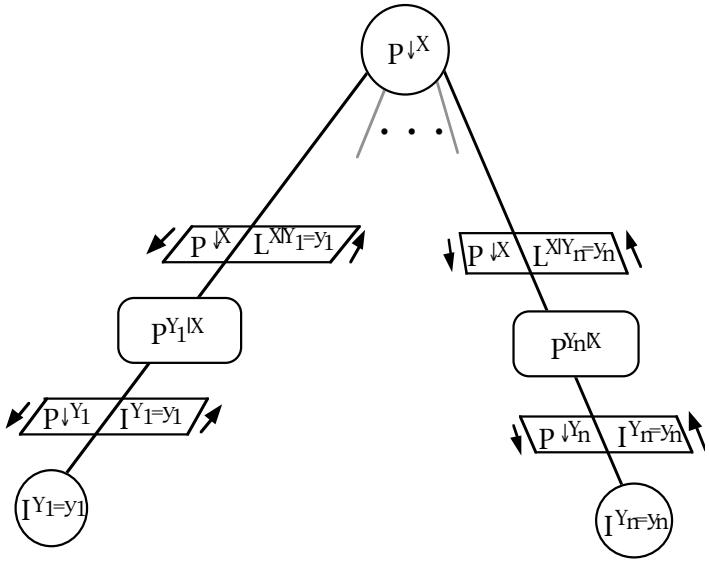


The statement of Proposition 6.12 is so lengthy that we can lose sight of the fact that it is a generalization of Bayes's theorem. As an antidote to this, let us glance at a simple special case that is generally regarded as a version of Bayes's theorem. This is the case where we observe variables Y_1, ..., Y_k, which were initially independent given X. The Markov tree is shown in Figure 6.7. As this figure indicates, the vertex {X} computes a potential that is proportional to its posterior distribution by multiplying its original potential, $P^{\downarrow X}$, by the n messages it receives:

$$P^{X|Y1=y1,\ ...,Yn=yn} \propto P^{\downarrow X} L^{X|Y1=y1} ... L^{X|Yn=yn}.$$

This is Bayes's theorem for n independent observations. When n=1, it reduces to the version of Bayes's theorem given in (6.17).

Figure 6.7. Bayes's theorem for independent observations.



Figures 6.1–6.7 and Proposition 6.12 have an alluring elegance. They are elegant because they unify modeling and computation; they show a computational procedure in which every step can be related to the conditional independence assumptions that underlie the probability model. As in the case of Bayes's theorem, we can imagine that the conditional probabilities are a permanent and stable

feature of our system, which is applied to different individual cases by using different observations and different prior distributions for the root.

We need to recognize, however, that this allure can be dangerous. It can cause confusion. It can even cause willful misunderstanding of the strength and nature of our evidence. If we do not have the evidence needed to complete a causal probability model to match our observations in the way Proposition 6.12 requires, then we should not allow elegance to force us into pretence.

In general, we must distinguish between the needs of modeling and the needs of computation. When we are modeling, we must pay attention to evidence. When we are computing, we have no need for interpretations in terms of modeling. Bayes's theorem, and its generalization, Proposition 6.12, are computational devices that are sometimes applicable, not propositions with foundational or philosophical significance.

6.5. Lauritzen and Spiegelhalter's Algorithm

In this section, we describe Lauritzen and Spiegelhalter's algorithm for computing the marginals of a factored potential for all the vertices of a Markov tree.

Lauritzen and Spiegelhalter's algorithm is best understood in relation to the algorithm of section 3.6, especially the version of that algorithm that sweeps inward to a predetermined vertex and then back outward again.

The main difference between the two algorithms lies in the way they avoid the double counting that can occur if assessments originating in an outer vertex is both kept at that vertex and sent inward on the inward sweep. (If the information sent inward is incorporated into the message coming back on the outward sweep, then it may be misperceived, when it comes back, as independent confirmation.)

The algorithm of section 3.6 avoids double counting by keeping track of the origin (from a local perspective) of all information. During the inward sweep, each vertex stores all messages in registers identifying their source, and during the outward sweep, when g combines the messages it has received to find a message to send back to f, it deliberately omits the message it had received from f.

Lauritzen and Spiegelhalter's algorithm, on the other hand, uses division to avoid double counting. When a vertex sends a message inward, it divides the potential it has stored for itself by that message. In effect, it does not keep the message sent. Thus there is no double counting when it gets the message back.

Lauritzen and Spiegelhalter's algorithm requires much less storage than the algorithm of section 3.6. It requires no storage of messages not currently in use; throughout the computation there is just one potential stored at each vertex. Their algorithm also requires less multiplication on the outward sweep. In the algorithm of section 3.6, the computation of the array that a vertex projects to an outward neighbor involves a different multiplication for each neighbor. In Lauritzen and Spiegelhalter's algorithm, on the other hand, a vertex projects the same potential to all its outward neighbors.

The savings in multiplication that Lauritzen and Spiegelhalter's algorithm achieves on the outward sweep must be balanced against the cost of the divisions that it introduces on the inward sweep. It is difficult to assess this balance in general, but since division is more expensive than multiplication, it is doubtful that Lauritzen and Spiegelhalter's algorithm will achieve great computational savings over the algorithm of section 3.6. As we will see, the attractiveness of their algorithm derives from its interpretation rather than from its efficiency. The divisions that the algorithm introduces produce conditional probabilities, and many of the algorithm's intermediate steps can therefore be interpreted in terms of conditional probability.

We should reiterate that Lauritzen and Spiegelhalter's algorithm is an algorithm for potentials, not an algorithm for arrays in general. The divisions that the algorithm uses will not work, in general, if the arrays are not potentials. Thus we assume that we are working with a potential, and that the factorization is a factorization into potentials.

We begin by discussing Lauritzen and Spiegelhalter's algorithm from a purely computational point of view, without reference to the probabilistic interpretation. We turn then to the issue of updating, and then to the probabilistic interpretation.

The Inward Sweep. The inward sweep of Lauritzen and Spiegelhalter's algorithm is similar to the algorithm of section 3.5, our algorithm for computing the marginal

on a single vertex of a Markov tree. The only difference is that in Lauritzen and Spiegelhalter's algorithm, each vertex does more than send a message inward; it also divides its own potential by the message that it sends inward.

The division does not affect the computation of the marginal for the vertex on which the inward sweep converges, because this computation involves only the messages sent, not the potentials kept. But the division does obviously affect the set of potentials that remain on the vertices when the inward sweep is completed. As we shall see, it means that these potentials constitute a new factorization of the potential whose marginal we are computing.

We can gain some insight into each step of the inward sweep by looking again at Proposition 3.3, where we studied projection from an arbitrary twig in a hypergraph. That proposition tells us that if $\mathcal{H}$ is a hypergraph on $\mathcal{X}$, t is a twig in $\mathcal{H}$, b is a branch for t, and A is an array on $\mathcal{X}$ that factors on $\mathcal{H}$, say

$$A = \Pi\{A_h \mid h \in \mathcal{H}\}, \tag{6.20}$$

then

$$A^{\downarrow \mathcal{X}'} = (A_b A_t^{\downarrow t \cap b}) \, \Pi\{A_h \mid h \in \mathcal{H} - \{t, b\}\}, \tag{6.21}$$

where $\mathcal{X}' = \mathcal{X} - (t - b)$. (Formula (6.21) differs from (3.3) in that we have replaced $A_t^{\rightarrow b}$ by $A_t^{\downarrow t \cap b}$. The replacement makes no difference, since $A_t^{\rightarrow b}$ is the vacuous extension of $A_t^{\downarrow t \cap b}$ to b, and this vacuous extension is implicit in the multiplication by A_b.) We can rewrite (6.20) as

$$A = A_t \, A_b \, \Pi\{A_h \mid h \in \mathcal{H} - \{t, b\}\}. \tag{6.22}$$

If (6.20) is a factorization of a potential into potentials, then part (v) of Proposition 6.1 allows us to divide and multiply the right-hand side of (6.22) by $A_t^{\downarrow t \cap b}$, obtaining

$$A = (A_t / A_t^{\downarrow t \cap b}) \, (A_b A_t^{\downarrow t \cap b}) \, \Pi\{A_h \mid h \in \mathcal{H} - \{t, b\}\}. \tag{6.23}$$

Formula (6.23) tells us that when $A_t^{\downarrow t \cap b}$ is divided out of t's potential and multiplied into b's potential, the result is a new factorization of A on $\mathcal{H}$.

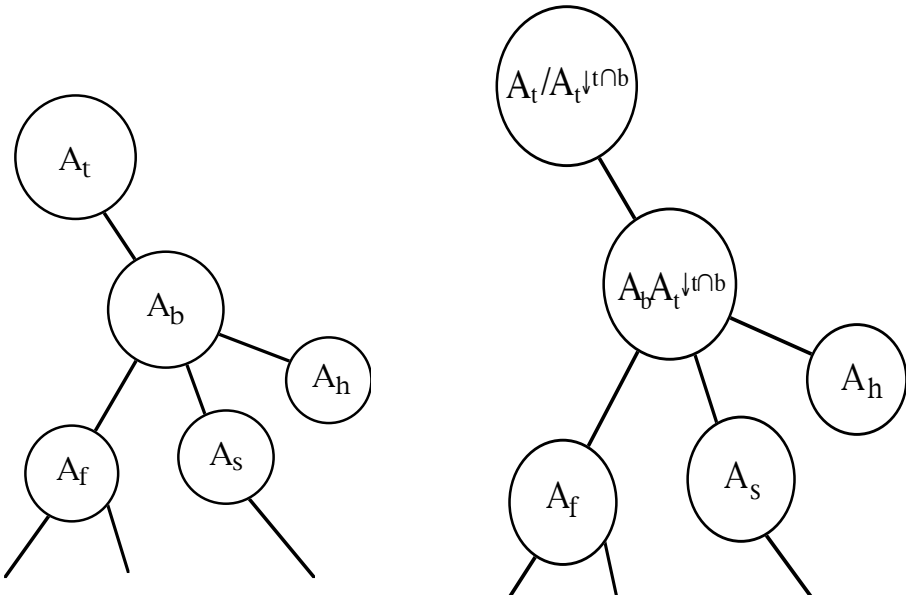
We can say even more. Comparing (6.23) with (6.21), we see that

$$A = (A_t / A_t^{\downarrow t \cap b}) \, A^{\downarrow \mathcal{X}'}; \tag{6.24}$$

the part of the new factorization that remains when t is removed is a factorization of $A^{\downarrow \mathcal{X}'}$. This is the key to the conclusion that repeated application of (6.23) on a

hypertree results in a new factorization. If $\mathcal{H}$ is a hypertree, then removal of the twig t leaves us with a factorization of $A^{\downarrow \mathcal{X}'}$ on the hypertree $\mathcal{H}$–{t}, and hence we can repeat the operation.

Figure 6.8. Revising a factorization on a hypergraph. The new factorization differs from the old only in that $A_t^{\downarrow t\cap b}$ has been divided out of t's potential and multiplied into b's potential. The potentials on all the other hyperedges have been left the same. The new factorization has the property that if the twig t were removed, the potentials on the remaining hyperedges in the factorization would constitute a factorization of the marginal on the union of these hyperedges.



The initial factorization. The new factorization.

In order to formalize our understanding of the inward sweep, we formulate a new basic operation, analogous to Operation H of section 3.5. The setting of the new operation is the same: a hypertree $\mathcal{H}$ on a set of variables $\mathcal{X}$, and an potential A on $\mathcal{X}$, factored into potentials on the hyperedges, as in (6.20). We have chosen a root h_1 and a branching β with this root. Thus the operation can be thought of as an

operation in the hypertree $\mathcal{H}$ with branching β, or, equivalently, as an operation in the Markov tree determined by $\mathcal{H}$ and β.

> *Operation L&S$_1$.* Compute the marginal on h∩β(h) of the potential now on h. Change the potential now on h by dividing it by this marginal. Change the potential now on β(h) by multiplying it by this marginal.

The inward sweep consists of a series of applications of Operation L&S$_1$, converging on the root h_1. We choose any hypertree construction sequence that has β as its branching, say $h_1h_2...h_n$. We apply Operation L&S$_1$ first for $h=h_n$, then for $h=h_{n-1}$, and so on, down to $h=h_2$.

At the end of the inward sweep, the potential assigned to h_1 is the marginal of A on h_1. This is because the messages sent inward toward h_1 are exactly the same as in the scheme described in section 3.5. Operation L&S$_1$ and Operation H send the same message to β(h); they differ only in what they do to the array that remains at h.

Since the messages sent in our inward sweep are exactly the same as the messages sent in section 3.5, we have the same flexibility of control that we observed there. In terms of the hypertree, we can choose any construction sequence with the branching β. In terms of the Markov tree determined by $\mathcal{H}$ and β, we can choose any construction sequence with the budding β. In other words, the only restriction on the order in which the vertices act is that all the vertices that use a given vertex as a bud must act before it acts.

We can allow the vertices to delay their multiplications until they have received all their messages, as we did when formulating Operation M$_2$ in section 3.5. One motivation for the division in Lauritzen and Spiegelhalter's algorithm, however, is to avoid the storage requirements that such delay entails. In our further discussion, therefore, we will assume that messages are absorbed as they are sent.

The Outward Sweep. Suppose we have completed the inward sweep. We now have new potentials on all the vertices, and the new potential on h_1 is $A^{\downarrow h_1}$. How do we compute the marginals for the other vertices?

The answer is that we proceed from h_1 back outward in the Markov tree, projecting from bud to leaf, and multiplying the potential on the leaf by the projection. At each step, as we shall see, this product is the marginal for the leaf.

In this outward sweep, we do not perform any divisions. Thus our basic operation is simply the following:

> *Operation L&S$_2$.* Compute the marginal on $\beta(h) \cap h$ of the array now on $\beta(h)$. Change the potential now on h by multiplying it by this marginal.

The budding β is, of course, the same budding we used on the inward sweep. Its root is h_1, the vertex on which the inner sweep converged.

We can apply Operation L&S$_2$ following the same construction sequence $h_1h_2...h_n$ that we used in the inward sweep. We apply the operation first for $h=h_2$, then for $h=h_3$, and so on, up to $h=h_n$. (Since $\beta(h_2)=h_1$, the first projection will be from h_1 to h_2.) Alternatively, we can follow some other tree construction sequence that uses the budding β. In other words, we move outward in the Markov tree however we like, provided only that we never execute Operation L&S$_2$ for a vertex before having executed it for the vertex's bud.

We can now summarize how Lauritzen and Spiegelhalter's algorithm compares with the algorithm of section 3.6. Both algorithms involve an inward and an outward sweep. The only difference in the flow of the computation is that in Lauritzen and Spiegelhalter's algorithm, we must single out a vertex h_1 in the Markov tree and force the inward sweep to converge on that vertex. In the algorithm of section 3.6, making such an a priori choice of a central vertex is optional, and it is permissible to have two central vertices that send their messages to each other simultaneously. On the inward sweep, the message sent by each vertex is the same under the two algorithms, but Lauritzen and Spiegelhalter's algorithm has each vertex divide its own potential by that message. On the outward sweep, Lauritzen and Spiegelhalter's algorithm has each vertex multiply its original potential by all its messages received and project this to all its outward neighbors, whereas the algorithm of 3.6 omits from the projection to each neighbor the message received from that neighbor.

