KU SCHOOL OF BUSINESS WORKING PAPER No. 270

BINARY JOIN TREES FOR COMPUTING MARGINALS IN THE SHENOY-SHAFER ARCHITECTURE

Prakash P. Shenoy

December 1995
Revised March 1997[†]

School of Business
University of Kansas
Summerfield Hall
Lawrence, KS 66045-2003, USA
pshenoy@ku.edu
http://lark.cc.ku.edu/~pshenoy

[†] Appeared in *International Journal of Approximate Reasoning*, **17**(2-3), 1997, 239–263.

TABLE OF CONTENTS

ABSTRACT……1
1 INTRODUCTION……1
2 THE VALUATION NETWORK FRAMEWORK ……2
3 AXIOMS FOR LOCAL COMPUTATION……5
4 COMPUTING A MARGINAL USING THE FUSION ALGORITHM ……6
5 FUSION ALGORITHM AS MESSAGE PASSING IN JOIN TREES ……9
6 COMPUTING MULTIPLE MARGINALS……13
7 BINARY JOIN TREES……15
8 CONCLUSIONS ……20
ACKNOWLEDGMENTS……21
REFERENCES ……21
SELECTED WORKING PAPERS ……24

BINARY JOIN TREES FOR COMPUTING MARGINALS IN THE SHENOY-SHAFER ARCHITECTURE

PRAKASH P. SHENOY

School of Business
University of Kansas
Summerfield Hall
Lawrence KS 66045-2003
pshenoy@ukans.edu

ABSTRACT

The main goal of this paper is to describe a data structure called binary join trees that are useful in computing multiple marginals efficiently in the Shenoy-Shafer architecture. We define binary join trees, describe their utility, and describe a procedure for constructing them.

Key Words: local computation, fusion algorithm, join trees, binary join trees

---

1 INTRODUCTION

The main goal of this paper is to describe a data structure called binary join trees that are useful in computing multiple marginals efficiently in the Shenoy-Shafer architecture [Shenoy and Shafer 1990]. We define binary join trees, describe their utility, and describe a procedure for constructing them.

In the last decade, much work has been done in the uncertain reasoning community on exact computation of marginals using local computation [see, e.g., Pearl 1986, Kong 1986, Lauritzen and Spiegelhalter 1988, Dempster and Kong 1988, Shenoy and Shafer 1990, Jensen *et al.* 1990, Almond 1995, Lauritzen and Jensen 1996, Shafer 1996]. The main idea behind local computation is to compute marginals of the joint distribution without actually computing the joint distribution. Local computation can be described as message passing in data structures called join trees. Join trees are also called junction trees [Jensen et al. 1990], clique trees [Lauritzen and Spiegelhalter 1988], qualitative Markov trees [Shafer *et al*. 1987], and hypertrees [Shenoy and Shafer 1990].

The efficiency of the message-passing algorithms depend on the sizes of the subsets in a join tree. The problem of finding a join tree that minimizes the size of the largest subset has been shown to be NP-complete [Arnborg *et al*. 1987]. Consequently, much attention has been devoted to finding heuristics for constructing good join trees [see, e.g., Olmsted 1983, Kong 1986, Mellouli 1987, Zhang 1988, Kjærulff 1990].

In this paper, we focus on another aspect of join trees, the number of neighbors of nodes in a join trees. If a node in a join tree has many neighbors, then it leads to much inefficiencies, especially in the Shenoy-Shafer architecture. This motivates the definition of binary join trees which is a join tree such that no node has more than three neighbors. The main idea behind a binary join tree is that all combinations are done on a binary basis, i.e., we combine functions two at a time.

Local computation has also been studied in many other domains besides uncertain reasoning such as solving systems of equations [Rose 1970], optimization [Bertele and Brioschi 1972], and relational databases [Beeri *et al*. 1983]. In order to keep the applicability of the results as wide as possible, we describe our work using the abstract framework of valuation networks [Shenoy 1989, 1992a, 1994a, 1994b, 1994c].

An outline of this paper is as follows. Section 2 introduces the framework of valuation networks. Section 3 describes three axioms that enable local computation. Section 4 describes a fusion algorithm for computing a marginal of the joint valuation. Section 5 describes the fusion algorithm in terms of message passing in join trees. Section 6 describes a message passing algorithm for computing multiple marginals—the Shenoy-Shafer architecture. Section 7 introduces the concept of binary join trees, its utility in reducing the number of combinations done in computing marginals, and a procedure for constructing them. Finally Section 8 contains concluding remarks.

2 THE VALUATION NETWORK FRAMEWORK

This section describes the abstract valuation network (VN) framework. In a VN, we represent knowledge by entities called valuations, and we make inferences using two operators called marginalization and combination that operate on valuations.

Variables and Configurations. We use the symbol Ω_X for the set of possible values of a variable X, and we call Ω_X the *state space* for X. We are concerned with a finite set Ψ of variables, and we assume that all the variables in Ψ have finite state spaces. We use upper-case Roman letters such as X, Y, Z, etc., to denote variables, and we use italicized lower-case Roman letters such as *r*, *s*, *t*, etc., to denote sets of variables.

Given a nonempty set *s* of variables, let Ω_s denote the Cartesian product of Ω_X for $X \in s$; $\Omega_s = \times\{\Omega_X \mid X \in s\}$. We call Ω_s the *state space* for *s*. We call the elements of Ω_s *configurations* of *s*. We use lower-case, bold-faced letters such as **x**, **y**, **z**, etc., to denote configurations.

It is convenient to extend this terminology to the case where the set *s* is empty. We adopt the convention that the state space for the empty set ø consists of a single configuration, and we use the symbol ♦ to name that configuration; $\Omega_\emptyset = \{\blacklozenge\}$.

Valuations. Given a subset s of variables (possible empty), there is a set ϑ_s. We call the elements of ϑ_s *valuations for s*. Let ϑ denote the set of all valuations, i.e., $\vartheta = \cup\{\vartheta_s \mid s \subseteq \Psi\}$. If σ is a valuation for s, we say s is the *domain* of σ. We use lower-case Greek letters such as ρ, σ, τ, etc., to denote valuations.

Valuations are primitives in the VN framework and as such require no definition. But, as we shall see shortly, they are entities that can be combined (with other valuations) and marginalized. Intuitively, a valuation for s represents some knowledge about the variables in s.

In probability theory, valuations are called probability potentials. A *probability potential* for r is a function $\rho: \Omega_r \rightarrow [0, 1]$. In Dempster-Shafer's belief function theory, valuations are called bpa potentials. A *bpa potential* for m is a function $\mu: 2^{\Omega_m} \rightarrow [0, 1]$, where 2^{Ω_m} denotes the set of all nonempty subsets of Ω_m. In Spohn's epistemic belief theory, valuations are called disbelief potentials. A *disbelief potential* for d is a function $\delta: \Omega_d \rightarrow \mathbf{N}$, where $\mathbf{N}$ is the set of natural numbers. In Zadeh's possibility theory, valuations are called possibility potentials. A *possibility potential* for p is a function $\pi: \Omega_p \rightarrow [0, 1]$.

Marginalization. We assume that for each nonempty $s \subseteq \Psi$, and for each $X \in s$, there is a mapping $\downarrow(s - \{X\}): \vartheta_s \rightarrow \vartheta_{s-\{X\}}$, called *marginalization* to $s - \{X\}$, such that if σ is a valuation for s, then $\sigma^{\downarrow(s-\{X\})}$ is a valuation for $s - \{X\}$. We call $\sigma^{\downarrow(s-\{X\})}$ the *marginal* of σ for $s - \{X\}$.

Intuitively, marginalization corresponds to coarsening of knowledge. If σ is a valuation for s representing some knowledge about variables in s, and $X \in s$, then $\sigma^{\downarrow(s-\{X\})}$ represents the knowledge about variables in $s - \{X\}$ implied by σ if we disregard variable X.

For probability potentials, marginalization from s to $s - \{X\}$ is addition over the state space for X. For bpa potentials, marginalization from s to $s - \{X\}$ is addition over the subsets of the state space for s that include $\{X\}$. For disbelief potentials, marginalization from s to $s - \{X\}$ is minimization over the state space for X. And for possibility potentials, marginalization from s to $s - \{X\}$ is maximization over the state space for X.

Combination. We assume there is a mapping $\otimes: \vartheta \times \vartheta \rightarrow \vartheta$, called *combination*, such that if ρ and σ are valuations for r and s, respectively, then $\rho \otimes \sigma$ is a valuation for $r \cup s$.