Here is a formal statement of our claim that the outward sweep will produce all the marginals.

> *Proposition 6.13.* At the end of the outward sweep, the potential on each vertex is the marginal of A for that vertex.

The formal proof given at the end of the chapter is based, of course, on the comparison with the algorithm of section 3.6. We already know that the two algorithms give the same answer for h_1. We complete the proof by induction on the outward sweep.

Updating. In section 3.6, we noted that the algorithm presented there offered opportunities for modest computational savings in updating. If just one potential in the original factorization were changed, the marginals could be updated with only about half the amount of message passing required in the initial computation. This opportunity for savings in updating is not so manifest in Lauritzen and Spiegelhalter's algorithm.

In order for updating to be possible at all, of course, we must retain a factorization. The way we have described Lauritzen and Spiegelhalter's algorithm, the factorization is not retained. At the end of the inward sweep, the factorization has been replaced by a new factorization. At the end of the outward sweep, this new factorization has been replaced by the marginals, which are not a factorization. But we can store the original factorization if we want, and we can update by repeating the algorithm with whatever changes in that factorization we want.

Another approach would be to store the new factorization that has been obtained at the end of the inward sweep, and to use it for updating. As we will see in a moment, this approach does provide some opportunity for computational savings.

Interpretation. We now turn to the probabilistic interpretation of the operations in Lauritzen and Spiegelhalter's algorithm.

Let us return to equations (6.20)–(6.24), where we analyzed the removal of a twig t from a factorization of a potential A on a hypergraph $\mathcal{H}$. Equation (6.24) says that

$$A = (A_t / A_t^{\downarrow t\cap b})\, A^{\downarrow \mathcal{X}'}. \tag{6.24}$$

We can justify dividing both sides of this equation by $A^{\downarrow \mathcal{X}'}$, thus obtaining

$$A / A^{\downarrow \mathcal{X}'} = (A_t^{t|t\cap b})^{\uparrow \mathcal{X}}.$$

Since $A/A^{\downarrow \mathcal{X}'} = A^{t|\mathcal{X}'}$, we have the following proposition.

Proposition 6.14. Suppose $\mathcal{H}$ is a hypergraph on $\mathcal{X}$, t is a twig in $\mathcal{H}$, b is a branch for t, $\mathcal{X}'=\mathcal{X}-(t-b)$, A is a potential on $\mathcal{X}$, and $A = \Pi\{A_h | h\in\mathcal{H}\}$, where A_h is a potential on h. Then

$$A^{t|\mathcal{X}'} = (A_t^{t|t\cap b})^{\uparrow \mathcal{X}}. \tag{6.25}$$

So if we divide the potential on t by its projection to b and multiply the potential on b by this same projection, the result is a new factorization, such that the potential on t is $A^{t|\mathcal{X}'}$, and the potentials on the other hyperedges form a factorization of $A^{\downarrow \mathcal{X}'}$.

We can express (6.25) in words if A is a probability distribution. It says that when we divide the potential on the twig by its projection to its branch, we obtain the conditional probabilities for the twig given the variables that remain when the twig is removed.

In Lauritzen and Spiegelhalter's inward sweep, we perform the computation (6.25) first for the twig h_n in $\mathcal{H}$, then for the twig h_{n-1} in $\mathcal{H}-\{h_n\}$, and so on. The first step leaves $A^{h_n|h_1\cup\ldots\cup h_{n-1}}$ on h_n, and a factorization of $A^{\downarrow h_1\cup\ldots\cup h_{n-1}}$ on $\{h_1, \ldots, h_{n-1}\}$. The second step leaves $A^{h_{n-1}|h_1\cup\ldots\cup h_{n-2}}$ on h_{n-1}, and a factorization of $A^{\downarrow h_1\cup\ldots\cup h_{n-2}}$ on $\{h_1, \ldots, h_{n-2}\}$. And so on. So at the end of the inward sweep, we have the factorization

$$A = A^{\downarrow h_1}\, A^{h_2|h_1\cup h_2} \ldots A^{h_{n-1}|h_1\cup\ldots\cup h_{n-2}}\, A^{h_n|h_1\cup\ldots\cup h_{n-1}}. \tag{6.26}$$

Writing the factorization in this form makes it look more complex than it really is, of course; (6.25) tells us that $A^{h_i|h_1\cup\ldots\cup h_{i-1}}$ is carried by h_i, and we will have it stored at h_i, as a potential on h_i.

In the case where we are working with a probability distribution P, the language of conditional probability provides us with many different ways of describing the potential stored at h_i at the end of the inward sweep. Indeed, since $\beta(h_i) \twoheadrightarrow \perp_P [h_i, h_1 \cup \cdots \cup h_{i-1}]$, part (vii) of Proposition 6.9 tells us that $P^{h_i | h_1 \cup \ldots \cup h_{i-1}}$ is the vacuous extension of $P^{h_i | \beta(h_i)}$. And

$$P^{h_i | \beta(h_i)} = P^{h_i | h_i \cap \beta(h_i)} = P^{h_i - \beta(h_i) | h_i \cap \beta(h_i)}.$$

The first equality follows from $h_i \cap \beta(h_i) \twoheadrightarrow \perp_P [h_i, \beta(h_i)]$, the second from the fact that $A^{f|g} = A^{f-g|g}$ for any f and g.

As we see here, the very richness of the conditional probability language can be a source of some confusion. It is easy enough to say that the potentials on the vertices at the end of the inward sweep are conditional probabilities. But then we have to decide whether to call them conditional probabilities for the twig given the variables that remain when the twig is removed, conditional probabilities for the twig given the branch, conditional probabilities for the twig given the variables in both the twig and the branch, or conditional probabilities for the variables only in the twig given the variables in both the twig and the branch. All these descriptions are correct.

This confusing richness of the conditional probability language is one reason why we have avoided this language altogether in our initial description of Lauritzen and Spiegelhalter's algorithm.

If we write (6.26) as

$$P = P^{\downarrow h_1} P^{h_2 | h_2 \cap \beta(h_2)} \ldots P^{h_n | h_n \cap \beta(h_n)}, \tag{6.27}$$

it becomes clear that the outward sweep consists simply of repeated applications of the rule of total probability. Before the vertex h_i acts, its branch $\beta(h_i)$ has already computed $P^{\downarrow \beta(h_i)}$, marginalized this to $P^{\downarrow h_i \cap \beta(h_i)}$, and sent this marginal to h_i. Vertex h_i simply multiplies this message by its own potential, $P^{h_i | h_i \cap \beta(h_i)}$, obtaining

$$P^{\downarrow h_i \cap \beta(h_i)} P^{h_i | h_i \cap \beta(h_i)} = P^{\downarrow \beta(h_i)}. \tag{6.27}$$

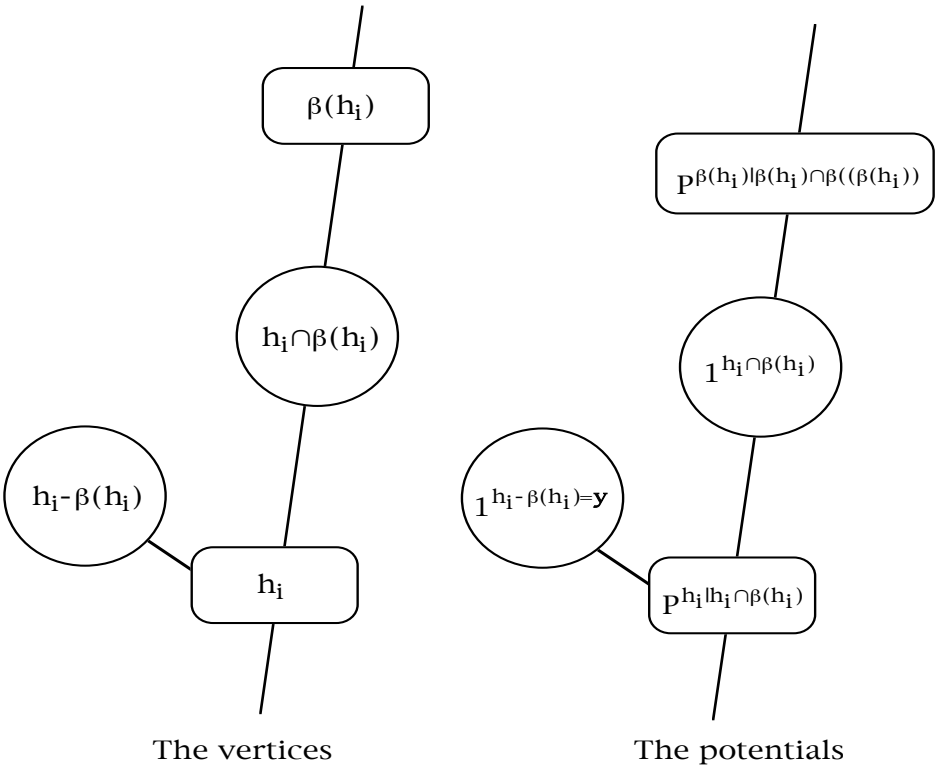
The action (6.27) on the outward sweep is obviously analogous to the action that we described in the probability tree in Figure 6.3 above. The h_i here are analogous to the $\{X_i, b(X_i)\}$ there, and the intersections $h_i \cap \beta(h_i)$ here are analogous to the single variables $\{X_i\}$ there.

To push this analogy further, consider the hypertree obtained by adding the intersections $h_i \cap \beta(h_i)$ and the differences $h_i - \beta(h_i)$ to $\mathcal{H}$. The sequence

$$h_1,\ h_2 \cap \beta(h_2),\ h_2,\ h_2 - \beta(h_2),\ \ldots,\ h_n \cap \beta(h_n),\ h_n,\ h_n - \beta(h_n),$$

with any duplications omitted, is a construction sequence for this hypertree. We can use $\beta(h_i)$ as the branch for $h_i \cap \beta(h_i)$, $h_i \cap \beta(h_i)$ as the branch for h_i, and h_i as the branch for $h_i - \beta(h_i)$. Figure 6.9 shows a fragment of the resulting Markov tree. If we put the potential $P^{\downarrow h_1}$ on h_1, the potentials $P^{h_i | h_i \cap \beta(h_i)}$ on the other h_i, and potentials identically equal to one on the other vertices, then running the algorithm of section 3.6 will reproduce Lauritzen and Spiegelhalter's outward sweep. If we replace the potentials on some of the $h_i - \beta(h_i)$ by indicator potentials, representing observations of some of the variables, then running the algorithm again will produce messages that are all likelihoods or probabilities, as in the case of the probability tree. If we have only one observation, say, then only about half the messages will need to be changed, so we can achieve our usual modest savings in updating.

Figure 6.9. Fragment of a Markov tree.



Though we have drawn an analogy to the probability tree of section 6.4, we do not have here the alluring unification of modeling and computation that we saw there. There the conditional probabilities we were using for the computation were assessed directly in the modeling and seemed to be permanent or stable features of the system. Here these conditional probabilities, since they are the result of the inward sweep, are more distant from the original modeling more likely to incorporate aspects of the evidence unique to the particular case.

6.6. Proofs

Proof of Proposition 6.1. To prove (i), we simply recognize that $G^{\downarrow h}(\mathbf{x}^{\downarrow h})$ is the sum of $G(\mathbf{x})$ together with $G(\mathbf{y})$ for all other $\mathbf{y}$ such that $\mathbf{y}^{\downarrow h}=\mathbf{x}^{\downarrow h}$; since G is a potential, all the terms in the sum are non–negative.

Statement (ii) follows immediately from statement (i).

Statement (iii) follows from (ii) together with our convention that a ratio with zero denominator is zero. It follows from this convention that a ratio will be zero if and only if its numerator, its denominator, or both are zero. Statement (ii) eliminates the possibility that the denominator of $G/G^{\downarrow h}$ can be zero without the numerator being zero, so this ratio is zero if and only if its numerator is zero.

Statement (iv) follows from (iii) and the fact that G is a potential. To see that (v) is true, first translate its hypothesis into the statement that $G(\mathbf{x})/G^{\downarrow h}(\mathbf{x}^{\downarrow h}) = B(\mathbf{x})$ for all $\mathbf{x}\in\mathcal{W}_g$. For $\mathbf{x}$ such that $G^{\downarrow h}(\mathbf{x}^{\downarrow h}) > 0$, it follows directly that $G(\mathbf{x}) = B(\mathbf{x})G^{\downarrow h}(\mathbf{x}^{\downarrow h})$. For $\mathbf{x}$ such that $G^{\downarrow h}(\mathbf{x}^{\downarrow h}) = 0$, we see from (ii) that $G(\mathbf{x}) = B(\mathbf{x})G^{\downarrow h}(\mathbf{x}^{\downarrow h})$ because both sides are zero. The proof of (vi) is analogous.

Proof of Proposition 6.2. The proof uses Proposition 3.1 once and Proposition 3.2 twice: $(GH)^{\downarrow g\cap h} = ((GH)^{\downarrow h})^{\downarrow g\cap h} = (G^{\downarrow g\cap h}H)^{\downarrow g\cap h} = G^{\downarrow g\cap h}H^{\downarrow g\cap h}$.

Proof of Proposition 6.3. When we write the ratio potentials in (ii), (iii), and (iv) in explicit form (i.e., write $A^{\downarrow h}/A^{\downarrow g\cap h}$ for $A^{h|g\cap h}$, etc.), we see each of these equations can be reduced to (i) by multiplying both sides by $A^{\downarrow g\cap h}$. So the equivalence of (i), (ii), (iii) and (iv) follows from parts (v) and (vi) of Proposition 6.1.

Statements (ii) and (iii) are factorizations of A on {g, h}. So in order to prove that statement (v) is equivalent to the first four statements, we need only derive one of the four from (v). We will derive (i). Assuming (v), we write A=GH, where G is an array on g, and H is an array on h. By Proposition 3.2, $A^{\downarrow g} = GH^{\downarrow g\cap h}$ and $A^{\downarrow h} = G^{\downarrow g\cap h}H$. By Proposition 6.2, $A^{\downarrow g\cap h} = G^{\downarrow g\cap h}H^{\downarrow g\cap h}$. So $AA^{\downarrow g\cap h} = GHG^{\downarrow g\cap h}H^{\downarrow g\cap h} = (GH^{\downarrow g\cap h})(G^{\downarrow g\cap h}H) = A^{\downarrow g}A^{\downarrow h}$.

???

If we write (vi) as $(A/A^{\downarrow h})^{\downarrow g} = A^{g|g\cap h}$, then we see its equivalence to (ii) by (iv) and (v) of Proposition 6.1. Similarly, (vii) is equivalent to (iii).

Finally, we prove the equivalence of (viii) and (vi); the equivalence of (ix) and (vii) is analogous. Suppose (viii) holds. Then we can write $A^{g|h} = B^{\uparrow g\cup h}$, or $A/A^{\downarrow h} = B^{\uparrow g\cup h}$, where B is a potential on g. By Proposition 6.1, this implies that $A = BA^{\downarrow h}$. Applying Proposition 3.1 to this equation, we obtain $A^{\downarrow g} = BA^{\downarrow g\cap h}$. Using Proposition 6.1 again, this time to divide both sides by $A^{\downarrow g\cap h}$, we obtain (vi).