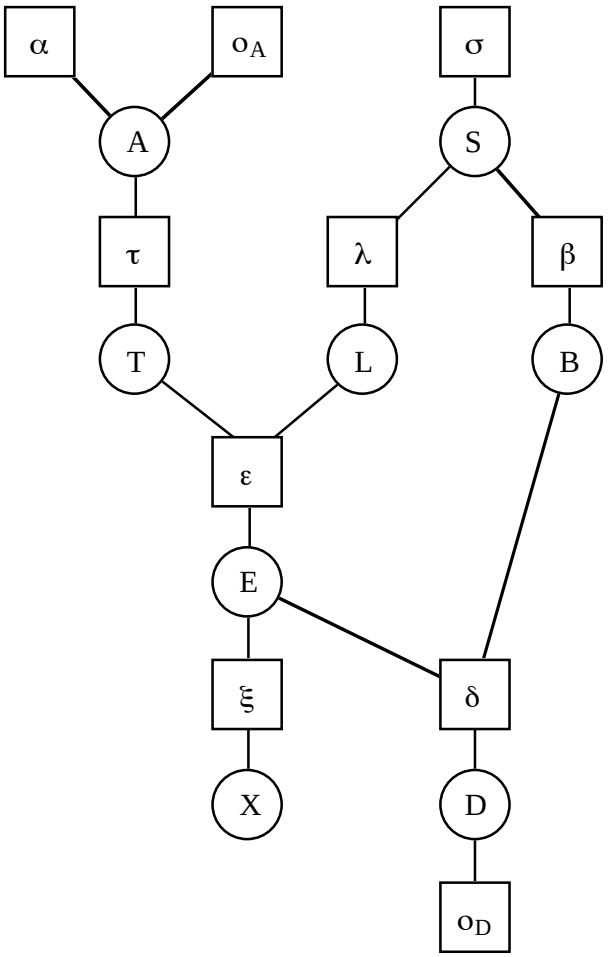
Intuitively, combination corresponds to aggregation of knowledge. If ρ and σ are valuations for r and s representing knowledge about variables in r and s, respectively, then $\rho \otimes \sigma$ represents the aggregated knowledge about variables in $r \cup s$.

For probability potentials, combination is pointwise multiplication followed by normalization if normalization is possible, and no normalization if normalization is not possible. For bpa potentials, combination is the product-intersection rule followed by normalization if normalization is possible, and no normalization if normalization is not possible. This rule is also known as *Dempster's rule* [Dempster 1966]. For disbelief potentials, combination is pointwise addition followed by normalization [Shenoy 1991a]. And for possibility potentials, combination is multiplication

followed by normalization if normalization is possible, and no normalization if normalization is not possible [Shenoy 1992b].

In summary, a *valuation network* consists of a 5-tuple $\{\Psi, \{\Omega_X\}_{X\in\Psi}, \{\tau_1, ..., \tau_m\}, \downarrow, \otimes\}$ where Ψ is a set of variable, $\{\Omega_X\}_{X\in\Psi}$ is a collection of state spaces, $\{\tau_1, ..., \tau_m\}$ is a collection of valuations, $\downarrow$ is the marginalization operator, and $\otimes$ is the combination operator.

Figure 1. The valuation network for Example 1.



Example 1. (*Chest Clinic*) Consider Lauritzen and Spiegelhalter's [1988] medical example:

> Shortness-of-breath (dyspnoea) may be due to tuberculosis, lung cancer or bronchitis, or none of them, or more than one of them. A recent visit to Asia increases the chances of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X-ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea.

This needs to be applied to the following hypothetical situation. A patient presents at a chest clinic with dyspnoea, and has recently visited Asia. The doctor would like to know the chance that each of the diseases is present.

In this example, we have eight variables—A (visit to Asia), S (Smoking), T (Tuberculosis), L (Lung cancer), B (Bronchitis), E (Either tuberculosis or lung cancer), X (positive X-ray), and D (Dyspnoea). Prior to any observations, we have eight valuations—α for {A}, σ for {S}, τ for {A, T}, λ for {S, L}, β for {S, B}, ε for {T, L, E}, ξ for {E, X}, and δ for {E, B, D}. In VNs, observations are also modeled as valuations. Suppose, for example, we observe that a patient has visited Asia recently, and is suffering from Dyspnoea. These two observations can be modeled as valuations o_A for {A} and o_D for {D}, respectively. ■

Valuation Networks. A graphical display of a valuation network is also called a *valuation network*. In a valuation network, variables are represented by circular nodes, and valuations are represented by rectangular nodes. Also, each valuation node is connected by an undirected edge to each variable node in its domain. Figure 1 shows the valuation network for Example 1.

Making Inference in VN. In a VN, the combination of all valuations is called the *joint valuation*. Given a VN, we make inferences by computing the marginal of the joint valuation for each variable of interest.

If there are n variables in a VN, and each variable has two configurations in its state space, then there are 2^n configurations of all variables. Hence, it is not computationally tractable to compute the joint valuation when there are a large number of variables. In Section 4, we describe an algorithm for computing the marginal of the joint valuation for a variable without explicitly computing the joint valuation. To ensure that this algorithm gives us the correct answers, we require that marginalization and combination satisfy some axioms. The axioms are described in the next section.

3 AXIOMS FOR LOCAL COMPUTATION

In this section, we state three axioms that enable efficient local computation of marginals of the joint valuation. These axioms were first formulated by Shenoy and Shafer [1990]. Other axiomatic systems have been defined by Shafer [1991] and Cano *et al*. [1993].

Axiom A1 (*Order of deletion does not matter*): Suppose σ is a valuation for s, and suppose $X_1, X_2 \in s$. Then

$$(\sigma^{\downarrow(s - \{X_1\})})^{\downarrow(s - \{X_1, X_2\})} = (\sigma^{\downarrow(s - \{X_2\})})^{\downarrow(s - \{X_1, X_2\})}.$$

Axiom A2 (*Commutativity and associativity of combination*): Suppose ρ, σ, and τ are valuations for r, s, and t, respectively. Then

$$\rho \otimes \sigma = \sigma \otimes \rho, \text{ and } \rho \otimes (\sigma \otimes \tau) = (\rho \otimes \sigma) \otimes \tau.$$

Axiom A3 (*Distributivity of marginalization over combination*): Suppose ρ and σ are valuations for r and s, respectively, suppose $X \in s$, and suppose $X \notin r$. Then

$$(\rho \otimes \sigma)^{\downarrow((r \cup s) - \{X\})} = \rho \otimes (\sigma^{\downarrow(s - \{X\})}).$$

If we regard marginalization as a coarsening of a valuation by deleting variables, then Axiom A1 says that the order in which the variables are deleted does not matter. Thus variables can be deleted in any order. One implication of this axiom is that $(\sigma^{\downarrow(s - \{X_1\})})^{\downarrow(s - \{X_1, X_2\})}$ can be written simply as $\sigma^{\downarrow(s - \{X_1, X_2\})}$, i.e., we need not indicate the order in which the variables are deleted.

Axiom A2 allows us to combine valuations in any order. One implication of this axiom is that when we have multiple combinations of valuations, we can write it without using parenthesis. For example, $(...((\sigma_1 \otimes \sigma_2) \otimes \sigma_3) \otimes ... \otimes \sigma_m)$ can be written simply as $\otimes\{\sigma_i \mid i = 1, ..., m\}$ or as $\sigma_1 \otimes ... \otimes \sigma_m$, i.e., we need not indicate the order in which the combinations are carried out. Another implication of Axiom A2 is that the set of valuations and the combination operator can be regarded as a commutative semigroup.

Axiom A3 is the axiom that makes local computation possible. Axiom A3 states that computation of $(\rho \otimes \sigma)^{\downarrow((r \cup s) - \{X\})}$ can be accomplished without having to compute $\rho \otimes \sigma$. Notice that $\rho \otimes \sigma$ is a valuation for $r \cup s$ whereas $\rho \otimes (\sigma^{\downarrow(s - \{X\})})$ is a valuation for $(r \cup s) - \{X\}$.

4 COMPUTING A MARGINAL USING THE FUSION ALGORITHM

In this section, we describe the fusion algorithm for computing the marginal for a variable using local computation [Shenoy 1992a]. The fusion algorithm was first described by Cannings *et al*. [1978] in the context of probability models in genetics, and they called their procedure "peeling."

Suppose $\{\{\tau_1, ..., \tau_m\}, \downarrow, \otimes\}$ is a VN with m valuations. Suppose that marginalization and combination satisfy the three axioms stated in Section 4. Suppose we need to compute the marginal of the joint valuation for subset t, $(\tau_1 \otimes ... \otimes \tau_m)^{\downarrow t}$. The basic idea of the fusion algorithm is to successively delete all variables in $\Psi - t$ from the VN. The variables may be deleted in any sequence. Axiom A1 tells us that all deletion sequences lead to the same answer. But, different deletion sequences may involve different computational efforts. We will comment on good deletion sequences at the end of this section.

First let us deal with the case of deleting one variable. Suppose we have a set of k valuations $\sigma_1, ..., \sigma_k$. Suppose σ_i is a valuation for s_i. Then $\sigma_1 \otimes ... \otimes \sigma_k$ is a valuation for $s_1 \cup ... \cup s_k$. Let $Y \in s_1 \cup ... \cup s_k$ and suppose we wish to delete Y from $\sigma_1 \otimes ... \otimes \sigma_k$. Lemma 1 tells us that we can accomplish this using local computation.

Lemma 1. Under the assumptions of the previous paragraph,

$$(\sigma_1 \otimes ... \otimes \sigma_k)^{\downarrow(s_1 \cup ... \cup s_k - \{Y\})} = \sigma^{\downarrow(s - \{Y\})} \otimes (\otimes\{\sigma_i \mid Y \notin s_i\})$$

where $\sigma = \otimes\{\sigma_i \mid Y \in s_i\}$, and $s = \cup\{s_i \mid Y \in s_i\}$.

Lemma 1 follows directly from Axiom A3 by letting $\rho = \otimes\{\sigma_i \mid Y \notin s_i\}$ and $\sigma = \otimes\{\sigma_i \mid Y \in s_i\}$.

If $\sigma = \sigma_1 \otimes ... \otimes \sigma_k$, then we say $\sigma_1, ..., \sigma_k$ are the *factors* of σ. If we compare the factors of $(\sigma_1 \otimes ... \otimes \sigma_k)^{\downarrow(s_1 \cup ... \cup s_k - \{Y\})}$ with the factors of $\sigma_1 \otimes ... \otimes \sigma_k$, we observe that in deleting Y, the

factors that do not contain Y in their domains remain unchanged and the factors that contain Y in their domains are first combined and then Y is deleted from the combination. We call this operation *fusion*. A formal definition is as follows. Consider a set of k valuations $\sigma_1, ..., \sigma_k$. Suppose σ_i is a valuation for s_i. Let $\text{Fus}_Y\{\sigma_1, ..., \sigma_k\}$ denote the set of valuations after fusing the valuations in the set $\{\sigma_1, ..., \sigma_k\}$ with respect to variable Y. Then

$$\text{Fus}_Y\{\sigma_1, ..., \sigma_k\} = \{\sigma^{\downarrow(s - \{Y\})}\} \cup \{\sigma_i \mid Y \notin s_i\}$$

where $\sigma = \otimes\{\sigma_i \mid Y \in s_i\}$, and $s = \cup\{s_i \mid Y \in s_i\}$. After fusion, the set of valuations is changed as follows. All valuations that have Y in their domains are combined, and the resulting valuation is marginalized such that Y is eliminated from its domain. The valuations that do not have Y in their domains remain unchanged.

Using the definition of fusion, we can express the result of Lemma 1 as follows:

$$(\sigma_1 \otimes ... \otimes \sigma_k)^{\downarrow(s_1 \cup ... \cup s_k - \{Y\})} = \otimes \text{Fus}_Y\{\sigma_1, ..., \sigma_k\}$$

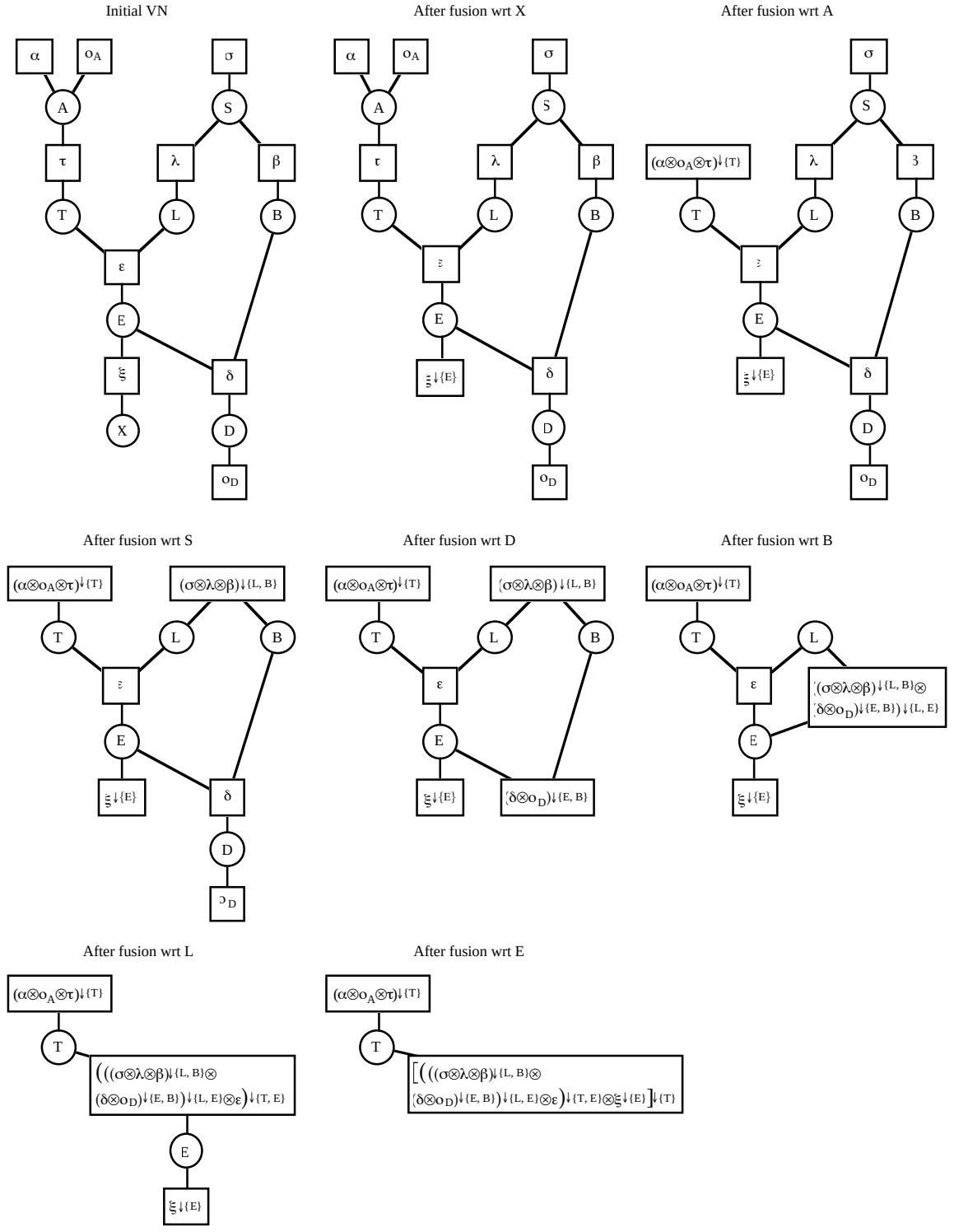
By successively deleting variables, we can find the marginal of the joint for any subset. Axiom A1 tells us that we can use any sequence. This result is stated formally as follows.

Theorem 1 (*Fusion Algorithm*) [Shenoy 1992a]. Suppose $\{\{\tau_1, ..., \tau_m\}, \downarrow, \otimes\}$ is a VN where τ_i is a valuation for t_i, and suppose $\downarrow$ and $\otimes$ satisfy axioms A1–A3. Let Ψ denote $t_1 \cup ... \cup t_m$. Suppose $t \subseteq \Psi$, and suppose $X_1X_2...X_n$ is a sequence of variables in $\Psi - t$. Then

$$(\tau_1 \otimes ... \otimes \tau_m)^{\downarrow t} = \otimes \left\{ \text{Fus}_{X_n}\left\{ ... \text{Fus}_{X_2}\{\text{Fus}_{X_1}\{\tau_1, ..., \tau_m\}\}\right\}\right\}.$$