Proof of Proposition 6.4. The sum of the values of $P^{|h=\mathbf{y}}$ is

$$\begin{aligned}\Sigma\{P^{|h=\mathbf{y}}(\mathbf{x}) \mid \mathbf{x}\in\mathcal{W}_\chi\} &= \Sigma\{P(\mathbf{x})I^{\downarrow h}(\mathbf{x}^{\downarrow h})/P^{\downarrow h}(\mathbf{x}^{\downarrow h}) \mid \mathbf{x}\in\mathcal{W}_\chi\}\\ &= \Sigma\{P(\mathbf{x})/P^{\downarrow h}(\mathbf{y}) \mid \mathbf{x}\in\mathcal{W}_\chi \text{ and } \mathbf{x}^{\downarrow h}=\mathbf{y}\}\\ &= \Sigma\{P(\mathbf{x}) \mid \mathbf{x}\in\mathcal{W}_\chi \text{ and } \mathbf{x}^{\downarrow h}=\mathbf{y}\}\ /P^{\downarrow h}(\mathbf{y})\\ &= P^{\downarrow h}(\mathbf{y}) / P^{\downarrow h}(\mathbf{y}) = 1.\end{aligned}$$

Proof of Proposition 6.5. To see that (ii) and (iii) are equivalent, notice that (ii) says that $P=P/P^{\downarrow f}$, and by (v) and (vi) of Proposition 6.1, this is equivalent to (iii).

To see that (iii) and (i) are equivalent, notice that the relation $P(\mathbf{y})=P(\mathbf{y})P^{\downarrow f}(\mathbf{y}^{\downarrow f})$ can hold for all $\mathbf{y}$ if and only if $P^{\downarrow f}(\mathbf{y}^{\downarrow f})$ is equal to one whenever $P(\mathbf{y})>0$. Since a probability distribution can assign the value one to at most one configuration, this is equivalent to (i).

Proof of Proposition 6.7. The equivalence of (i) and (ii) is obvious.

The equivalence of (ii) and (iii) follows from the definition of conditional probability; if $\delta(\mathbf{x})^{\downarrow g}=\mathbf{x}$ and $P^{\downarrow g}(\mathbf{x})>0$, then $P^{g\cup h|g=\mathbf{x}}(\delta(\mathbf{x})) = P^{\downarrow g\cup h}(\delta(\mathbf{x})) / P^{\downarrow g}(\mathbf{x})$, so $P^{g\cup h|g=\mathbf{x}}(\delta(\mathbf{x})) = 1$ is equivalent to $P^{\downarrow g\cup h}(\delta(\mathbf{x})) = P^{\downarrow g}(\mathbf{x})$.

The equivalence of (iii) and (iv) follows from the definition of marginal probability. This definition says that

$$P^{\downarrow g}(\mathbf{x}) = \Sigma\{P^{\downarrow g\cup h}(\mathbf{y}) \mid \mathbf{y}\in\mathcal{W}_{g\cup h} \text{ and } \mathbf{y}^{\downarrow g}=\mathbf{x}\}.$$

Since all the terms in the sum are non-negative, only one of these terms being positive is equivalent to the existence of one that is equal to the sum.

We leave to the reader the proof that (v) is also equivalent to the preceding statements.

Proof of Proposition 6.8. Both $P^{h \cup f|g}$ and $P^{h|g \cup f}$ have the domain $f \cup g \cup h$, and both take positive values only for those $\mathbf{z} \in \mathcal{W}_{f \cup g \cup h}$ such that $P^{\downarrow f \cup g \cup h}(\mathbf{z}) > 0$. By part (v) of Proposition 6.7, $P^{\downarrow g \cup f}(\mathbf{z}) = P^{\downarrow g}(\mathbf{z})$ and hence

$$P^{h \cup f|g}(\mathbf{z}) = P^{\downarrow f \cup g \cup h}(\mathbf{z}) \,/\, P^{\downarrow g}(\mathbf{z}) = P^{\downarrow f \cup g \cup h}(\mathbf{z}) \,/\, P^{\downarrow g \cup f}(\mathbf{z}) = P^{h|g \cup f}(\mathbf{z})$$

for any such $\mathbf{z}$.

Proof of Proposition 6.13. The proof is based on a comparison between Lauritzen and Spiegelhalter's algorithm (algorithm L&S) and the algorithm for simultaneous computation given in section 3.6 (algorithm SM, for Simultaneous computation of Marginals). We assume that both algorithms use the construction sequence $h_1 h_2 \ldots h_n$ for both the inward and outward sweep.

The two algorithms send the same messages inward, and they both incorporate messages received by multiplication, so as the propagation proceeds they will agree on the potential stored at any vertex that has received but not yet sent messages. At the end of the inward sweep, the root h_1 has not yet sent a message; this is how we know its potential under L&S is the same as its potential under SM, $A^{\downarrow h_1}$.

Let us step back a moment, to the point just before the end of the inward sweep, just before h_2 sends a message to h_1. At that point, neither h_1 nor h_2 has sent a message, and the same potentials have arrived at the two vertices under L&S as under SM. The vertex h_1 has its original potential and messages from all its neighbors except h_2. Under SM, these potentials are all stored; under L&S, their product

$$B_1 = A_{h_1} \Pi\{M^{g \to h_1} \mid g \in (\mathcal{N}_{h_1} - \{h_2\})\}.$$

is stored. Similarly, h_2 has its original potential and messages from all its neighbors except h_1. Under SM, these are all stored; under L&S their product

$$B_2 = A_{h_2} \Pi\{M^{g \to h_2} \mid g \in (\mathcal{N}_{h_2} - \{h_1\})\}.$$

is stored.

Figure 6.10 shows what happens next under SM. First, the inward sweep is completed as h_2 marginalizes B_2 to $h_2 \cap h_1$. This is the end of the inward sweep, and h_1 is now able to multiply all its messages together, obtaining its marginal, $B_1 B_2^{\downarrow h_1 \cap h_2}$. Then, as the first step of the outward sweep, h_1 multiplies everything

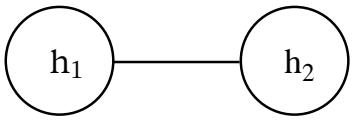
except the message from h_1, obtaining B_1, and sends the message $B_1^{\downarrow h_1 \cap h_2}$ to h_2. Now h_2 has also received all its messages; it multiplies them all together, obtaining its marginal, $B_2 B_1^{\downarrow h_1 \cap h_2}$.

Figure 6.11 shows what happens next under L&S. Initially we have the product B_1 at h_1 and the product B_2 at h_2. The inward sweep is completed as h_2 marginalizes B_2 to $h_1 \cap h_2$. The message, $B_2^{\downarrow h_1 \cap h_2}$ multiplies the potential at h_1 and divides the potential at h_2, leaving $B_1 B_2^{\downarrow h_1 \cap h_2}$ at h_1 and $B_2/B_2^{\downarrow h_1 \cap h_2}$ at h_1. Then, as the first step of the outward sweep, h_1 marginalizes its current potential, $B_1 B_2^{\downarrow h_1 \cap h_2}$, to h_2. The marginal multiplies the potential already at h_2, yielding, by Propositions 3.2 and 6.1,

$$\begin{aligned}(B_1 B_2^{\downarrow h_1 \cap h_2})^{\downarrow h_1 \cap h_2} B_2/B_2^{\downarrow h_1 \cap h_2} &= B_1^{\downarrow h_1 \cap h_2} B_2^{\downarrow h_1 \cap h_2} B_2/B_2^{\downarrow h_1 \cap h_2} \\ &= B_1^{\downarrow h_1 \cap h_2} B_2,\end{aligned}$$

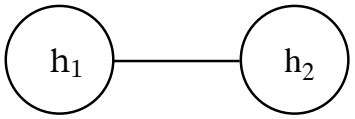
the marginal on h_2.

Figure 6.10. Algorithm SM.



	Product of the potentials stored-	
	-at h_i:	-at h_2:
Just before the last step of inward sweep.	B_1	B_2
At end of inward sweep.	$B_1 B_2^{\rightarrow h_1}$	B_2
Just before h_1 projects back outward to h_2.	$B_1 B_2^{\rightarrow h_1}$	$B_2 B_1^{\rightarrow h_2}$

Figure 6.11. Algorithm L&S.



	Product of the potentials stored- -at h_i:	-at h_2:
Just before the last step of inward sweep.	B_1	B_2
At end of inward sweep.	$B_1B_2^{\rightarrow h_1}$	$B_2/B_2^{\rightarrow h_1}$
Just before h_1 projects back outward to h_2.	$B_1B_2^{\rightarrow h_1}$	$(B_2/B_2^{\rightarrow h_2})(B_1B_2^{\rightarrow h_1})^{\rightarrow h_2}$

We have established that L&S produces the marginal on the first vertex to which it projects in its outward sweep. But the same argument also works for any other vertex. The situation just before g projects to h on the outward sweep will be the same no matter whether this is the first projection on the outward sweep or a later one. In either case, the situation will be described by the middle lines in Figures 6.10 and 6.11. The vertex g will have all its messages (under SM) or their product (under L&S). The vertex h will have all its messages except the message from g or their product divided by the message h sent to g (under L&S).

CHAPTER SEVEN

An Axiomatic Framework for Discrete Optimization

The main objective of this chapter is to describe an axiomatic paper for representing and solving discrete optimization problems. There are several reasons why this is useful.

First, in chapter 4, we described an axiomatic framework for computing the marginals of the joint valuation in a valuation-based system (VBS). In this chapter, I show that these systems also have the expressive power to represent and solve optimization problems.

Second, problems in Bayesian decision analysis involve managing probabilities and optimization. That these problems can be solved in a common framework suggests that Bayesian decision problems also can be represented and solved in the framework of VBS. Indeed, Shenoy [1990a,b, 1991] shows that this is true. In fact, the solution procedure of VBSs when applied to symmetric Bayesian decision problems results in a method that is computationally more efficient than decision trees and influence diagrams.

Third, the solution procedure of VBS when applied to optimization problems results in a method called non-serial dynamic programming [Bertele and Brioschi, 1972]. Thus in an abstract sense, the local computation algorithms that have been described by, for example, Pearl [1986], Shenoy and Shafer [1986], Dempster and Kong [1988], Lauritzen and Spiegelhalter [1988], and Shafer and Shenoy [1990] are just dynamic programming.

Fourth, we provide an answer to the question: What is dynamic programming? Dynamic programming is commonly viewed as an optimization technique. This is how Bellman [1957] described it. However, it is also recognized that dynamic programming is more than an optimization technique. For example, Aho, Hopcroft and Ullman [1974] refer to dynamic programming as a "divide-and-conquer" methodology. In this paper, we give a formal definition of a problem and a formal method solving the problem. The formal method for solving the problem can be thought of as an abstract definition of dynamic programming solution methodology.

Fifth, we provide an answer to the question: When does dynamic programming work? We describe some simple axioms for combination and marginalization that enable the use of dynamic programming for solving optimization problems. We believe these axioms are new. They are weaker than those proposed by Mitten [1964].

Sixth, the VBS described here can be easily adapted to represent propositional logic [Shenoy 1990c,d] and constraint satisfaction problems [Shenoy and Shafer, 1988]. Constraint satisfaction problems is dealt with in chapter 8 of this book.

An outline of this chapter is as follows. In section 7.1, we show how to represent an optimization problem as a VBS. In section 7.2, we state some simple axioms that justify the use of local computation in solving VBSs. In section 7.3, we show how to solve a VBS. Throughout the paper, we use one example to illustrate all definitions and the solution method. In section 7.4, we compare our axioms to those proposed by Mitten [1964] for serial dynamic programming. In section 7.5, we describe two different applications of the axiomatic framework. In section 7.6, we make some concluding remarks. Finally, in section 7.7 we provide proofs for all results in the paper.

Most of the material in this chapter previously appeared in [Shenoy, 1991].

7.1. The Axiomatic Framework

A valuation-based system representation of an optimization problem uses variables, frames, and valuations. We will discuss each of these in detail. We will illustrate all definitions using an optimization problem from Bertele and Brioschi [1972].

An Optimization Problem. There are five variables labeled as A, B, C, D, and E. Each variable has two possible values. Let a and ~a denote the possible values of A, etc. The joint objective function F for variables A, B, C, D, and E factors additively as follows: $F(v, w, x, y, z) = F_1(v, x, z) + F_2(v, w) + F_3(w, y, z)$, where F_1, F_2, and F_3, are as shown below in Figure 7.1. The problem is to find the minimum value of F and a configuration (v, w, x, y, z) that minimizes F.

Figure 7.1. The factors of the objective function, F_1, F_2, and F_3.

$\mathbf{w}\in W_{\{A,C,E\}}$	$F_1(\mathbf{w})$
a c e	1
a c ~e	3
a ~c e	5
a ~c ~e	8
~a c e	2
~a c ~e	6
~a ~c e	2
~a ~c ~e	4

$\mathbf{w}\in W_{\{A,B\}}$	$F_2(\mathbf{w})$
a b	4
a ~b	8
~a b	0
~a ~b	5

$\mathbf{w}\in W_{\{B,D,E\}}$	$F_3(\mathbf{w})$
b d e	0
b d ~e	5
b ~d e	6
b ~d ~e	3
~b d e	5
~b d ~e	1
~b ~d e	4
~b ~d ~e	3

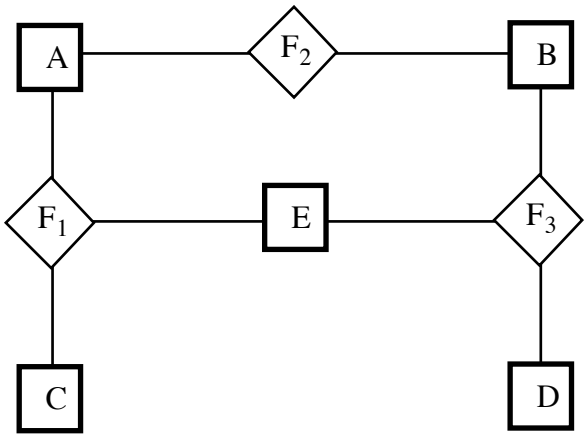
Values and Valuations. We are concerned with a set V whose elements are called *values*. V may be finite or infinite. Given a set h of variables, we call any function $H: W_h \to V$, a *valuation for h*. Note that to specify a valuation for ∅, we need to specify only a single value, H(♦). If H is a valuation for h and X∈h, then we say H *bears on* X.

In our problem, the set V corresponds to the set of real numbers, and we have three valuations F_1, F_2 and F_3. F_1 is a valuation for {A, C, E}, F_2 is a valuation for {A, B} and F_3 is a valuation for {B, D, E}. Figure 7.2 shows a graphical

depiction of the optimization problem. We call such a graph a *valuation network*. In a valuation network, square nodes represent variables, and diamond-shaped nodes represent valuations. Each valuation is linked to the variables it bears on.

Let $\mathcal{V}_h$ denote the set of valuations for h, and let $\mathcal{V}$ denote the set of valuations, i.e., $\mathcal{V} = \cup\{\mathcal{V}_h \mid h \subseteq \mathcal{X}\}$.

Figure 7.2. The valuation network for the optimization problem.



Combination. We assume there is a mapping $\copyright: \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V}$ called *combination* so that if u, v $\in \mathcal{V}$, then u©v is the value representing the combination of u and v. We define a mapping $\oplus: \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V}$ in terms of ©, also called *combination*, such that if G and H are valuations for g and h, respectively, then G⊕H is the valuation for g∪h given by

$$(G \oplus H)(\mathbf{x}) = G(\mathbf{x}^{\downarrow g}) \copyright H(\mathbf{x}^{\downarrow h}) \tag{7.1}$$

for all $\mathbf{x} \in \mathcal{W}_g$. We call G⊕H the *combination of G and H*.