To illustrate Theorem 1, consider the VN of Example 1. Suppose we need to compute the marginal of the joint for {T}, i.e., $(\alpha \otimes o_A \otimes \sigma \otimes \tau \otimes \lambda \otimes \beta \otimes \varepsilon \otimes \xi \otimes \delta \otimes o_D)^{\downarrow\{T\}}$. Consider the deletion sequence XASDBLE. First, after fusion with respect to X, we have $\{\alpha, \tau, \sigma, \lambda, \beta, \varepsilon, \xi^{\downarrow\{E\}}, \delta, o_A, o_D\}$. Second, after fusion with respect to A, we have $\{(\alpha \otimes o_A \otimes \tau)^{\downarrow\{T\}}, \sigma, \lambda, \beta, \varepsilon, \xi^{\downarrow\{E\}}, \delta, o_D\}$. Third, after fusion with respect to S, we have $\{(\alpha \otimes o_A \otimes \tau)^{\downarrow\{T\}}, (\sigma \otimes \lambda \otimes \beta)^{\downarrow\{L, B\}}, \varepsilon, \xi^{\downarrow\{E\}}, \delta, o_D\}$. Fourth, after fusion with respect to D, we have $\{(\alpha \otimes o_A \otimes \tau)^{\downarrow\{T\}}, (\sigma \otimes \lambda \otimes \beta)^{\downarrow\{L, B\}}, \varepsilon, \xi^{\downarrow\{E\}}, (\delta \otimes o_D)^{\downarrow\{E, B\}}\}$. Fifth, after fusion with respect to B, we have $\{(\alpha \otimes o_A \otimes \tau)^{\downarrow\{T\}}, ((\sigma \otimes \lambda \otimes \beta)^{\downarrow\{L, B\}} \otimes (\delta \otimes o_D)^{\downarrow\{E, B\}})^{\downarrow\{L, E\}}, \varepsilon, \xi^{\downarrow\{E\}}\}$. Sixth, after fusion with respect to L, we have $\{(\alpha \otimes o_A \otimes \tau)^{\downarrow\{T\}}, (((\sigma \otimes \lambda \otimes \beta)^{\downarrow\{L, B\}} \otimes (\delta \otimes o_D)^{\downarrow\{E, B\}})^{\downarrow\{L, E\}} \otimes \varepsilon)^{\downarrow\{T, E\}}, \xi^{\downarrow\{E\}}\}$. Finally, after fusion with respect to E, we have $\{(\alpha \otimes o_A \otimes \tau)^{\downarrow\{T\}}, [(((\sigma \otimes \lambda \otimes \beta)^{\downarrow\{L, B\}} \otimes (\delta \otimes o_D)^{\downarrow\{E, B\}})^{\downarrow\{L, E\}} \otimes \varepsilon)^{\downarrow\{T, E\}} \otimes \xi^{\downarrow\{E\}}]^{\downarrow\{T\}}\}$. Theorem 4.1 tells us that $(\alpha \otimes o_A \otimes \tau)^{\downarrow\{T\}} \otimes [(((\sigma \otimes \lambda \otimes \beta)^{\downarrow\{L, B\}} \otimes (\delta \otimes o_D)^{\downarrow\{E, B\}})^{\downarrow\{L, E\}} \otimes \varepsilon)^{\downarrow\{T, E\}} \otimes \xi^{\downarrow\{E\}}]^{\downarrow\{T\}} = (\alpha \otimes o_A \otimes \sigma \otimes \tau \otimes \lambda \otimes \beta \otimes \varepsilon \otimes \xi \otimes \delta \otimes o_D)^{\downarrow\{T\}}$. The fusion algorithm is shown graphically in Figure 2.

Figure 2. The fusion algorithm for the VN of Example 1 using deletion sequence XASDBLE.



Deletion Sequences. Different deletion sequences may involve different computational efforts. Finding an optimal deletion sequence is a secondary optimization problem that has been shown to be NP-complete [Arnborg *et al*. 1987]. But, there are several heuristics for finding good deletion sequences [Kong 1986, Mellouli 1987, Zhang 1988, Kjærulff 1990].

One such heuristic is called one-step-look-ahead [Olmsted 1983, Kong 1986]. This heuristic tells us which variable to delete next. As per this heuristic, the variable that should be deleted next is one that leads to combination over the smallest state space with ties broken arbitrarily. In particular, if a variable appears in the domain of only one valuation, then such variables should be deleted first as no combination is involved.

For example, in the VN of Example 1, if we assume that each variable has a state space consisting of two configurations, then this heuristic would pick X for the first deletion since deletion of X involves no combination whereas deleting any other variable involves some combination. The deletion sequence XASDBLE used to illustrate the fusion algorithm is one of the many deletion sequences suggested by the one-step-look-ahead heuristic.

5 FUSION ALGORITHM AS MESSAGE PASSING IN JOIN TREES

If we can compute the marginal of the joint valuation for one variable, then we can compute the marginals for all variables. We simply compute them one after the other. It is obvious, however, that this will involve much repetition of effort. To avoid this repetition, we will describe the fusion algorithm as message-passing in join trees. A join tree can be thought of as a data structure that allows us to organize the computation, and more importantly, that allows us to cache the computations to avoid repetition of effort. In the next section, we will describe how the join tree data structure allows us to efficiently compute multiple marginals.

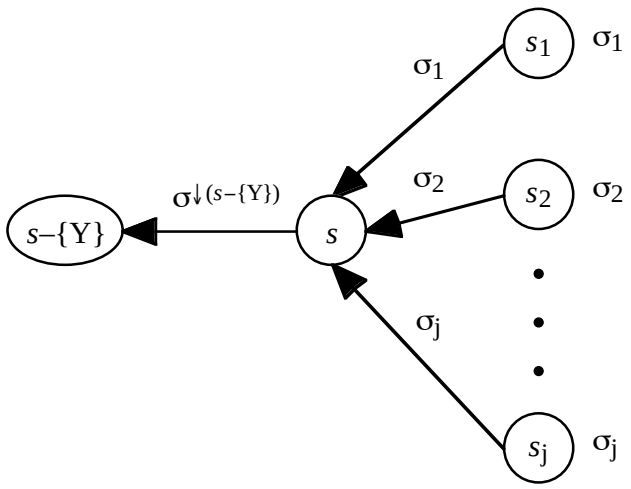
Join Trees. A *join tree* is a tree whose nodes are subsets of Ψ such that if a variable is in two distinct nodes, then it is in every node on the path between the two nodes [Maier 1983]. As we will see, join trees are useful data structures to cache computation.

Consider again the definition of fusion. Suppose we have valuations $\sigma_1, ..., \sigma_k$, where σ_i is a valuation for s_i. Suppose the valuations are labeled such that $s_1, .., s_j$ contain Y and $s_{j+1}, ..., s_k$ do not contain Y. Then

$$\mathrm{Fus}_Y\{\sigma_1, ..., \sigma_k\} = \{\sigma^{\downarrow(s - \{Y\})}\} \cup \{\sigma_{j+1}, ..., \sigma_k\}$$

where $\sigma = \sigma_1 \otimes ... \otimes \sigma_j$, and $s = s_1 \cup ... \cup s_j$. We will now describe the fusion operation as message-passing in a rooted join tree.

Figure 3. Fusion as message-passing in a rooted join tree.



Suppose that s_1, ..., s_k, s, and $s - \{Y\}$ are all distinct subsets of Ψ. We can describe the fusion operation as follows. We imagine that s_1, ..., s_j, s, and $s - \{Y\}$ are all nodes connected together in a rooted join tree as shown in Figure 3 where the arrows point toward the root. Nodes s_1, ..., s_j have stored in them the valuations σ_1, ..., σ_j, respectively. Nodes s and $s - \{Y\}$ have nothing stored in them. First nodes s_1, ..., s_j send messages to their inward neighbor s consisting of the valuations stored in them. Next, node s first combines all messages it receives from its outward neighbors, marginalizes the combination σ to $\sigma^{\downarrow(s-\{Y\})}$, and sends this valuation as a message to its inward neighbor $s - \{Y\}$. At the end of the fusion operation, we are only concerned with node $s - \{Y\}$ with a message $\sigma^{\downarrow(s-\{Y\})}$.

If $s - \{Y\}$ is not distinct from s_{j+1}, ..., s_k, say $s - \{Y\} = s_{j+1}$, then the only change we need to make is that node $s - \{Y\} = s_{j+1}$ will have valuation σ_{j+1} stored in it, but this valuation plays no role in fusion with respect to Y. Also, if not all s_1, ..., s_j are distinct, say $s_1 = s_2$ then we have only one node for s_1 and s_2, and we imagine that this node either has two valuations σ_1 and σ_2 stored in it or that it has one valuation $\sigma_1 \otimes \sigma_2$ stored in it. The basic idea is that any subset of variables should appear only once in the rooted tree.[1]

In all cases, the rule for messages is stated as Rule 1.

Rule 1 (*Messages*). Each node sends a message to its inward neighbor (toward the root). The message that a node sends to its inward neighbor is as follows. First it combines all messages it receives from its outward neighbors together with its own valuation (if any).

[1] There is no problem with having a subset appear more than once in a rooted tree. But since there is no need for it at this stage, we try to avoid it. In Section 7, we will describe binary join trees that have multiple copies of subsets to make the message passing algorithm more efficient.

Next it marginalizes the combination to the intersection of itself with its inward neighbor.

Each node sends a message when it has received messages from all its outward neighbors.

Leaves have no outward neighbors and can send messages right away.

As we continue to delete variables using fusion, we recursively grow the rooted join tree. When we have deleted all but one variable, say X, we have a rooted join tree with {X} as the root. A formal description of the process of constructing a join tree is as follows.

Rooted Join Tree Construction. Let Ψ denote the set of variables, let Φ denote the set of subsets of variables for which we have valuations, let $|\Phi|$ denote the number of elements of Φ, let N denote the nodes of the rooted join tree, and let E denote the edges of the rooted join tree. A procedure in pseudocode for constructing a rooted join tree (N, E) for computing the marginal for {X} is as follows [Shenoy 1991b].