In our optimization problem, © is simply addition, i.e.

$$(G \oplus H)(\mathbf{x}) = G(\mathbf{x}^{\downarrow g}) + H(\mathbf{x}^{\downarrow h}) \tag{7.2}$$

Using (7.2), we can express the joint objective function F as follows $F = F_1 \oplus F_2 \oplus F_3$.

Marginalization. We assume that for each $h \subseteq \mathcal{X}$, there is a mapping $\downarrow h: \cup\{\mathcal{V}_g \mid g \supseteq h\} \rightarrow \mathcal{V}_h$, called *marginalization to h*, such that if G is a valuation for g and $g \supseteq h$, then $G^{\downarrow h}$ is a valuation for h. We call $G^{\downarrow h}$ the *marginal of G for h*.

For our optimization problem, we define marginalization as follows:

$$G^{\downarrow h}(\mathbf{x}) = \text{MIN}\{G(\mathbf{x}, \mathbf{y}) \mid \mathbf{y} \in \mathcal{W}_{g-h}\} \tag{7.3}$$

for all $\mathbf{x} \in \mathcal{W}_h$. Thus, if F is an objective function, then $F^{\downarrow \varnothing}(\blacklozenge)$ represents the minimum value of F.

In an optimization problem, besides the minimum value, we are usually also interested in finding a configuration where the minimum of the joint valuation is achieved. This motivates the following definition.

Solution for a Valuation. Suppose H is a valuation for h. We call $\mathbf{x} \in \mathcal{W}_h$ a *solution for H* if $H(\mathbf{x}) = H^{\downarrow \varnothing}(\blacklozenge)$.

Solution for a Variable. As we will see, once we have computed the minimum value of a valuation, computing a solution for the valuation is a matter of bookkeeping. Each time we eliminate a variable from a valuation using minimization, we store a table of configurations of the eliminated variable where the minimums are achieved. We can think of this table as a function. We call this function "a solution for the variable." Formally, we define a solution for a variable as follows. Suppose X is a variable, suppose g is a subset of variables containing X, and suppose G is a valuation for g. We call a function $\Psi_X: \mathcal{W}_{g-\{X\}} \rightarrow \mathcal{W}_X$ a *solution for X (with respect to G)* if

$$G^{\downarrow (g-\{X\})}(\mathbf{c}) = G(\mathbf{c}, \Psi_X(\mathbf{c})) \tag{7.4}$$

for all $\mathbf{c} \in \mathcal{W}_{g-\{X\}}$.

If $\mathcal{X}$ is a large set of variables, then a brute force computation of F and an exhaustive search of the set of all configurations of $\mathcal{X}$ to determine a solution for F is computationally infeasible. In the next section we will state axioms for combination and marginalization that make it possible to use local computation to compute the minimum value of F and a solution for F.

7.2 The Axioms

We state three axioms. Axiom A1' is for combination. Axiom A2 is for marginalization. And Axiom A3 is for combination and marginalization. Axioms A2 and A3 are the same as those defined in Chapter 4.

> **A1'.** (*Commutativity and associativity of combination*). Suppose u, v, and w are values. Then u©w = v©u and u©(v©w) = (u©v)©w.
>
> **A2.** (*Consonance of marginalization*). Suppose G is a valuation for g, and k⊆h⊆g. Then $(G^{\downarrow h})^{\downarrow k} = G^{\downarrow k}$.
>
> **A3.** (*Distributivity of marginalization over combination*). Suppose G and H are valuations for g and h, respectively. Then $(G \oplus H)^{\downarrow g} = G \oplus (H^{\downarrow g \cap h})$.

Note that axiom A1' implies axiom A1 defined in Chapter 4, i.e., if axiom A1' holds, then ⊕ is commutative and associative. Therefore, the combination of several valuations can be written without using parentheses. For example, $(...((F_1 \oplus F_2) \oplus F_3) \oplus ... \oplus F_k)$ can be simply written as $F_1 \oplus ... \oplus F_k$ without specifying the order in which to do the combination.

If we regard marginalization as a reduction of a valuation by deleting variables, then axiom A2 can be interpreted as saying that the order in which we delete the variables does not matter.

Axiom A3 is the crucial axiom that makes local computation of marginals and solution possible. Axiom A3 states that computation of $(G \oplus H)^{\downarrow g}$ can be done without having to compute G⊕H.

For our optimization problem, it is easy to see that the definitions of combination in (7.2) and marginalization in (7.3) satisfy the three axioms.

7.3. Solving a VBS Using Local Computation

Suppose we are given a VBS consisting of a collection of valuations $\{F_1, ..., F_k\}$ where each valuation F_i is for subset h_i of $\mathcal{X}$. The problem is (i) to find the value of $F^{\downarrow\varnothing}(\blacklozenge) = (F_1 \oplus ... \oplus F_k)^{\downarrow\varnothing}(\blacklozenge)$ and (ii) to find a solution for F. We assume that combination and marginalization satisfy the three axioms.

In the case of an optimization problem, $F^{\downarrow\varnothing}(\blacklozenge)$ represents the minimum value of the joint objective function, and a solution for F represents a configuration of all variables where the minimum is achieved.

Solving a VBS proceeds in three phases. In phase one, we arrange the subsets of variables in $\mathcal{H}$ in a "rooted Markov tree." In the phase two, we "propagate" the valuations $\{F_1, ..., F_k\}$ in the rooted Markov tree using a local message-passing scheme resulting in the computation of the marginal $F^{\downarrow\varnothing}$. In the phase three, we construct a solution for F again using a local message-passing scheme.

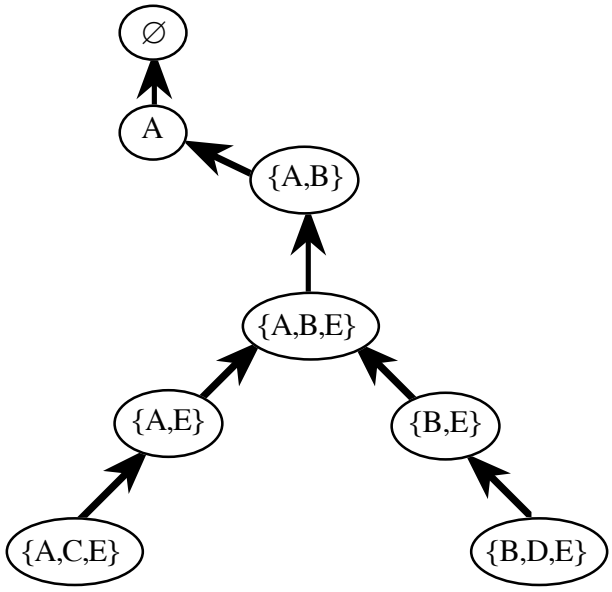
7.3.1. Phase One: Finding a Rooted Markov Tree Arrangement

A *rooted Markov tree* is a Markov tree with the empty subset $\varnothing$ as the root and such that all edges in the tree are directed toward the root. Figure 7.3 shows a rooted Markov tree arrangement of the subsets {A, C, E}, {A, B}, and {B, D, E}.

The computational efficiency of phase two depends on the sizes of the frames of the vertices of the Markov tree constructed in the phase one. Finding an optimal rooted Markov tree (a rooted Markov tree whose largest frame is as small as possible) has been shown to be a NP-complete problem [Arnborg *et al.*, 1987]. Thus we have to balance the computational efforts in the two phases. We will describe a heuristic called "one-step-look-ahead" due to Kong [1986] to find a good rooted Markov tree.

The method described below for arranging a hypergraph in a rooted Markov tree is due to Kong [1986] and Mellouli [1987].

Figure 7.3. A rooted Markov tree for the optimization problem.



Suppose $\mathcal{H}$ is a hypergraph on $\mathcal{X}$. To arrange the subsets in $\mathcal{H}$ in a Markov tree, we first pick a sequence of variables in $\mathcal{X}$. As we will see, each sequence of the variables gives rise to a Markov tree arrangement. Mellouli [1987] has shown that an optimal Markov tree arrangement can be found by picking some sequence. Of course, since there are an exponential number of sequences, finding an optimal sequence is, in general, a difficult problem.

Suppose we have a sequence of variables. Consider the first variable, say X_1, in the sequence. We add two subsets $g_1 = \cup\{h \mid X_1 \in h\}$ and $f_1 = g_1 - \{X_1\}$ to $\mathcal{H}$. We form the rooted Markov tree $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{h \in \mathcal{H} \mid X_i \in h\} \cup \{f_i\} \cup \{g_i\}$ and $\mathcal{E} = \{(h, g_i) \mid h \in (\mathcal{H} - \{g_i, f_i\}), X_i \in h\} \cup \{(g_i, f_i)\}$. We now consider X_1 as *marked* and subsets that contain X_1 as *arranged*. We repeat this process for the unarranged subsets until all variables are marked.

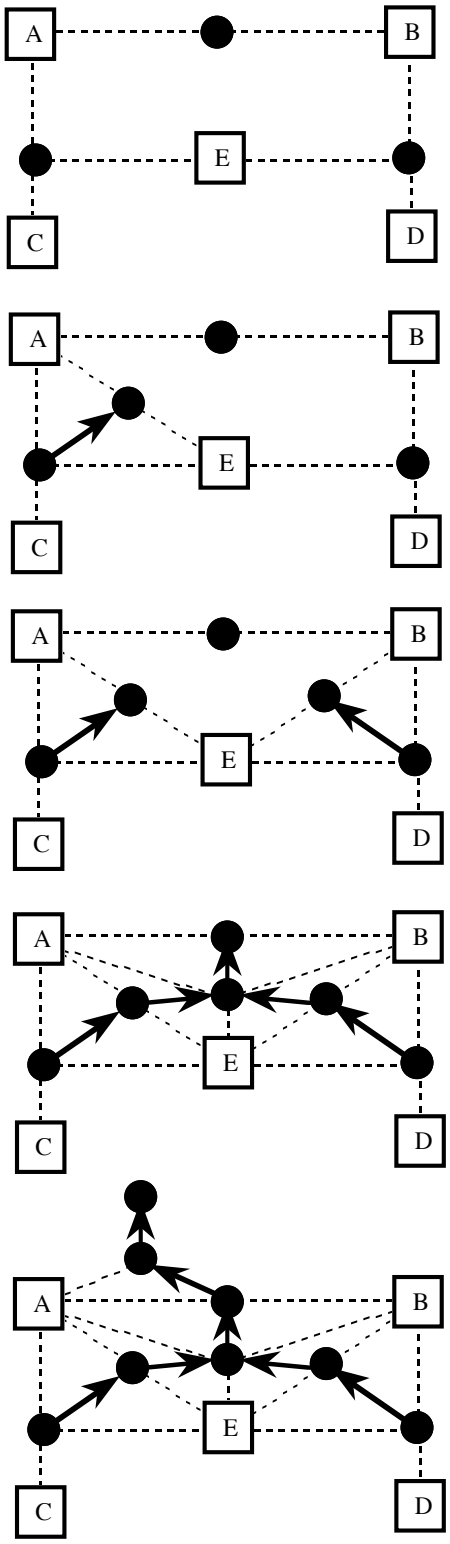
Consider the following set of instructions in pseudo-Pascal:

u := $\mathcal{X}$ {Initialization}
$\mathcal{H}_0 := \mathcal{H}$ {Initialization}
$\mathcal{V} := \varnothing$ {Initialization}
$\mathcal{E} := \varnothing$ {Initialization}
for i = 1 **to** n **do**
begin
Pick a variable from set u and call it X_i
u := u – $\{X_i\}$
$g_i := \cup\{h \in \mathcal{H}_{i-1} \mid X_i \in h\}$.
$f_i := g_i - \{X_i\}$.
$\mathcal{V} := \mathcal{V} \cup \{h \in \mathcal{H}_{i-1} \mid X_i \in h\} \cup \{f_i\} \cup \{g_i\}$
$\mathcal{E} := \mathcal{E} \cup \{(h, g_i) \mid h \in (\mathcal{H}_{i-1} - \{g_i, f_i\}), X_i \in h\} \cup \{(g_i, f_i)\}$
$\mathcal{H}_i := \{h \in \mathcal{H}_{i-1} \mid X_i \notin h\} \cup \{f_i\}$
end {for}

After the execution of the above set of instructions, it is easily seen that the pair ($\mathcal{V}$, $\mathcal{E}$) is a rooted Markov tree arrangement of $\mathcal{H}$ where $\mathcal{V}$ denotes the set of vertices of the rooted Markov tree and $\mathcal{E}$ denotes the set of edges directed toward the root.

Kong [1986] has suggested a heuristic called *one-step-look-ahead* for finding a good Markov tree. This heuristic tells us which variable to mark next. As the name of the heuristic suggests, the variable that should be marked next is an unmarked variable X_i such that the cardinality of $\mathcal{W}_{f_i}$ is the smallest. Thus, the heuristic attempts to keep the sizes of the frames of the added vertices as small as possible by focusing only on the next subset added. In the optimization problem, a marking sequence selected by the one-step-look-ahead heuristic is CDEBA. Figure 7.4 illustrates the construction of a rooted Markov tree using this marking sequence. The resulting Markov tree is the same as that shown in Figure 7.3. See Zhang [1988] for other heuristics for good Markov tree construction.

Figure 7.4. The construction of the rooted Markov tree for the optimization problem.



1. The initial hypergraph. Variables are shown as squares and subsets are shown as black disks. The elements of each subset are indicated by dotted lines.

2. The Markov tree fragment after C is marked. Subset {A,E} is added to the hypergraph. Subset {A,C,E} is now arranged.

3. The Markov tree fragment after D is marked. Subset {B,E} is added to the hypergraph. Subset {B,D,E} is now arranged.

4. The Markov tree fragment after E is marked. Subset {A,B,E} is added to the hypergraph. Subsets {A,E}, {B.E} and {A,B,E} are now arranged.

5. The Markov tree fragment after B and then A are marked. Subsets {A} and ∅ are added to the hypergraph. All subsets are now arranged.

7.3.2. Phase Two: Finding the Marginal of the Joint Valuation

Suppose we have arranged the hypergraph $\mathcal{H}$ in a rooted Markov tree. Let $\mathcal{H}'$ denote the set of subsets in the Markov tree. Clearly $\mathcal{H}' \supseteq \mathcal{H}$. To simplify the exposition, we assume there is exactly one valuation for each nonempty subset $h \in \mathcal{H}'$. If h is a subset that was added during the rooted Markov tree construction process, then we can associate the vacuous valuation (the valuation whose values are all 0) with it. On the other hand, if subset h had more than one valuation defined for it, then we can combine these valuations to obtain one valuation.

If we assume that the directed edges of a rooted Markov tree point from a child to its parent, then the rooted Markov tree defines a parent-child relation between adjacent vertices. If there is an edge (h_i, h_j) in the rooted Markov tree, we refer to h_j as h_i's *parent* and refer to h_i as h_j's *child*. Let $h_0 = \varnothing$ denote the *root* of the Markov tree. Let Pa(h) denote h's parent and let Ch(h) denote the set of h's children. Every non-root vertex has exactly one parent. Some vertices have no children and we call such vertices *leaves*. Note that the root has exactly one child.