Procedure for Constructing a Rooted Join Tree

INPUT: Ψ, Φ

OUTPUT: N, E

INITIALIZATION

$\Psi_u \leftarrow \Psi - \{X\}$ /* Ψ_u denotes the set of variables in Ψ that have not yet been deleted */

$\Phi_u \leftarrow \Phi$ /* Φ_u denotes the subsets in Φ that have not yet been arranged in the join tree */

$N \leftarrow \emptyset$

$E \leftarrow \emptyset$

DO WHILE $|\Phi_u| > 1$ /* If $|\Phi_u| = 1$, then we are done */

Pick a variable $Y \in \Psi_u$ /* using some heuristic */

$s \leftarrow \cup\{s_i \in \Phi_u \mid Y \in s_i\}$.

$N \leftarrow N \cup \{s_i \in \Phi_u \mid Y \in s_i\} \cup \{s\} \cup \{s - \{Y\}\}$

$E \leftarrow E \cup \{(s_i, s) \mid s_i \in [\Phi_u - \{s\}], Y \in s_i\} \cup \{(s, s - \{Y\})\}$

$\Psi_u \leftarrow \Psi_u - \{Y\}$

$\Phi_u \leftarrow [\Phi_u - \{s_i \in \Phi_u \mid Y \in s_i\}] \cup \{s - \{Y\}\}$

END DO

$N \leftarrow N \cup \Phi_u$ /* This is needed for the trivial case $|\Phi| = 1$ */

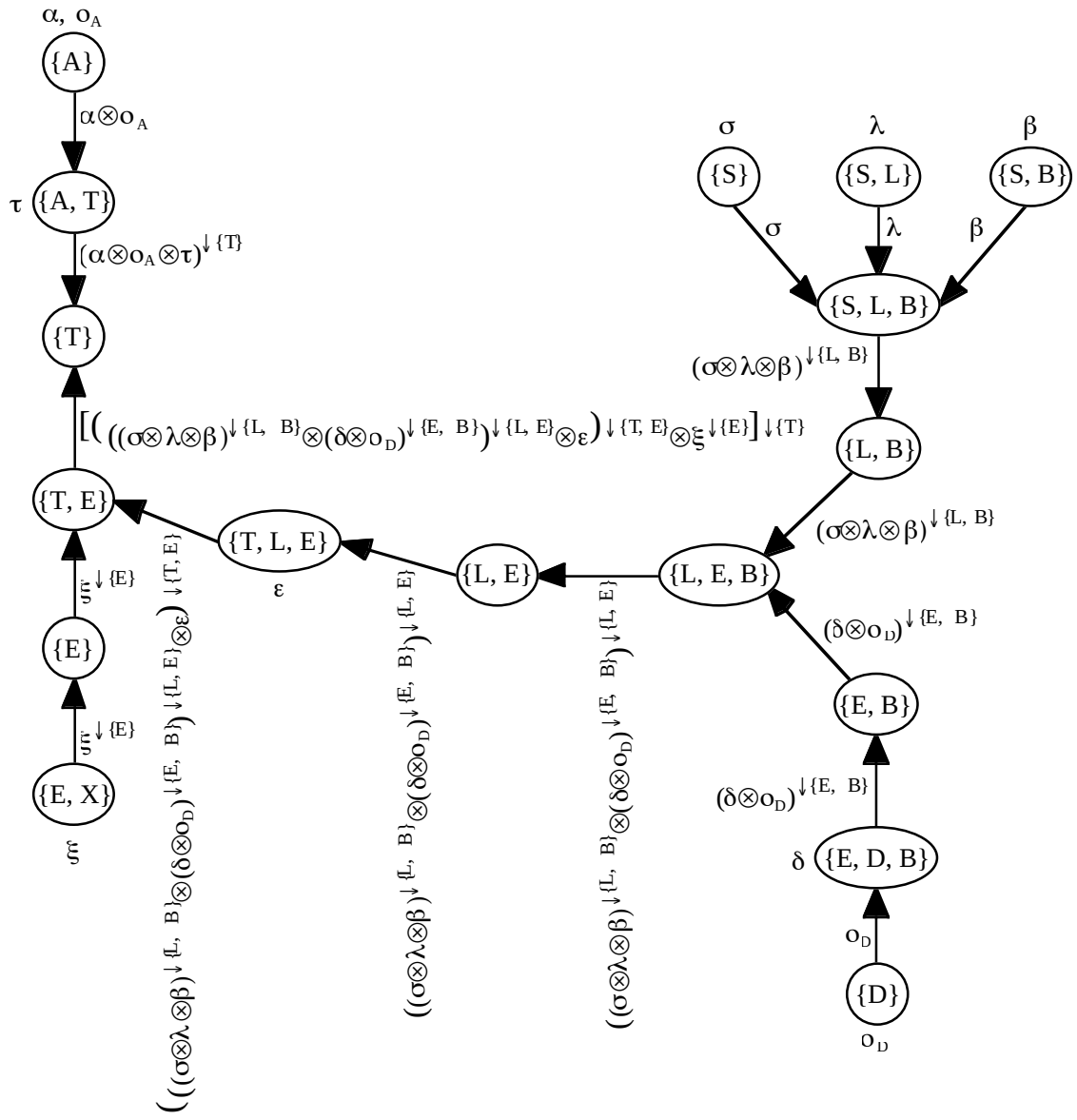
END

The root node in the rooted join tree has no inward neighbor and does not send a message to any node. Instead, the root node simply combines all messages it receives from its outward neighbors resulting in the desired marginal (as per Theorem 1). We describe this as Rule 2.

Rule 2 (*Marginal*).When the root node has received a message from each of its outward neighbors, it combines all messages together with its own valuation and reports the result as its marginal.

The construction of the join tree and Rules 1 and 2 completely describe the fusion algorithm in terms of message passing in join trees. Figure 4 illustrates the construction of the join tree and the messages for computing the marginal for T in the Chest Clinic problem (Example 1).

Figure 4. Fusion algorithm as message passing in a join tree. Valuations shown adjacent to nodes are the inputs. Valuation shown adjacent to edges are the messages.



6 COMPUTING MULTIPLE MARGINALS

In this section, we show how we can adapt the message-passing algorithm of the previous section so we can compute multiple marginals of the joint valuation.

The join tree constructed in the preceding section was rooted only to indicate the direction of the messages. The directions of the edges served no other purpose. If the join tree constructed for the computation of the marginal for variable X also contains a node corresponding to singleton subset {Z}, then the same join tree (with some of the directions changed) will serve for the computation of the marginal for Z. We simply redirect some of the edges so that {Z} is now the root (instead of {X}).

For example, for the valuation network of Example 1, if we would like the marginal of the joint for E instead of T, we have to reverse the directions of two edges—({T, E}, {T}), and ({E}, {T, E}) (see Figure 4). Suppose we have saved the messages computed for the marginal for {T}. Then, to compute the marginal for {E}, we have to compute only two new messages—the message from {T} to {T, E}, and the message from {T, E} to {E}. If we use Rule 1 to compute these messages, then the message from {T} to {T, E} is $(\alpha\otimes o_A\otimes\tau)^{\downarrow\{T\}}$. And the message from {T, E} to {E} is $\left[(\alpha\otimes o_A\otimes\tau)^{\downarrow\{T\}}\otimes\left(\left((\sigma\otimes\lambda\otimes\beta)^{\downarrow\{L,B\}}\otimes(\delta\otimes o_D)^{\downarrow\{E,B\}}\right)^{\downarrow\{L,E\}}\otimes\varepsilon\right)^{\downarrow\{T,E\}}\right]^{\downarrow E}$. If {E} now uses Rule 2, the marginal for {E} is given by $\left[(\alpha\otimes o_A\otimes\tau)^{\downarrow\{T\}}\otimes\left(\left((\sigma\otimes\lambda\otimes\beta)^{\downarrow\{L,B\}}\otimes(\delta\otimes o_D)^{\downarrow\{E,B\}}\right)^{\downarrow\{L,E\}}\otimes\varepsilon\right)^{\downarrow\{T,E\}}\right]^{\downarrow E}\otimes\xi^{\downarrow E}$.