In describing the process of finding the marginal of the joint valuation for the empty set, we will pretend that there is a processor at each vertex of the rooted Markov tree. Also, we assume these processors are connected using the same architecture as the Markov tree. In other words, each processor can directly communicate only with its parent and its children.

In the propagation process, each subset (except the root h_0) transmits a valuation to its parent. We call the valuation transmitted by subset h_i to its parent $Pa(h_i)$ a *valuation message* and denote it by $V^{h_i \to Pa(h_i)}$. Suppose $\mathcal{H}' = \{h_0, h_1, ..., h_k\}$ and let F_i denote the valuation associated with nonempty subset h_i. Then, the valuation message transmitted by a subset h_i to its parent $Pa(h_i)$ is given by

$$V^{h_i \to Pa(h_i)} = (\oplus\{V^{h \to h_i} \mid h \in Ch(h_i)\} \oplus F_i)^{\downarrow (h_i \cap Pa(h_i))} \qquad (7.5)$$

In words, the valuation message transmitted by a subset to its parent consists of the combination of the valuation messages it receives from its children plus its own valuation suitably marginalized. Note that the combination operation in (7.5) is on the frame $\mathcal{W}_{h_i}$.

Expression (7.5) is a recursive formula. We need to start the recursion somewhere. Note that if subset h_i has no children, then $Ch(h_i) = \varnothing$ and the expression in (7.5) reduces to

$$V^{h_i \to Pa(h_i)} = (F_i)^{\downarrow (h_i \cap Pa(h_i))} \tag{7.6}$$

Thus the leaves of the Markov tree (the subsets that have no children) can send valuation messages to their parents right away. The others wait until they have heard from all their children before they send a valuation message to their parent.

The following theorem states that the valuation message from h_0's child to h_0 is indeed the desired marginal.

> ***Theorem 7.1.*** The marginal of the joint valuation for the empty set is equal to the message received by the root, i.e., $(F_1 \oplus ... \oplus F_k)^{\downarrow \varnothing} = V^{h \to h_0}$.

Theorem 7.1 is valid not only for optimization problems but for any VBS where axioms A1', A2, A3 hold. We give a simple proof of Theorem 7.1 in section 7.6.

The essence of the propagation method described above is to combine valuations on smaller frames instead of combining all valuations on the global frame associated with X. To ensure that this method gives us the correct answers, the smaller frames have to be arranged in a rooted Markov tree.

Figure 7.5 shows the propagation of valuations in the optimization problem. Figure 7.6 shows the details of the valuation messages. As is clear from Figure 7.6, the minimum value of the joint objective function F is 2.

Figure 7.5. The propagation of valuations in the optimization problem. The valuation messages are shown as rectangles overlapping the corresponding edges. The valuations associated with the vertices are shown as diamonds linked to the corresponding vertices by dotted lines.

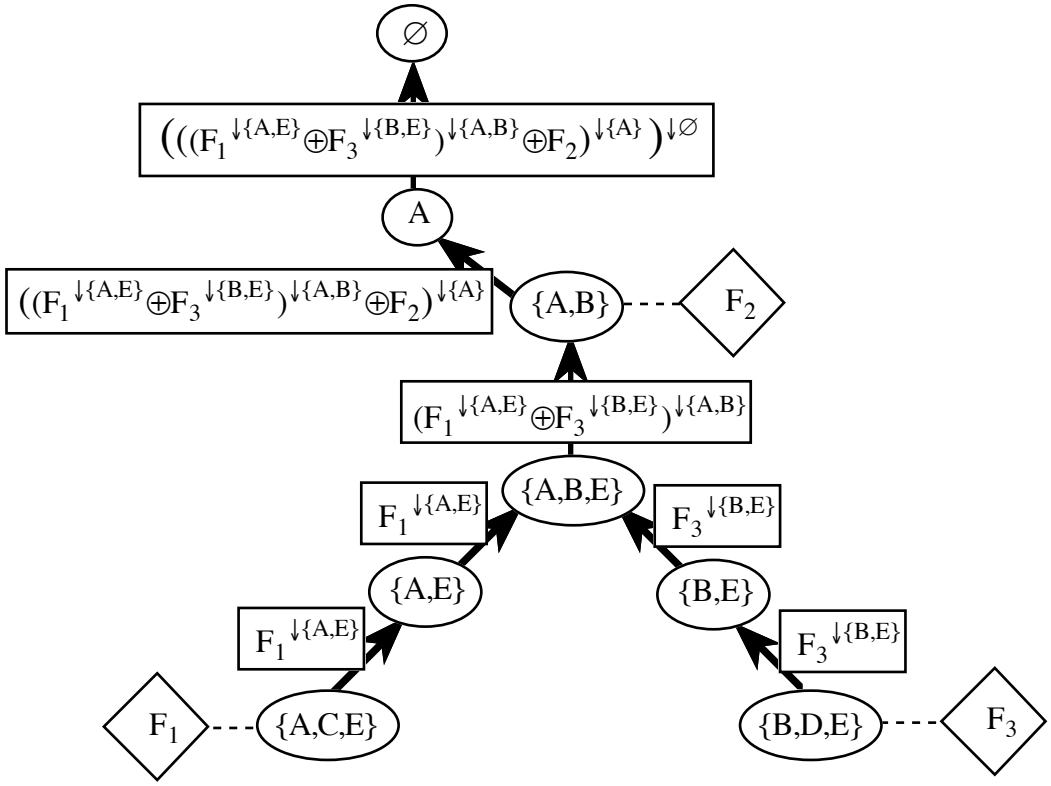


Figure 7.6. The details of the valuation messages for the optimization problem.

$w_{\{A,C,E\}}$	F_1
a c e	1
a c ~e	3
a ~c e	5
a ~c ~e	8
~a c e	2
~a c ~e	6
~a ~c e	2
~a ~c ~e	4

$w_{\{A,E\}}$	$F_1^{\downarrow\{A,E\}}$	Ψ_C
a e	1	c
a ~e	3	c
~a e	2	c or ~c
~a ~e	4	~c

$w_{\{B,D,E\}}$	F_3
b d e	0
b d ~e	5
b ~d e	6
b ~d ~e	3
~b d e	5
~b d ~e	1
~b ~d e	4
~b ~d ~e	3

$w_{\{B,E\}}$	$F_3^{\downarrow\{B,E\}}$	Ψ_D
b e	0	d
b ~e	3	~d
~b e	4	~d
~b ~e	1	d

$w_{\{A,B,E\}}$	$F_1^{\downarrow\{A,E\}}$	$F_3^{\downarrow\{B,E\}}$	$F_1^{\downarrow\{A,E\}} \oplus F_3^{\downarrow\{B,E\}}$
a b e	1	0	1
a b ~e	3	3	6
a ~b e	1	4	5
a ~b ~e	3	1	4
~a b e	2	0	2
~a b ~e	4	3	7
~a ~b e	2	4	6
~a ~b ~e	4	1	5

$w_{\{A,B\}}$	$(F_1^{\downarrow\{A,E\}} \oplus F_3^{\downarrow\{B,E\}})^{\downarrow\{A,B\}}$	Ψ_E
a b	1	e
a ~b	4	~e
~a b	2	e
~a ~b	5	~e

$w_{\{A,B\}}$	$(F_1^{\downarrow\{A,E\}} \oplus F_3^{\downarrow\{B,E\}})^{\downarrow\{A,B\}}$	F_2	$(F_1^{\downarrow\{A,E\}} \oplus F_3^{\downarrow\{B,E\}})^{\downarrow\{A,B\}} \oplus F_2$
a b	1	4	5
a ~b	4	8	12
~a b	2	0	2
~a ~b	5	5	10

$w_{\{A\}}$	$((F_1^{\downarrow\{A,E\}} \oplus F_3^{\downarrow\{B,E\}})^{\downarrow\{A,B\}} \oplus F_2)^{\downarrow\{A\}}$	Ψ_B
a	5	b
~a	2	b

$w_{\varnothing}$	$(((F_1^{\downarrow\{A,E\}} \oplus F_3^{\downarrow\{B,E\}})^{\downarrow\{A,B\}} \oplus F_2)^{\downarrow\{A\}})^{\downarrow\varnothing}$	Ψ_A
◆	2	~a

7.3.3. Phase Three: Finding a Solution

In phase two, each time we marginalize a variable, assume that we store the corresponding solution for that variable at the vertex where we do the marginalization. For example, in the optimization problem, we store a solution for C at vertex {A, C, E}, we store a solution for D at vertex {B, D, E}, we store a solution for E at vertex {A, B, E}, we store a solution for B at vertex {A, B}, and we store a solution for A at vertex {A} (see Figures 4, 5, and 6).

In this phase, each vertex of the rooted Markov tree sends a configuration to each of its children. We call the configuration transmitted by vertex h_i to its child $h_j \in Ch(h_i)$ as a *configuration message* and denote it by $\mathbf{c}^{h_i \to h_j}$. $\mathbf{c}^{h_i \to h_j}$ will always be an element of $\mathcal{W}_{h_i \cap h_j}$. As in phase two, we give a recursive definition of configuration messages.

The messages start at the root and travel toward the leaves. The configuration message from vertex $\varnothing$ to its child, say h_1, is given by

$$\mathbf{c}^{\varnothing \to h_1} = \blacklozenge. \tag{7.7}$$

In general, consider vertex h_i. It receives a configuration message $\mathbf{c}^{Pa(h_i) \to h_i}$ from its parent $Pa(h_i)$. Let h_j be a child of h_i. The configuration message from h_i to h_j depends on whether h_i has a solution for a variable stored at its location. (Remember that vertex h_i has a solution for X stored with it if $h_i – Pa(h_i) = \{X\}$).

If h_i has a solution for a variable stored at its location, then

$$\mathbf{c}^{h_i \to h_j} = (\mathbf{c}^{Pa(h_i) \to h_i}, \Psi_X(\mathbf{c}^{Pa(h_i) \to h_i}))^{\downarrow (h_i \cap h_j)} \tag{7.8}$$

where X is such that $\{X\} = h_i – Pa(h_i)$.

If h_i has no solution for a variable stored at its location, then

$$\mathbf{c}^{h_i \to h_j} = (\mathbf{c}^{Pa(h_i) \to h_i})^{\downarrow (h_i \cap h_j)}. \tag{7.9}$$

We stop the message passing process when each vertex that has a solution stored at its location has received a configuration message.

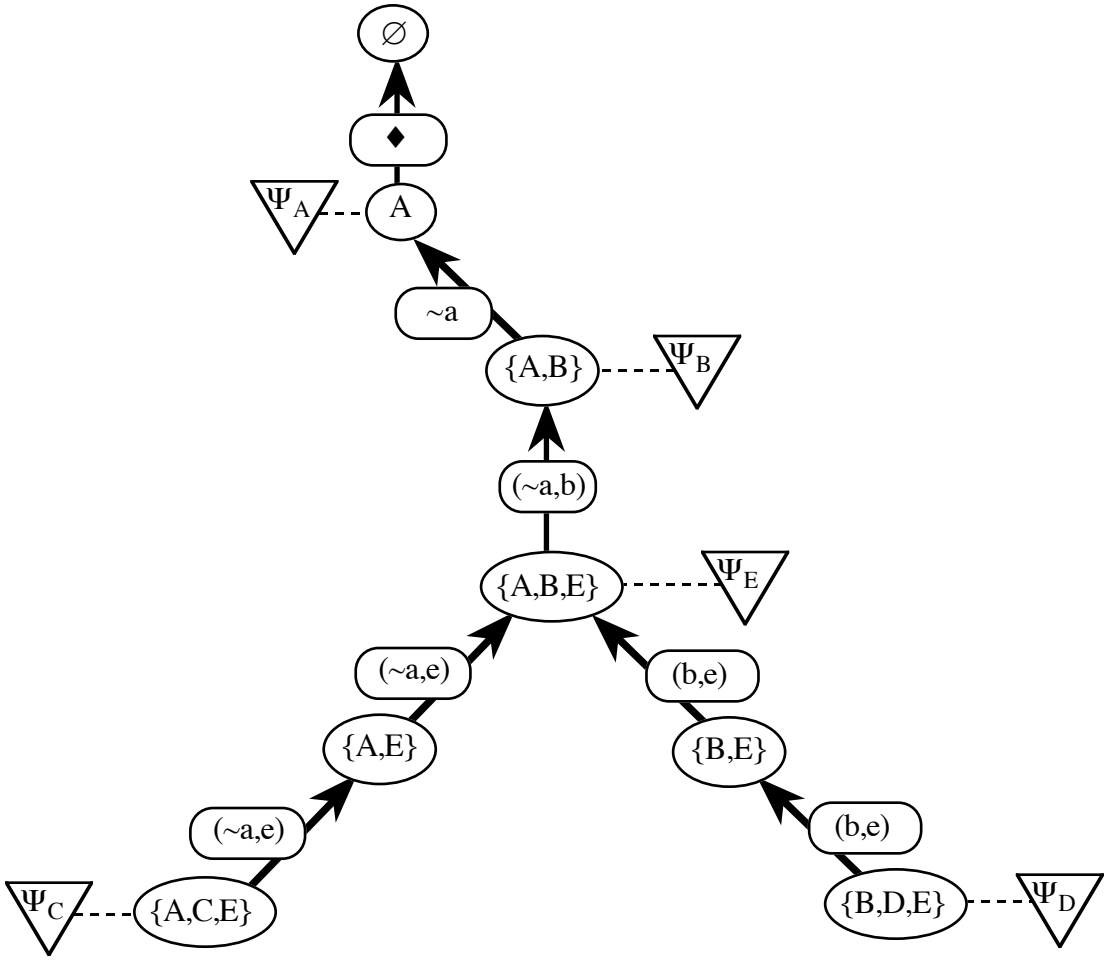
Theorem 7.2. Suppose h_X denotes the vertex that has the solution for X stored at its location. Then $\mathbf{z} \in \mathcal{W}_X$ given by

$$\mathbf{z}^{\downarrow\{X\}} = \Psi_X(\mathbf{c}^{Pa(h_X)\rightarrow h_X}), \text{ for every } X\in\mathcal{X} \quad (7.10)$$

is a solution for $F_1\oplus...\oplus F_k$.

Figure 7.7 illustrates the message passing scheme for the optimization problem. As per Theorem 7.2, a solution for F is given by ($\Psi_A(\mathbf{c}^{\varnothing\rightarrow\{A\}})$, $\Psi_B(\mathbf{c}^{\{A\}\rightarrow\{A,B\}})$, $\Psi_C(\mathbf{c}^{\{A,E\}\rightarrow\{A,C,E\}})$, $\Psi_D(\mathbf{c}^{\{B,E\}\rightarrow\{B,D,E\}})$, $\Psi_E(\mathbf{c}^{\{A,B\}\rightarrow\{A,B,E\}})$). From Figures 6 and 7, we see that configurations (~a, b, c, d, e) and (~a, b, ~c, d, e) are both solutions for F.

Figure 7.7. The propagation of configuration messages in the optimization problem. The configuration messages are shown as rectangles with rounded corners overlapping the corresponding edges. Note that the direction of messages is opposite to the direction of the edges. The solutions for the five variables are shown as inverted triangles attached to the vertices (where they are stored) by dotted lines.