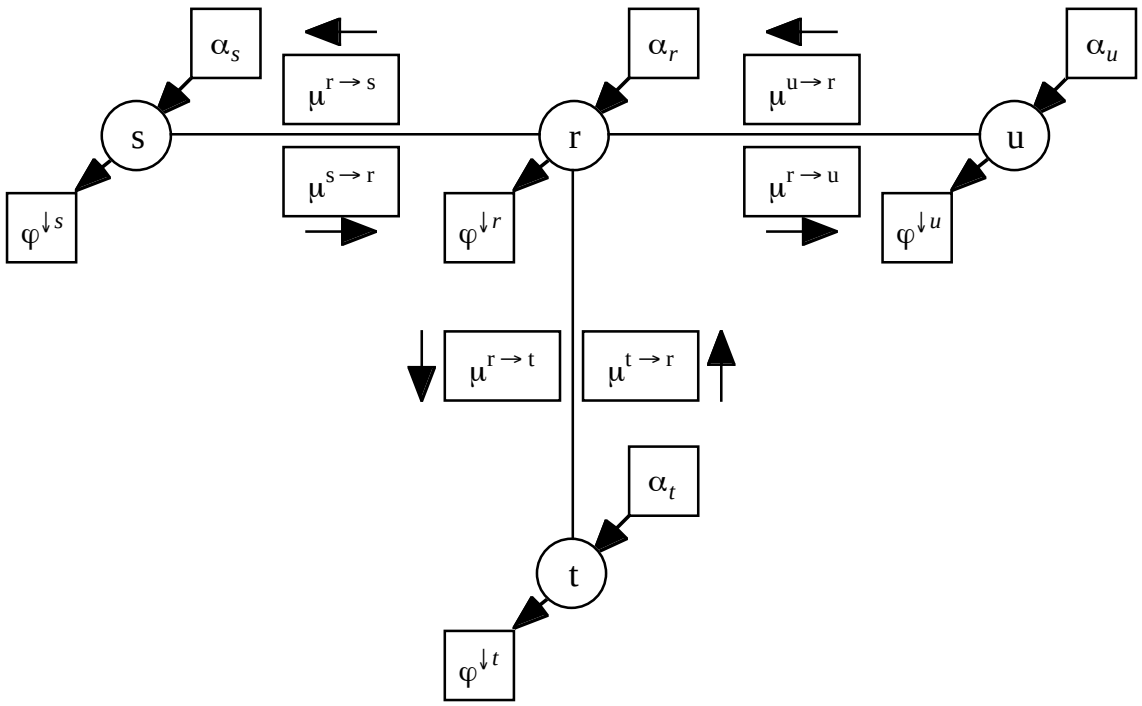
Instead of directing the edges of the join tree, it will be easier to leave the edges of the join tree undirected, and simply associate directions with the messages. Also, if each node sends a message to each of its neighbors, then we can compute the marginals for every subset in the join tree. We do this by changing the two rules as follows.

Rule 1' (*Messages*) Each node sends a message to each of its neighbors. Suppose $\mu^{r\to s}$ denotes the message from r to s, suppose $N(r)$ denotes the neighbors of r in the join tree, and suppose the valuation associated with node r is denoted by α_r, then the message from node r to its neighboring node s is given as follows:

$$\mu^{r\to s} = \left(\otimes\left\{\mu^{t\to r} \mid t\in(N(r)-\{s\})\right\}\otimes\alpha_r\right)^{\downarrow r\cap s} \tag{7.1}$$

In words, the message that r send to its neighbor s is the combination of all messages that r receives from its other neighbors together with its own valuation suitably marginalized. Regarding timing, it is clear that node r sends a message to neighbor s only when r has received a message from each of its other neighbors. A leaf of the join tree has only one neighbor, and therefore it can send a message to its neighbor right away without waiting for any messages.

Figure 5. An architecture for computing multiple marginals.



Rule 2' (*Marginals*). When a node r has received a message from each of its neighbors, it combines all messages together with its own valuation and reports the results as its marginal. If φ denotes the joint valuation, then

$$\varphi^{\downarrow r} = \otimes\{\mu^{t \to r} \mid t \in \mathrm{N}(r)\} \otimes \alpha_r \tag{7.2}$$

Using Rules 1' and 2', we can compute the marginal of the joint for each subset in the join tree. Thus, if we know the subsets for which we need marginals, we simply include these subsets (along with the subsets for which we have valuations) in Φ during the construction of the join tree.

Rules 1' and 2' suggest an architecture as shown in Figure 5. Each node in the join tree would have two storage registers, one for the input valuation, and one for reporting the marginal of the joint. Also, each edge in the join tree would have two storage registers for the two messages, one in each direction.

7 BINARY JOIN TREES

In this section, we introduce the concept of a binary join tree. Binary join trees are important from a computational viewpoint since they reduce the number of computations involved in computing marginals.

A *binary join tree* is a join tree such that no node has more than three neighbors. The join tree shown in Figure 4 for the Chest Clinic problem is not binary since node {S, L, B} has four neighbors.

To explain the importance of a binary join tree, we will describe by means of an example, the inefficiencies of computation in a non-binary join tree.

Example 2. Consider a valuation network consisting of four variables W, X, Y, and Z, and four valuations α for {W, X}, β for {W, Y}, γ for {W, Z}, and δ for {X, Y, Z}. A non-binary join tree with the messages between adjacent nodes is shown in Figure 6. We make some observations about inefficiencies of computation in this non-binary join tree.

1. (*Domain of Combination*). First, consider the message $(\alpha\otimes\beta\otimes\gamma)^{\downarrow\{X, Y, Z\}}$ (from {W, X, Y, Z} to {X, Y, Z}). The computation of this message involves combination of the valuations α, β, and γ on the domain {W, X, Y, Z}. In general, combination of m valuations on a domain with n configurations involves computation that is linear in m–1 and a monotonic increasing function of n. Suppose that W has 2 states, X has 3 states, and Y has 4 states and Z has 5 states. Then the

Figure 6. A non-binary join tree with messages for the VN in Example 2.

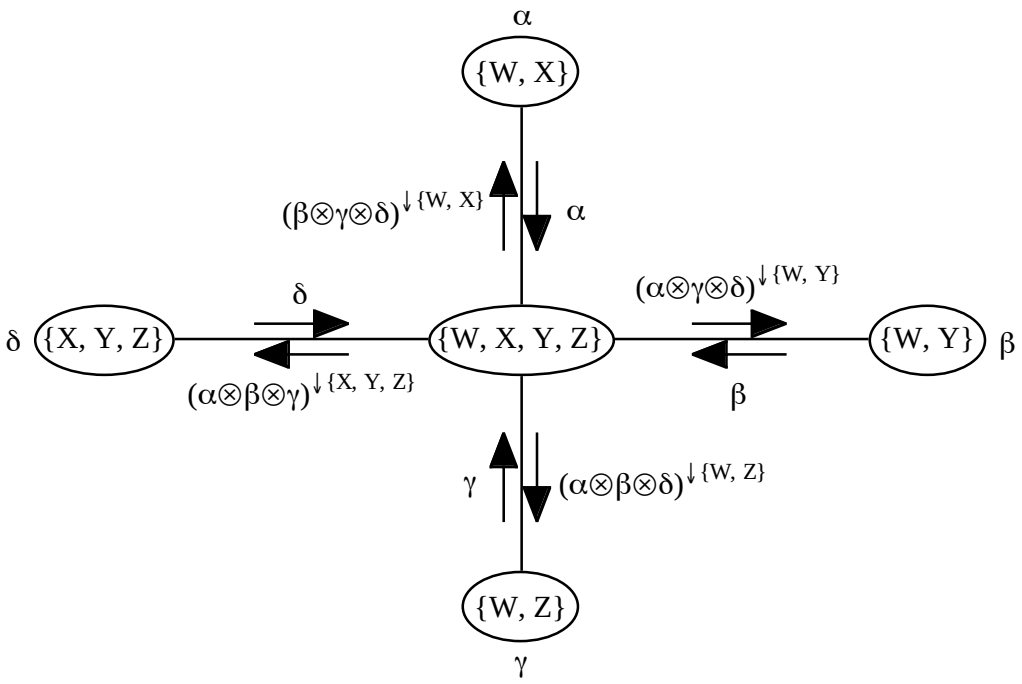
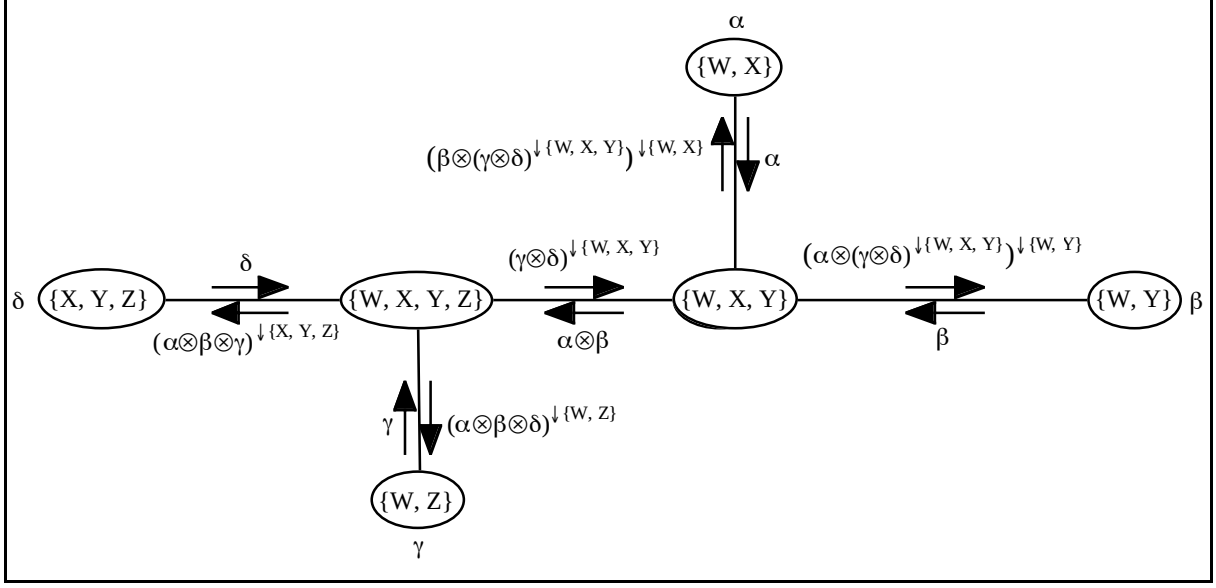