7.4. Mitten's Axioms for Dynamic Programming

In optimization problems, the computational scheme described in section 4 is essentially the same as the method of non-serial dynamic programming (Nemhauser, 1966; Bertele and Brioschi, 1972). Bellman's dynamic programming methodology appealed to a principle of optimality that translates into axiom A3 with combination interpreted as addition and marginalization interpreted as maximization over the deleted variables (Bellman 1957). Mitten (1964) has described a more general framework for discrete dynamic programming. In this section, we describe Mitten's framework in terms of our notation.

Values and Valuations. The *value space* is $\mathcal{R}$, the set of real numbers. A *valuation for* h is a real-valued function on $\mathcal{W}_h$.

Combination. There is a mapping ©: $\mathcal{R}\times\mathcal{R} \rightarrow \mathcal{R}$ that is commutative and associative. Define a mapping $\oplus: \mathcal{V}\times\mathcal{V} \rightarrow \mathcal{V}$ such that whenever G and H are valuations for g and h respectively, G⊕H is a valuation for g∪h given by

$$(G\oplus H)(\mathbf{x}) = G(\mathbf{x}^{\downarrow g}) \copyright H(\mathbf{x}^{\downarrow h})$$

for all $\mathbf{x}\in\mathcal{W}_{g\cup h}$.

Monotonicity of Combination. We say that © is *monotonic* if $u\copyright v_1 \geq u\copyright v_2$ whenever $v_1 \geq v_2$. Suppose H_1 and H_2 are valuations for h. We say that $H_1 \geq H_2$ if $H_1(\mathbf{x}) \geq H_2(\mathbf{x})$ for all $\mathbf{x}\in\mathcal{W}_h$. Note that if © is monotonic, then $G\oplus H_1 \geq G\oplus H_2$ whenever $H_1 \geq H_2$.

Marginalization. Define a mapping $\downarrow h: \cup\{\mathcal{V}_g \mid g\supseteq h\} \rightarrow \mathcal{V}_h$ such that whenever G is a valuation for g, $G^{\downarrow h}$ is a valuation for h given by

$$G^{\downarrow h}(\mathbf{x}) = \mathrm{MAX}\{G(\mathbf{x}, \mathbf{y}) \mid \mathbf{y}\in\mathcal{W}_{g-h}\} \qquad (7.11)$$

for all $\mathbf{x}\in\mathcal{W}_h$.

> ***Theorem 7.3***. Suppose the value space is $\mathcal{R}$ and suppose marginalization is defined as in (7.11). If © is monotonic, and G and H are valuations for g and h, respectively, then $(G\oplus H)^{\downarrow g} = G\oplus(H^{\downarrow g\cap h})$.

Thus monotonicity of © implies axiom A3. The other condition that Mitten requires is called *separability* and it amounts to a serial factorization of the joint

objective function. In our framework, we do not require any particular structure for the factorization of the joint valuation.

7.5 Other Applications of the Axiomatic Framework

7.5.1. Most Probable Configuration

In many applications such as medical diagnosis, pattern recognition, circuit diagnosis, restoration of degraded images, etc, one is more interested in the values of some or all variables that have the highest joint probability (conditioned on all evidence) than in the marginal distributions of each of the variables (see for example, Pearl [1988], Geman and Geman [1984]). We shall refer to a configuration that has the maximum probability as a *most probable configuration*.

If one is working with a large number of variables, it is computationally infeasible to enumerate all configurations and compute the values of the joint distribution for each of these configurations. However, if the joint probability distribution factorizes on a hypertree with small hyperedges, then we can find a most probable configuration using the scheme described in section 7.3.

Valuations. In this section proper valuations will correspond to potentials as defined in chapter 3.

Combination. As in the case of probability propagation, when we refer to combination of potentials, we mean pointwise multiplication.

Suppose G and H are potentials on g and h, respectively, such that there exists an $\mathbf{x}\in\mathcal{W}_{g\cup h}$ such that $G(\mathbf{x}^{\downarrow g})H(\mathbf{x}^{\downarrow h}) > 0$. Then their *combination* RS is the potential on g given by

$$(RS)(\mathbf{x}) = R(\mathbf{x})S(\mathbf{x}). \tag{7.12}$$

If $G(\mathbf{x}^{\downarrow g})H(\mathbf{x}^{\downarrow h}) = 0$ for all $\mathbf{x}\in\mathcal{W}_{g\cup h}$, then we say that G and H are *not combinable*.

Marginalization. The marginalization operation for the computation of a most probable configuration differs from the marginalization operation defined for probability propagation.

Suppose G is a potential for g and suppose $h \subseteq g$, $h \neq g$. We will define the *marginal* of G to h, denoted by $G^{\downarrow h}$, to be a potential on h such that

$$G^{\downarrow h}(\mathbf{y}) = \text{MAX}\{G(\mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{g-h}\} \tag{7.13}$$

for all $\mathbf{y} \in \mathcal{W}_h$.

Proposition 7.1. The definitions of combination, and marginalization in (7.12) and (7.13) respectively, satisfy axioms A1', A2, and A3.

Thus we can compute a most probable configuration using the scheme described in section 7.3.

7.5.2. Most Plausible Configuration

Suppose that we have several independent pieces of evidence represented by belief functions. We would like to find a configuration that has the maximum plausibility function value where the plausibility function corresponds to the belief function that has been obtained by combining all independent pieces of evidence. We will refer to this problem as finding the most plausible configuration.

Our strategy is to reduce the problem to finding a most probable configuration. To do so, we need to identify the potentials corresponding to the various belief functions.

First let us state the problem more formally. Suppose $\mathcal{X}$ is a finite set of variables and suppose $\mathcal{H}$ is a hypertree on $\mathcal{X}$. Suppose $\text{Bel} = \oplus\{\text{Bel}_h \mid h \in \mathcal{H}\}$ where Bel_h is a belief function on $\mathcal{W}_h$. The belief functions Bel_h correspond to independent pieces of evidence and Bel is the joint belief function representing all evidence. Suppose Pl is the plausibility function corresponding to Bel and Pl_h is the plausibility function corresponding to Bel_h for each $h \in \mathcal{H}$. The problem is to find a $\mathbf{x}^* \in \mathcal{W}_\mathcal{X}$ such that

$$\text{Pl}(\{\mathbf{x}^*\}) = \text{MAX}\{\text{Pl}(\{\mathbf{x}\}) \mid \mathbf{x} \in \mathcal{W}_\mathcal{X}\}$$

Define potential R for $\mathcal{X}$ by $R(\mathbf{x}) = \text{Pl}(\{\mathbf{x}\})$ for each $\mathbf{x} \in \mathcal{W}_\mathcal{X}$. Also for each $h \in \mathcal{H}$, define potential R_h for h by $R_h(\mathbf{x}) = \text{Pl}_h(\{\mathbf{x}\})$ for each $\mathbf{x} \in \mathcal{W}_h$. The next proposition states that the potential R factorizes on $\mathcal{H}$ with factors R_h, $h \in \mathcal{H}$.

Proposition 7.2. Under the assumptions of the preceding paragraph,

$$R \propto \Pi\{R_h \mid h \in \mathcal{H}\}$$

In terms of the potential R, the problem of finding a most plausible configuration for Pl is equivalent to finding an optimal configuration for R. Proposition 7.2 tells us that we have a factorization of R on a hypertree. Thus we can use the method described in the previous section and our problem is solved.

7.6. Conclusions

In the introduction, we raised two questions: What is dynamic programming? And, when does dynamic programming work? The main contribution of this chapter is the abstract framework of valuation-based systems consisting of variables, frames of variables, values, valuations, and two operations—combination and marginalization. Assuming that combination and marginalization satisfy three simple axioms, we have described a method for computing a solution for the joint valuation using only local computation. We can think of the framework and its solution method as the answer to the first question. The three axioms constitute one answer to the second question.

7.7. Proofs

In this section, we provide proofs for the Theorems 1 and 2 stated in section 5 and Theorem 7.3 stated in section 6. We prove Theorems 1 and 2 only using axioms A1', A2 and A3. In other words, we do not assume that combination is addition and marginalization is minimization.

Lemma 7.1. Suppose h_1, ..., h_k are the vertices of a rooted Markov tree. Suppose for i = 1, ..., k, vertex h_i has the valuation F_i associated with it, where F_i is a valuation for h_i. Suppose h_k is a leaf in the rooted Markov tree with parent h_{k-1}. Suppose $\mathcal{X}$ denotes $h_1 \cup ... \cup h_k$ and $\mathcal{X}'$ denotes $h_1 \cup ... \cup h_{k-1}$. Then

$$(F_1 \otimes ... \otimes F_k)^{\downarrow \mathcal{X}'} = F_1 \otimes ... \otimes F_{k-2} \otimes (F_{k-1} \otimes F_k^{\downarrow (h_k \cap h_{k-1})}) \qquad (7.14)$$

Proof of Lemma 7.1. Note that axiom A1' allows us to write the LHS of (7.14) as is written above. The result in (7.14) follows directly from axiom A3 by substituting $\mathcal{X}'$ for g, h_k for h, $F_1 \otimes ... \otimes F_{k-1}$ for G, and F_k for H. Since h_k is a leaf in the rooted Markov tree with parent h_{k-1}, $h_k \cap h_{k-1} \subseteq h_{k-1}$. Thus $F_{k-1} \otimes F_k^{\downarrow (h_k \cap h_{k-1})}$ is a valuation for h_{k-1}. ■

Proof of Theorem 7.1. By axiom A2, $(F_1 \otimes ... \otimes F_k)^{\downarrow \varnothing}$ is obtained by sequentially marginalizing all variables in any sequence. A proof of this theorem is obtained by repeatedly applying the result of Lemma 7.1. At each step, a leaf of the rooted Markov tree sends a message to its parent, the parent combines this message with its own valuation, and the leaf is deleted from the tree. When the tree is reduced to only one vertex, the root, we have the result. ■

Next, we state a lemma that is needed to prove Theorem 7.2.

Lemma 7.2. Suppose h_1, ..., h_k are the vertices of a rooted Markov tree. Suppose for i = 1, ..., k, vertex h_i has the valuation F_i associated with it, where F_i is a valuation for h_i. Suppose h_k is a leaf in the rooted Markov tree with parent h_{k-1} and suppose $h_k - (h_k \cap h_{k-1}) = \{X_j\}$ If Ψ_{X_j} is a solution for X_j (with respect to F_k), and $\mathbf{c}$ is a solution for $F_1 \otimes ... \otimes F_{k-2} \otimes F_{k-1} \otimes F_k^{\downarrow (h_k \cap h_{k-1})}$, then $(\mathbf{c}, \Psi_{X_j}(\mathbf{c}^{\downarrow (h_k \cap h_{k-1})}))$ is a solution for $F_1 \otimes ... \otimes F_k$.

Proof of Lemma 7.2. We need to prove that $(F_1 \otimes ... \otimes F_k)(\mathbf{c}, \Psi_{X_j}(\mathbf{c}^{\downarrow (h_k \cap h_{k-1})})) = (F_1 \otimes ... \otimes F_k)^{\downarrow \varnothing}(\blacklozenge)$. We have $(F_1 \otimes ... \otimes F_k)(\mathbf{c}, \Psi_{X_j}(\mathbf{c}^{\downarrow (h_k \cap h_{k-1})}))$

$= (F_1 \otimes ... \otimes F_{k-1})(\mathbf{c}) \copyright F_k(\mathbf{c}^{\downarrow (h_k \cap h_{k-1})}, \Psi_{X_j}(\mathbf{c}^{\downarrow (h_k \cap h_{k-1})}))$ (by definition of combination)

$= (F_1 \otimes ... \otimes F_{k-1})(\mathbf{c}) \copyright F_k^{\downarrow (h_k \cap h_{k-1})}(\mathbf{c}^{\downarrow (h_k \cap h_{k-1})})$

(since Ψ_{X_j} is a solution for X_j with respect to F_k)

$= (F_1 \otimes ... \otimes F_{k-2} \otimes F_{k-1} \otimes F_k^{\downarrow (h_k \cap h_{k-1})})(\mathbf{c})$ (by definition of combination)

$= (F_1 \otimes ... \otimes F_{k-2} \otimes F_{k-1} \otimes F_k^{\downarrow (h_k \cap h_{k-1})})^{\downarrow \varnothing}(\blacklozenge)$

(since $\mathbf{c}$ is a solution for $(F_1 \otimes ... \otimes F_{k-2} \otimes F_{k-1} \otimes F_k^{\downarrow (h_k \cap h_{k-1})})$)

$= ((F_1 \otimes ... \otimes F_k)^{\downarrow (h_1 \cup ... \cup h_k) - \{X_j\}})^{\downarrow \varnothing}(\blacklozenge)$ (using Lemma 7.1)

$= (F_1 \otimes ... \otimes F_k)^{\downarrow \varnothing}(\blacklozenge)$ (using axiom A2)

■

Proof of Theorem 7.2. A proof of this theorem is obtained by repeated application of Lemma 7.2. First we apply Lemma 7.2 for the entire rooted Markov tree. In our rooted Markov tree construction algorithm, if h_{k-1} is a parent of h_k, then either $h_k - (h_k \cap h_{k-1}) = \{X_j\}$ for some $j \in \mathcal{X}$, or $h_k \subseteq h_{k-1}$. The first case corresponds to the statement of Lemma 7.2. In the second case, when h_k sends a valuation message to h_{k-1}, there is no marginalization. Hence, there is no solution function stored at h_k. But in this case, $F_1 \otimes ... \otimes F_{k-2} \otimes F_{k-1} \otimes F_k^{\downarrow (h_k \cap h_{k-1})} = F_1 \oplus ... \oplus F_k$. Next, we apply Lemma 7.2 to the rooted Markov tree with vertex h_k and edge (h_k, h_{k-1}) deleted. And so on, until the only vertex left is $\varnothing$. But $\blacklozenge$ is the solution for $(F_1 \otimes ... \otimes F_k)^{\downarrow \varnothing}$. Thus the configuration messages as defined in (7.7), (7.8) and (7.9) give us the solution for $F_1 \otimes ... \otimes F_k$ as stated in Theorem 7.2. ■

Proof of Theorem 7.3. Suppose $\mathbf{x} \in \mathcal{W}_{g-h}$ and $\mathbf{y} \in \mathcal{W}_{g \cap h}$. Then

$$\begin{aligned}(G \oplus H)^{\downarrow g}(\mathbf{x}, \mathbf{y}) &= \text{MAX}\{(G \oplus H)(\mathbf{x}, \mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{h-g}\} \\ &= \text{MAX}\{G(\mathbf{x}, \mathbf{y}) \copyright H(\mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{h-g}\} \\ &\geq G(\mathbf{x}, \mathbf{y}) \copyright (\text{MAX}\{H(\mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{h-g}\})\end{aligned}$$

In other words, $(G \oplus H)^{\downarrow g} \geq G \oplus (H^{\downarrow g \cap h})$. But since © is monotonic and $\text{MAX}\{H(\mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{h-g}\} \geq H(\mathbf{y}, \mathbf{z})$ for all $\mathbf{z} \in \mathcal{W}_{h-g}$,we have

$$G(\mathbf{x}, \mathbf{y}) \copyright (\text{MAX}\{H(\mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{h-g}\}) \geq G(\mathbf{x}, \mathbf{y}) \copyright H(\mathbf{y}, \mathbf{z})$$

for all $\mathbf{z} \in \mathcal{W}_{h-g}$. In particular, this inequality must hold for the maximum of the RHS with respect to $\mathbf{z}$, i.e., $G(\mathbf{x}, \mathbf{y}) \copyright (\text{MAX}\{H(\mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{h-g}\})$

$\geq \text{MAX}\{G(\mathbf{x}, \mathbf{y}) \copyright H(\mathbf{y}, \mathbf{z}) \mid \mathbf{z} \in \mathcal{W}_{h-g}\}$, i.e., $G \oplus (H^{\downarrow g \cap h}) \geq (G \oplus H)^{\downarrow g}$. Since we have already shown that $(G \oplus H)^{\downarrow g} \geq G \oplus (H^{\downarrow g \cap h})$, we have the result. ■

Proof of Proposition 7.1. ■

Proof of Proposition 7.2. Suppose Q is the commonality function corresponding to Bel and Q_h is the commonality function corresponding to Bel_h for each $h \in \mathcal{H}$. Then

$$\begin{aligned} R(\mathbf{x}) = \text{Pl}(\{\mathbf{x}\}) = Q(\{\mathbf{x}\}) &\propto \Pi\{Q_h^{\uparrow \aleph}(\{\mathbf{x}\}) \mid h \in \mathcal{H}\} \\ &= \Pi\{Q_h(\{\mathbf{x}^{\downarrow h}\}) \mid h \in \mathcal{H}\} \\ &= \Pi\{\text{Pl}_h(\{\mathbf{x}^{\downarrow h}\}) \mid h \in \mathcal{H}\} \\ &= \Pi\{R_h(\mathbf{x}^{\downarrow h}) \mid h \in \mathcal{H}\}. \end{aligned}$$

■

CHAPTER EIGHT

Constraint Satisfaction Problems

In this chapter we consider constraint satisfaction problems. We show how this problem fits in the axiomatic framework described in chapter seven. We conclude this section by solving a small constraint satisfaction problem in detail.

A constraint satisfaction problem consists of finding a configuration of all variables that satisfies all constraints. Since the number of configurations is an exponential function of the number of variables, it is not possible to enumerate all configurations and check each one to see if it satisfies all constraints. However, if each constraint only involves a small subset of variables and these subsets form a hypertree, then it is possible to find a feasible configuration by local propagation using the method described in Chapter 7. Such solution procedures to constraint satisfaction problems have been proposed by Freuder [1982, 1985] and are known in the artificial intelligence literature as *backtrack-free* methods. Other solution procedures to constraint satisfaction problems involving backtracking have been proposed by Montanari [1974], Mackworth [1977], and Dechter and Pearl [1987].

8.1. Constraint Satisfaction Problems

A *constraint* for h, denoted by C_h, is a non-empty subset of the frame W_h for h. Intuitively, C_h represents the set of configurations of h that are feasible. Suppose

we are given a constraint $\mathcal{C}_h$ for each hyperedge h of a hypergraph $\mathcal{H}$. The *constraint satisfaction problem* (CSP) can be stated as follows:

(CSP P_1): Find a $\mathbf{x}\in\mathcal{W}_\chi$ such that $\mathbf{x}^{\downarrow h}\in\mathcal{C}_h$ for each $h\in\mathcal{H}$.

We will refer to a configuration satisfying the above condition *a feasible configuration* for the CSP P_1.

Valuations. Given a constraint $\mathcal{C}_h$ for h, we will construct a valuation $C_h:\mathcal{W}_h \rightarrow \{f, i\}$ as follows (f means feasible, and i means infeasible):

$$C_h(\mathbf{x}) = \begin{cases} f & \text{if } \mathbf{x}\in\mathcal{C}_h \\ \\ i & \text{if } \mathbf{x}\notin\mathcal{C}_h \end{cases} \tag{8.1}$$

for all $\mathbf{x}\in\mathcal{W}_h$.

Suppose C_h is a valuation on h. We shall say that C_h is a *proper* valuation if there exists an $\mathbf{x}\in\mathcal{W}_h$ such that $C_h(\mathbf{x}) = f$. Thus, a proper valuation cannot be identically equal to i for all configurations. Note that since each constraint $\mathcal{C}_h$ is a non-empty subset of $\mathcal{W}_h$, each of the valuations constructed using (8.1) are proper valuations.

Combination. Suppose that C_g and C_h are valuations on g and h respectively. We will define $C_g\otimes C_h$ to be the valuation on $g\cup h$ given by

$$(C_g\otimes C_h)(\mathbf{x}) = \begin{cases} f & \text{if } C_g(\mathbf{x}^{\downarrow g}) = f \text{ and } C_h(\mathbf{x}^{\downarrow h}) = f \\ \\ i & \text{otherwise} \end{cases} \tag{8.2}$$

for all $\mathbf{x}\in\mathcal{W}_{g\cup h}$.

It is clear from the definition of combination above that it satisfies axiom A1' (commutativity and associativity of combination).

Marginalization. Suppose C_g is a proper valuation on g and suppose $h\subseteq g$. Then the marginal of C_g for h, $C_g^{\downarrow h}$, is defined as follows:

$$C_g^{\downarrow h}(\mathbf{x}) = \begin{cases} f & \text{if there is a } \mathbf{y}\in\mathcal{W}_g \text{ such that } C_g(\mathbf{y})=f \text{ and } \mathbf{y}^{\downarrow h}=\mathbf{x} \\ \\ i & \text{otherwise} \end{cases} \tag{8.3}$$

for all $\mathbf{x}\in\mathcal{W}_h$.

It is clear from the definition of marginalization that it satisfies axiom A2a: If C_g is a proper valuation on g and $h_1 \subseteq h_2 \subseteq g$, then $(C_g^{\downarrow h_2})^{\downarrow h_1} = C_g^{\downarrow h_1}$.

Note that in terms of the definitions of the framework, we can restate CSP P_1 as follows: Find a solution configuration $\mathbf{x} \in \mathcal{W}_\chi$ for the joint constraint $\otimes\{C_h \mid h \in \mathcal{H}\}$.

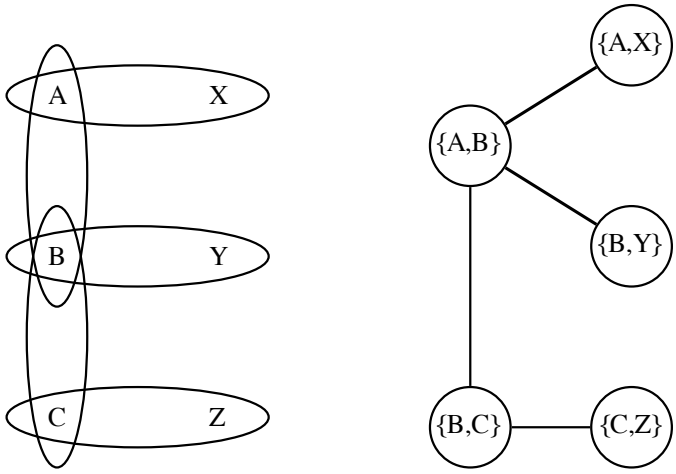
Proposition 8.1. The definitions of combination in (8.2), and marginalization in (8.3) satisfy axioms A1', A2, and A3.

Thus all axioms are satisfied making local computation of a feasible configuration of CSP P_1 possible.

8.2. An Example

This example is adapted from de Kleer [1986]. There are six variables A, B, C, X, Y, and Z. The frame for each variable has 2 configurations: 0 and 1. Thus we have $2^6 = 64$ configurations of all variables. There are 5 constraints specified as follows: A = X, B = Y, C = Z, A ≠ B, and B ≠ C. The hypergraph corresponding to these five constraints is $\mathcal{H}$ = {{A, X}, {B, Y}, {C, Z}, {A, B}, {B, C}}. Note that $\mathcal{H}$ is a hypertree (see Figure 8.1). A hypertree construction sequence for $\mathcal{H}$ is {A, B},{A, X},{B, Y},{B, C},{C, Z}. A branching β for this construction sequence is as follows: β({A, X}) = {A, B}, β({B, Y}) = {A, B}, β({B, C}) = {A, B}, β({C, Z}) = {B, C}. A Markov tree representative for $\mathcal{H}$ corresponding to branching β is also shown in Figure 8.1.

Figure 8.1. The hypertree and a Markov tree representative for it.



The five valuations corresponding to the five constraints are shown in Table 8.1.

Table 8.1. The valuations corresponding to the constraints.

a	x	$C_{\{A,X\}}$
0	0	f
0	1	i
1	0	i
1	1	f

b	y	$C_{\{B,Y\}}$
0	0	f
0	1	i
1	0	i
1	1	f

c	z	$C_{\{C,Z\}}$
0	0	f
0	1	i
1	0	i
1	1	f

a	b	$C_{\{A,B\}}$
0	0	i
0	1	f
1	0	f
1	1	i

b	c	$C_{\{B,C\}}$
0	0	i
0	1	f
1	0	f
1	1	i

Suppose we fix vertex {A, B} as the root. The scheme for computation of a feasible configuration as described in chapter 7 is as follows. First the vertices {A, X}, {B, Y} and {C, Z} marginalize the valuations at those vertices to the vertices contained in the intersection with their neighbors and send these valuations to their

neighbors. Table 8.2 shows the messages $(C_{\{A,X\}})^{\downarrow\{A\}}$, $(C_{\{B,Y\}})^{\downarrow\{B\}}$, and $(C_{\{C,Z\}})^{\downarrow\{C\}}$ and corresponding solution extensions $\psi_{\{A,X\}}$, $\psi_{\{B,Y\}}$, and $\psi_{\{C,Z\}}$.

Table 8.2. The marginals $(C_{\{A,X\}})^{\downarrow\{A\}}$, $(C_{\{B,Y\}})^{\downarrow\{B\}}$, and $(C_{\{C,Z\}})^{\downarrow\{C\}}$ and corresponding solution extensions.

a	$(C_{\{A,X\}})^{\downarrow\{A\}}$	$\psi_{\{A,X\}}(a) =$ a x
0	f	0 0
1	f	1 1

b	$(C_{\{B,Y\}})^{\downarrow\{B\}}$	$\psi_{\{B,Y\}}(b) =$ b y
0	f	0 0
1	f	1 1

c	$(C_{\{C,Z\}})^{\downarrow\{C\}}$	$\psi_{\{C,Z\}}(c) =$ c z
0	f	0 0
1	f	1 1

Next, vertex {B, C} combines the incoming message $(C_{\{C,Z\}})^{\downarrow\{C\}}$ with its valuation, marginalizes the combination to {B}, and transmits the resulting valuation to {A, B}. Table 8.3 shows the combination $(C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}}$, the marginal $((C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}})^{\downarrow\{B\}}$, and a solution extension $\psi_{\{B,C\}}$.

Table 8.3. The combination $(C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}}$, the marginal $((C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}})^{\downarrow\{B\}}$, and a solution extension $\psi_{\{B,C\}}$.

b	c	$(C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}}$
0	0	i
0	1	f
1	0	f
1	1	i

b	$((C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}})^{\downarrow\{B\}}$	$\psi_{\{B,C\}}(b) =$ b c
0	f	0 1
1	f	1 0

Next, root vertex {A,B} combines its valuation with all three incoming messages $(C_{\{A,X\}})^{\downarrow\{A\}}$, $(C_{\{B,Y\}})^{\downarrow\{B\}}$, and $((C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}})^{\downarrow\{B\}}$, computes a

feasible configuration of its variables which in this case can be observed from Table 8.4 to be (a, b) = (0,1), and transmits to its neighbors the projection of this configuration to its intersection with its neighbors. Note that since the combination $(C_{\{A, X\}})^{\downarrow\{A\}} \otimes (C_{\{B, Y\}})^{\downarrow\{B\}} \otimes ((C_{\{C, Z\}})^{\downarrow\{C\}} \otimes C_{\{B, C\}})^{\downarrow\{B\}} \otimes C_{\{A, B\}}$ is a proper valuation, the constraint satisfaction problem is feasible.

Table 8.4. The combination $(C_{\{A, X\}})^{\downarrow\{A\}} \otimes (C_{\{B, Y\}})^{\downarrow\{B\}} \otimes ((C_{\{C, Z\}})^{\downarrow\{C\}} \otimes C_{\{B, C\}})^{\downarrow\{B\}} \otimes C_{\{A, B\}}$.

a	b	$(C_{\{A,X\}})^{\downarrow\{A\}} \otimes (C_{\{B,Y\}})^{\downarrow\{B\}} \otimes ((C_{\{C,Z\}})^{\downarrow\{C\}} \otimes C_{\{B,C\}})^{\downarrow\{B\}} \otimes C_{\{A,B\}}$
0	0	i
0	1	f
1	0	f
1	1	i

Next each of the vertices {A, X}, {B, Y} and {B, C} computes a feasible configuration of its variables using the solution extension function and the incoming configuration message. From Tables 8.2 and 8.3, we can see that the feasible configurations of the variables at these three vertices are (a, x) = (0,0), (b, y) = (1,1) and (b, c) = (1,0). Vertex {B, C} transmits to its neighbor {C, Z} the projection of its feasible configuration to {C}.

Finally, vertex {C, Z} computes a feasible configuration of its variable using the incoming configuration (c) = (0) and the solution extension $\psi_{\{C,Z\}}$. From Table 8.2, we can see that a feasible configuration is (c, z) = (0,0). Thus (a, b, c, x, y, z) = (0,1,0,0,1,0) is a feasible configuration for the constraint satisfaction problem.

References

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.

Barnett, J. A. (1981), Computational methods for a mathematical theory of evidence, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC, 868–875.

Beeri, C., Fagin, R., Maier, D. and Yannakakis, M. (1983), "On the desirability of acyclic database schemes," *Journal of the Association for Computing Machinery*, **30**(3), 479-513.

Bellman, R. E. (1957), *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Berge, C. (1973), *Graphs and Hypergraphs*, translated from French by E. Minieka, North-Holland, Amsterdam.

Bertele, U. and Brioschi, F. (1972), *Nonserial Dynamic Programming*, Academic Press, Orlando, FL.

Besag, J. (1972), Nearest neighbor systems and the auto-logistic model for binary data, *Journal of the Royal Statistical Society*, series B, **34**, 75–83.

Besag, J. (1974), Spatial interaction and the statistical analysis of lattice systems (with discussion), *Journal of the Royal Statistical Society*, series B, **36**, 192–326.

Blalock, H. M. (1971), *Causal Models in the Social Sciences*, Macmillan, New York, NY.

Brownston, L. S., Farrell, R. G., Kant, E. and Martin, N. (1985), *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*, Addison-Wesley, Reading, MA.

Buchanan, B. G. and Shortliffe, E. H., eds. (1984), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, MA.

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

Chin, H. L. and Cooper, G. F. (1987), Stochastic simulation of Bayesian belief networks, in *Proceedings of the Third AAAI Workshop on Uncertainty in Artificial Intelligence*, Seattle, WA, 106–113.

Cohen, P., Shafer, G. and Shenoy, P. P. (1987), Modifiable combining functions, *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, **1**, 47–57.

Darroch, J. N., Lauritzen, S. L. and Speed, T. P. (1980), Markov fields and log-linear models for contingency tables, *Annals of Statistics*, **8**, 522–539.

Davis, R. and King, J. J. (1984), The origin of rule-based systems in AI, in Buchanan and Shortliffe [1984], 20–52.