Figure 7. A binary join tree with messages for the VN in Example 2.



state space of {W, X, Y, Z} has 120 configurations. Instead of combining α, β, and γ on the domain {W, X, Y, Z} that has 120 configurations, it is more efficient to first combine α and β on domain {W, X, Y} with 24 configurations, and next combine α⊗β with γ on the domain {W, X, Y, Z} with 120 configurations. A similar observation can be made for the message $(\alpha\otimes\beta\otimes\delta)^{\downarrow\{W, Z\}}$.

2. (*Non-Local Combination*). Second, consider the message $(\beta\otimes\gamma\otimes\delta)^{\downarrow\{W, X\}}$. Notice that Z is in the domain of γ and δ, but not in the domain of β. Thus it follows from Axiom A3 that $(\beta\otimes\gamma\otimes\delta)^{\downarrow\{W, X\}} = (\beta\otimes(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}})^{\downarrow\{W, X\}}$. It is computationally more efficient to compute $(\beta\otimes(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}})^{\downarrow\{W, X\}}$ than to compute $(\beta\otimes\gamma\otimes\delta)^{\downarrow\{W, X\}}$. Similarly, instead of computing $(\alpha\otimes\gamma\otimes\delta)^{\downarrow\{W, Y\}}$, it is more efficient to compute instead $(\alpha\otimes(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}})^{\downarrow\{W, Y\}}$.

3. (*Repetition of Combinations*). Third, consider the messages $(\alpha\otimes\beta\otimes\gamma)^{\downarrow\{X, Y, Z\}}$ and $(\alpha\otimes\beta\otimes\delta)^{\downarrow\{W, Z\}}$. Notice that if these two messages are computed separately, then the combination of α and β is repeated. Also for messages $(\beta\otimes\gamma\otimes\delta)^{\downarrow\{W, X\}}$ and $(\alpha\otimes\gamma\otimes\delta)^{\downarrow\{W, Y\}}$, the combination of γ and δ is repeated [Xu 1991, Xu and Kennes 1994].

Now consider a binary join tree for the same VN as shown in Figure 7. Compared to the non-binary join tree of Figure 6, the binary join tree has an additional node {W, X, Y} and an additional edge ({W, X, Y}, {W, X, Y, Z}).

First, notice that α⊗β is computed on the domain {W, X, Y} (as a message from {W, X, Y} to {W, X, Y, Z}) before we compute $(\alpha\otimes\beta\otimes\gamma)^{\downarrow\{X, Y, Z\}}$ (as a message from {W, X, Y, Z} to {X, Y, Z}) and $(\alpha\otimes\beta\otimes\delta)^{\downarrow\{W, Z\}}$ (as a message from {W, X, Y, Z} to {W, Z}). Thus we avoid combining valuations on domains bigger than is necessary.

Second, instead of computing $(\beta\otimes\gamma\otimes\delta)^{\downarrow\{W, X\}}$, we compute $(\beta\otimes(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}})^{\downarrow\{W, X\}}$, and instead of computing $(\alpha\otimes\gamma\otimes\delta)^{\downarrow\{W, Y\}}$, we compute $(\alpha\otimes(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}})^{\downarrow\{W, Y\}}$. Thus the messages are computed locally.

Third, the combination $(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}}$ that appears in messages $(\beta\otimes(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}})^{\downarrow\{W, X\}}$ and $(\alpha\otimes(\gamma\otimes\delta)^{\downarrow\{W, X, Y\}})^{\downarrow\{W, Y\}}$ is computed only once. Also, the combination $\alpha\otimes\beta$ is computed only once for the messages $(\alpha\otimes\beta\otimes\gamma)^{\downarrow\{X, Y, Z\}}$ and $(\alpha\otimes\beta\otimes\delta)^{\downarrow\{W, Z\}}$. Thus we avoid repetition of combinations.

For these three reasons, binary join trees are a more efficient way to organize the computations than non-binary join trees. ■

How does one construct a binary join tree? We will describe a technique based on the idea of binary fusion.

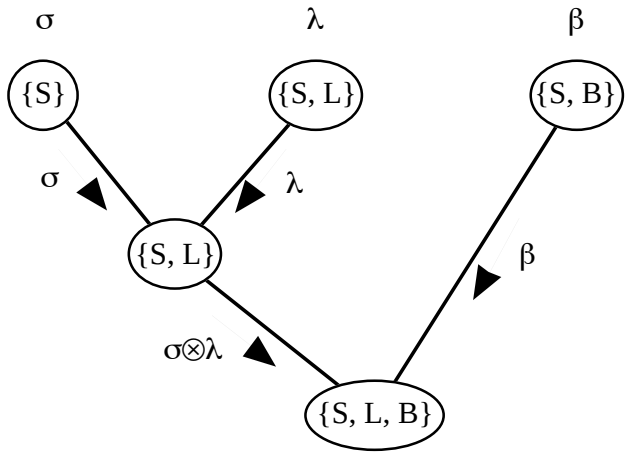
Binary Fusion. Consider again the definition of fusion. Suppose we have valuations σ_1, ..., σ_k, where σ_i is a valuation for s_i. Suppose the valuations are labeled such that s_1, .., s_j contain Y and s_{j+1}, ..., s_k do not contain Y. Then

$$\text{Fus}_Y\{\sigma_1, ..., \sigma_k\} = \{\sigma^{\downarrow(s - \{Y\})}\}\cup\{\sigma_{j+1}, ..., \sigma_k\}$$

where $\sigma = \sigma_1\otimes...\otimes\sigma_j$, and $s = s_1\cup...\cup s_j$. In binary fusion, we compute σ by combining valuations two at a time recursively as follows. We start with the set of valuations we need to combine to compute σ, namely $\{\sigma_1, ..., \sigma_j\}$. Let $\| s_i \|$ denote the number of configurations in the state space of s_i. Suppose that the subsets are labeled such that $\| s_1\cup s_2 \| =$ ARGMIN $\{\| s_p\cup s_q \| \mid 1 \le p, q \le j$. Then we first combine $\sigma_1\otimes\sigma_2$. Now the set of valuations that need to be combined is $\{\sigma_1\otimes\sigma_2, \sigma_3, ..., \sigma_j\}$. We repeat this procedure recursively till we end with a set of one valuation, $\{\sigma\}$. Binary fusion is essentially fusion with the added requirement that all combinations are done on a binary basis, i.e., two at a time.

As in the case of fusion, we can implement binary fusion as a message passing scheme in join trees. The join tree that is suggested by binary fusion is, of course, binary. For example, consider the Chest Clinic problem in Example 1. After fusion with respect to X and A we have valuations $\{(\alpha\otimes o_A\otimes\tau)^{\downarrow\{T\}}, \sigma, \lambda, \beta, \varepsilon, \xi^{\downarrow\{E\}}, \delta, o_D\}$. Next we wish to do fusion with respect to S. Ordinary fusion with respect to S leads to a non-binary join tree (see Figure 4). If we do binary fusion with respect to S, then to combine σ, λ, and β, we first combine σ and λ, and then we combine $\sigma\otimes\lambda$ and β. We can represent this as a message passing scheme in a binary join tree as shown in Figure 8.

Figure 8. Binary fusion as message passing in a binary join tree.



In the above procedure, notice that some subsets may appear more than once in the join tree. For example, $s_1 \cup s_2$ may be identical with, say, s_2. This poses no problem in practice. However, to accommodate this, we need to consider the set of nodes of a join tree as a multiset, i.e., a set of indexed subsets of Ψ (two identical subsets corresponding to distinct nodes will be identified unambiguously with their indices which will be different).

Binary Join Tree Construction. We will now formally describe a procedure in pseudocode that describes a construction of a binary join tree suggested by binary fusion.