Dawid, A. P. (1979), Conditional independence in statistical theory (with discussion), *Journal of the Royal Statistical Society, Series B*, **41**(1), 1-31.

Dawid, A. P. and Lauritzen, S. L. (1989), Markov distributions, hyper Markov laws and meta Markov models on decomposable graphs with applications to Bayesian learning in expert systems, Research Report No. R-89-31, Institute for Electronic Systems, University of Aalborg, Aalborg, Denmark.

Dechter, R. and Pearl, J. (1988), "Tree-clustering schemes for constraint processing," *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88),* St. Paul, MN, **1**, 150-154.

Dechter, R., Dechter A., and Pearl J. (1990), "Optimization in constraint networks," in Oliver, R. M. and Smith, J. Q. (eds.), *Influence Diagrams, Belief Nets, and Decision Analysis*, 411-425, John Wiley & Sons, New York, NY.

Dechter, R. and Pearl, J. (1987), Network-based heuristics for constraint-satisfaction problems, *Artificial Intelligence*, **34**, 1–38.

Dempster, A. P. (1966), "New methods for reasoning toward posterior distributions based on sample data," *Annals of Mathematical Statistics*, **37**, 355-374.

Dempster, A. P. (1967), Upper and lower probabilities induced by a multivalued mapping, *Annals of Mathematical Statistics*, **38**, 325-339.

Dempster, A. P. (1968), A generalization of Bayesian inference (with discussion), *Journal of the Royal Statistical Society, Series B*, **30**, 205-247.

Dempster, A. P. (1990), Construction and local computation aspects of network belief functions, in Oliver, R. M. and Smith, J. Q. (eds.), *Influence Diagrams, Belief Nets, and Decision Analysis*, 121-142, John Wiley & Sons, New York, NY.

Dempster, A. P. and A. Kong (1988), Uncertain evidence and artificial analysis, *Journal of Statistical Planning and Inference*, **20**, 355-368. Reprinted on pp. 522-528 of Shafer and Pearl (1990).

Dobruschin, P. L. (1968), The description of a random field by means of conditional probabilities and conditions of its regularity, *Theory of Probability and Its Applications*, **13**(2), 197-224.

de Kleer, J. (1986), An assumption-based TMS, *Artificial Intelligence*, **28**, 127–162.

Dubois, D. and H. Prade (1990), Inference in possibilistic hypergraphs, *Proceedings of the Third International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU-90)*, Paris, France, 228-230.

Duda, R., P. Hart and N. Nilsson (1981), Subjective Bayesian methods for rule-based inference systems, in Webber, B. L. and Nilsson, N. J. (eds.), *Readings in Artificial Intelligence*, 192-200, Tioga, Palo Alto, CA.

Edwards, D. and Kreiner, S. (1983), The analysis of contingency tables by graphical models, *Biometrika*, **70**, 553–565.

Freuder, E. C. (1982), A sufficient condition for backtrack-free search, *Journal of the Association of Computing Machinery*, **21**(1), 24–32.

Freuder, E. C. (1985), A sufficient condition for backtrack-bounded search, *Journal of the Association of Computing Machinery*, **32**(4), 755–761.

Frydenberg, M. (1989), The chain graph Markov property, Research Report No. 186, Department of Theoretical Statistics, University of Aarhus, Denmark.

Frydenberg, M. and S. L. Lauritzen (1989), Decomposition of maximum likelihood in mixed graphical interaction models, *Biometrika*, **76**(3), 539-555.

Geiger, D. (1990), Graphoids: A qualitative framework for probabilistic inference, Ph.D. dissertation, Department of Computer Science, University of California at Los Angeles, CA.

Geman, S. and Geman, D. (1984), Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, 721–741.

Goldman, S. A. and Rivest, R. L. (1988), A non-iterative maximum entropy algorithm, in Lemmer, J. F. and Kanal, L. N. (eds.), *Uncertainty in Artificial Intelligence*, *2*, 133–148, North-Holland, Amsterdam.

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

Henrion, M. (1988), Propagating uncertainty in Bayesian networks by probabilistic logic sampling, in Lemmer, J. F. and Kanal, L. N. (eds.), *Uncertainty in Artificial Intelligence, 2*, 149–164, North-Holland, Amsterdam.

Hsia, Y-T. and Shenoy, P. P. (1989), MacEvidence: A visual environment for constructing and evaluating evidential systems, *Proceedings of the World Conference on Information Processing and Communication (WOCON-INFOR 89),* Seoul, South Korea, 20-25.

Hsia, Y-T. and Shenoy, P. P. (1989), An evidential language for expert systems, in Ras, Z. W., ed., *Methodologies for Intelligent Systems, 4*, 9–16, North-Holland, Amsterdam.

Kelly, C. W. III and Barclay, S. (1973), A general Bayesian model for hierarchical inference, *Organizational Behavior and Human Performance*, **10**, 388–403.

Kiiveri, H., Speed, T. P. and Carlin, J. B. (1984), Recursive causal models, *Journal of the Australian Mathematics Society, Series A*, **36**, 30-52.

Kirkpatrick, S., Gelatt, C. D. Jr. and Vecchi, M. P. (1983), Optimization by simulated annealing, *Science*, **220**, 671–680.

Kong, A. (1986), Multivariate belief functions and graphical models, doctoral dissertation, Department of Statistics, Harvard University.

Lauritzen, S. L. (1989), *Lectures on Contingency Tables*, third edition. Research Report No. R-89-24, Institute for Electronic Systems, University of Aalborg, Aalborg, Denmark.

Lauritzen, S. L. (1989), Mixed graphical association models, *Scandinavian Journal of Statistics*, **16**(4), 273-306.

Lauritzen, S. L. (1990), Propagation of probabilities, means and variances in mixed graphical association models, Research Report No. R-90-18, Institute for Electronic Systems, University of Aalborg, Aalborg, Denmark.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., and Leimer, H. G. (1988), Independence properties of directed Markov fields, Research Report No. R-88-32, Institute for Electronic Systems, University of Aalborg, Aalborg, Denmark. To appear in *Networks*.

Lauritzen, S. L. and Frydenberg, M. (1988), Decomposition of maximum likelihood in mixed graphical interaction models, Research Report No. R-88-17, Institute for Electronic Systems, University of Aalborg, Aalborg, Denmark. To appear in *Biometrika*.

Lauritzen, S. L., Speed, T. P. and Vijayan, K. (1984), Decomposable graphs and hypergraphs, *Journal of the Australian Mathematical Society*, series A, **36**, 12–29.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988), Local computations with probabilities on graphical structures and their application to expert systems (with discussion), *Journal of the Royal Statistical Society*, series B, **50**, to appear.

Lauritzen, S. L. and Wermuth, N. (1989), Graphical models for associations between variables, some of which are qualitative and some quantitative, *The Annals of Statistics*, **17**(1), 31-57.

Leimer, H. G. (1989), Triangulated graphs with marked vertices, *Annals of Discrete Mathematics*, **41**, 311-324.

Lemmer, J. F. and Kanal, L. N., eds. (1988), *Uncertainty in Artificial Intelligence 2*, North-Holland, Amsterdam.

Mackworth, A. K. (1977), Consistency in networks of relations, *Artificial Intelligence*, **8**(1), 99–118.

Maier, D. (1983), *The Theory of Relational Databases*, Computer Science Press.

Mellouli, K. (1987), On the propagation of beliefs in networks using the Dempster-Shafer theory of evidence, doctoral dissertation, School of Business, University of Kansas.

Mellouli, K., Shafer, G. and Shenoy, P. P. (1987), Qualitative Markov Networks, in Bouchon, B. and Yager, R. R. (eds.), *Uncertainty in Knowledge-Based Systems*, Lecture Notes in Computer Science Series, **286**, 69–74, Springer-Verlag, Berlin, Germany.

Montanari, U. (1974), Networks of constraints: Fundamental properties and applications to picture processing, *Information Science*, **7**, 95–132.

Moussouris, J. (1974), Gibbs and Markov random systems with constraints, *Journal of Statistical Physics*, **10**(1), 11–33.

Ndilikilikesha, P. (in progress), A study of influence diagrams and their generalizations, doctoral dissertation, School of Business, University of Kansas.

Pearl, J. (1986), Fusion, propagation and structuring in belief networks, *Artificial Intelligence*, **29**, 241–288.

Pearl, J. (1987), "Distributed revision of composite beliefs," *Artificial Intelligence*, **33**(2), 173-215.

Pearl, J. (1987b), Evidential reasoning using stochastic simulation of causal models, *Artificial Intelligence*, **32**, 245–257.

Pearl, J. (1987c), "Addendum: Evidential Reasoning using stochastic simulation of causal models," *Artificial Intelligence*, **33**, 131.

Pearl, J. (1988), *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA.

Pearl, J. and Paz, A. (1987), Graphoids: A graph based logic for reasoning about relevance relations, in Boulay, B. D. et al (eds.), *Advances in Artificial Intelligence*, **2**, 357-363, North-Holland, Amsterdam.

Rose, D. J. (1970), Symmetric elimination on sparse positive definite systems and the potential flow network problem, Ph.D. dissertation, Harvard University.

Rose, D. J. (1970b), Triangulated graphs and the elimination process, *Journal of Mathematical Analysis and Applications*, **32**, 597–609.

Rose, D. J. (1973), A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, in Read, R. C. (ed.), *Graph Theory and Computing*, 183-217, Academic Press, Orlando, FL.

Rose, D. J., Tarjan, R. E. and Leuker, G. S. (1976), Algorithmic aspects of vertex elimination on graphs, *SIAM Journal of Computing*, **5**(2), 266-283.

Shachter, R. D. and Heckerman, D. (1987), A backwards view for assessment, *AI Magazine*, **8**(3), 55-61.

Shafer, G. (1976), *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, NJ.

Shafer, G. (1982), Belief functions and parametric models (with discussion), *Journal of the Royal Statistical Society,* series B, **44**, 322–352.

Shafer, G. (1985), Conditional Probability, *International Statistical Review*, **53**, 261–277.

Shafer, G. (1987), Belief functions and possibility measures, in Bezdek, J., ed., *The Analysis of Fuzzy Information*, **1**, 51–84, CRC Press, Boca Raton, FL.

Shafer, G. (1990), Perspectives on the theory and practice of belief functions, *International Journal of Approximate Reasoning*, **4**, 323-362.

Shafer, G. and Logan, R. (1987), Implementing Dempster's rule for hierarchical evidence, *Artificial Intelligence*, **33**, 271–298.

Shafer, G. and Pearl, J., eds. (1990), *Readings in Uncertain Reasoning*, Morgan Kauffman, San Mateo, CA.

Shafer, G. and Shenoy, P. P. (1988), Bayesian and belief-function propagation, School of Business Working Paper No. 192, University of Kansas, Lawrence, Kansas.

Shafer, G. and Shenoy, P. P. (1988), Local computation in hypertrees, School of Business Working Paper No. 201, University of Kansas.

Shafer, G. and Shenoy, P. P. (1990), Probability propagation, *Annals of Mathematics and Artificial Intelligence*, **2**(1–4), 327–352.

Shafer, G., Shenoy, P. P. and Mellouli, K. (1987), Propagating belief functions in qualitative Markov trees, *International Journal of Approximate Reasoning*, **1**(4), 349–400.

Shafer, G., Shenoy, P. P. and Srivastava, R. P. (1988), AUDITOR'S ASSISTANT: A knowledge engineering tool for audit decisions*, Auditing Symposium IX: Proceedings of the 1988 Touche Ross/University of Kansas Symposium on Auditing Problems*, Lawrence, KS, 61–84.

Shafer, G. and Tversky, A. (1985), Languages and designs for probability judgment, *Cognitive Science*, **9**, 309–339.

Shenoy, P. P. (1989a), A valuation-based language for expert systems, *International Journal of Approximate Reasoning*, **3**(5), 383–411.

Shenoy, P. P. (1989b), On Spohn's rule for revisions of beliefs, School of Business Working Paper No. 213, University of Kansas. To appear in *International Journal of Approximate Reasoning* in 1991.

Shenoy, P. P. (1990a), Consistency in valuation-based systems, School of Business Working Paper No. 216, University of Kansas.

Shenoy, P. P. (1990b), Valuation-based systems for Bayesian decision analysis, School of Business Working Paper No. 220, University of Kansas.

Shenoy, P. P. (1990c), On Spohn's theory of epistemic beliefs, *Proceedings of the Third International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU-90)*, Paris, France, 455–457.

Shenoy, P. P. (1990d), Valuation-based systems: A framework for representing and reasoning with knowledge, *Proceedings of the FAW Workshop on Uncertainty in Knowledge-based Systems*, Ulm, Germany, 329–336.

Shenoy, P. P. (1990e), Valuation-based systems for discrete optimization, *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, Boston, MA, 334–343.

Shenoy, P. P. (1990f), Valuation-based systems for propositional logic, in Ras, Z. W., Zemankova, M., and Emrich, M. L., eds., *Methodologies for Intelligent Systems, 5*, 305–312, North-Holland, Amsterdam.

Shenoy, P. P. (1990g), A new method for representing and solving Bayesian decision problems, School of Business Working Paper No. 223, University of Kansas.

Shenoy, P. P. and Shafer, G. (1986), Propagating belief functions using local computations, *IEEE Expert*, **1**(3), 43–52.

Shenoy, P. P. and Shafer, G. (1988), An axiomatic framework for Bayesian and belief-function propagation, in *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, Minneapolis, MN, 307–314.

Shenoy, P. P. and Shafer, G. (1988), Constraint Propagation, School of Business Working Paper No. 208, University of Kansas.

Shenoy, P. P. and Shafer, G. (1990), Axioms for probability and belief-function propagation, in Shachter, R. D., Levitt, T. S., Lemmer, J. F., and Kanal, L. N., eds., *Uncertainty in Artificial Intelligence, 4*, 169–198, North-Holland, Amsterdam. Reprinted in Shafer and Pearl [1990], pp. 575–610.

Shenoy, P. P., Shafer, G. and Mellouli, K. (1986), Propagation of belief functions: A distributed approach, in *Proceedings of the Second Workshop on Uncertainty in Artificial Intelligence*, Philadelphia, PA, 249–260. Also in Lemmer and Kanal [1988], 325–336.

Speed, T. P. (1979), A note on nearest-neighbor Gibbs and Markov probabilities, *Sankhya: The Indian Journal of Statistics*, **41**, series A, 184–197.

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 Journal of Computing*, **13**, 566–579.

Wermuth, N. and Lauritzen, S. L. (1983), Graphical and recursive models for contingency tables, *Biometrika*, **70**, 537–552.

Wold, H. D. A. (1954), Causality and econometrics, *Econometrica*, **28**, 443–463.

Wright, S. (1934), The method of path coefficients, *Annals of Mathematical Statistics*, **5**, 161–215.

Yannakakis, M. (1981), Computing the minimum fill-in is NP-complete, *SIAM Journal of Algebraic Discrete Methods*, **2**, 77–79.

Zarley, D. K. (1988), An evidential reasoning system, School of Business Working Paper No. 206, University of Kansas.

Zarley, D. K., Hsia, Y. and Shafer, G. (1988), Evidential reasoning using DELIEF, in *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI–88)*, **1**, 205–209.

Zhang, L. (1988), Studies on finding hypertree covers for hypergraphs, School of Business Working Paper No. 198, University of Kansas.