Let Ψ denote the set of variables, let Φ denote the multiset of subsets of variables for which we have valuations or the subsets for which we need marginals, let N denote the multiset of nodes of the binary join tree, and E denote the edges of the binary join tree, let |Φ| denote the number of elements of set Φ, and let || *s* || denote the number of elements of the state space of subset *s*. We assume the subsets in Φ are indexed 1, ..., k, where k = |Φ|, i.e., $\Phi = \{s_1, ..., s_k\}$. A procedure in pseudocode for constructing a binary join tree (N, E) using binary fusion is as follows.

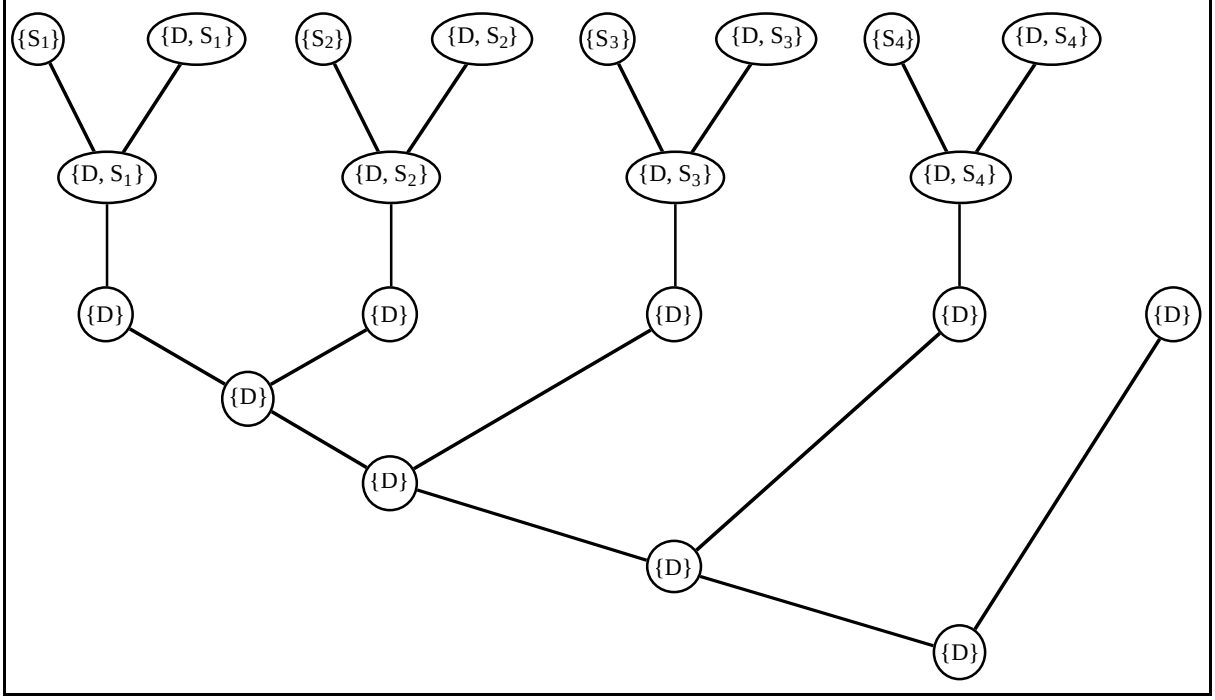
Procedure for Constructing a Binary Join Tree
INPUT: Ψ, Φ
OUTPUT: N, E
INITIALIZATION
$\Psi_u \leftarrow \Psi$ /* Ψ_u denotes the set of variables in Ψ that have not yet been deleted */
$\Phi_u \leftarrow \Phi$ /* Φ_u denotes the multiset of subsets, indexed as $s_1, ..., s_{|\Phi|}$, that have yet to been arranged in the join tree */
$N \leftarrow \emptyset$ /* N denotes the multiset of nodes of the binary join tree */

$E \leftarrow \emptyset$ /* E denotes the set of edges */
$k \leftarrow |\Phi| + 1$ /* k will be the index of the next subset created in the process of constructing a binary join tree */
DO WHILE $|\Phi_u| > 1$ /* If $|\Phi_u| = 1$, then we are done */
Pick a variable $Y \in \Psi_u$ (using some heuristic)
$\Phi_Y \leftarrow \{s_i \in \Phi_u \mid Y \in s_i\}$.
DO WHILE $|\Phi_Y| > 1$
$r_1 \leftarrow s_i$ and $r_2 \leftarrow s_j$ where $s_i, s_j \in \Phi_Y$ and $\| s_i \cup s_j \| \leq \| s_p \cup s_q \|$ for all $s_p, s_q \in \Phi_Y$
$s_k \leftarrow r_1 \cup r_2$ /* s_k is the new indexed subset created */
$N \leftarrow N \cup \{r_1, r_2, s_k\}$
$E \leftarrow E \cup \{\{r_1, s_k\}, \{r_2, s_k\}\}$
$\Phi_Y \leftarrow \Phi_Y - \{r_1, r_2\} \cup \{s_k\}$ /* Each time the number of subsets in Φ_Y is reduced by exactly 1 */
$k \leftarrow k+1$
END DO
IF $|\Psi_u| > 1$ **THEN DO**
$r \leftarrow s_i$ where $\{s_i\} = \Phi_Y$
$s_k \leftarrow r - \{Y\}$ /* s_k is the new indexed subset created */
$N \leftarrow N \cup \{r\} \cup \{s_k\}$
$E \leftarrow E \cup \{\{r, s_k\}\}$
$\Phi_u \leftarrow \Phi_u \cup \{s_k\}$
$k \leftarrow k+1$
END IF
$\Phi_u \leftarrow \Phi_u - \{s_i \in \Phi_u \mid Y \in s_i\}$
$\Psi_u \leftarrow \Psi_u - \{Y\}$
END DO
$N \leftarrow N \cup \Phi_u$ /* This is needed for the trivial case $|\Phi| = 1$ */
END

It is easy to see from the procedure that the join tree constructed will always be binary. We illustrate this procedure with an example.

Example 3. Consider a valuation network with valuations as follows: δ for {D}, σ_1 for {D, S_1}, σ_2 for {D, S_2}, σ_3 for {D, S_3}, σ_4 for {D, S_4}, o_1 for {S_1}, and o_2 for {S_2}. Suppose we need the marginal of the joint for all five variables. If we implement the binary fusion procedure for the multiset $\Phi = \{\{D\}, \{D, S_1\}, \{D, S_2\}, \{D, S_3\}, \{D, S_4\}, \{S_1\}, \{S_2\}, \{S_3\}, \{S_4\}\}$ using

Figure 9. A binary join tree resulting from the binary fusion procedure applied to the valuation network in Example 3.



deletion sequence $S_1S_2S_3S_4D$ suggested by the one-step-look-ahead heuristic, the resulting join tree is displayed in Figure 9.

8 CONCLUSIONS

The main goal of this paper is to describe a data structure called binary join trees that are useful in computing multiple marginals efficiently using the Shenoy-Shafer architecture. We define binary join trees, describe their utility, and describe a procedure for constructing them.

The join tree construction process described here is superficially different from the method described in Lauritzen and Spiegelhalter [1988] which consists of moralizing a directed acyclic graph, triangulating the moral graph using the maximum cardinality search method, and then arranging the cliques of the triangulated moral graph in a join tree. Instead of starting from a directed acyclic graph, we start with a more general setting—a hypergraph consisting of all subsets for which we have valuations and all subsets for which we desire marginals, and we use the fusion algorithm as a guide to constructing a join tree. Thus the join tree can be regarded as a data structure to organize the computations of the fusion algorithm.

Binary join trees further refine the data structure so that unnecessary computations are minimized. In particular, we identify three sources of inefficiencies associated with non-binary join

trees that are eliminated in binary join trees, and we describe an automatic procedure for constructing them based on the idea of binary fusion.

Although we have restricted our discussion in this paper to the Shenoy-Shafer architecture, binary join trees are also useful in the Lauritzen-Spiegelhalter [1988] and Hugin [Jensen *et al*. 1990, Lauritzen and Jensen 1996] architectures, albeit to a lesser extent. This is because in the Lauritzen-Spiegelhalter and Hugin architectures, some of the combination operations are replaced by the division operations. Since the main role of the binary join trees is to reduce the computation in combinations, their utility is less in these other architectures.

An interesting question is the relative computational efficiencies of the three architectures. It is commonly believed that the Hugin architecture is the most efficient (at least for the case of probabilistic reasoning). Ongoing work [Lepar and Shenoy 1996] leads us to believe that if binary join trees are used, then the Shenoy-Shafer architecture is at least as computationally efficient as—if not more than—the Hugin architecture.

ACKNOWLEDGMENTS

This work is based on research supported in part by Hughes Research Laboratories under grant No. GP3044.95-125. Any opinions, findings, and conclusion or recommendations expressed in this paper are those of the author and do not necessarily reflect the views of Hughes Research Laboratories. I am grateful to Liping Liu, Hong Xu, Steffen L. Lauritzen, Finn V. Jensen, Peter Gillett, Kurt Reiser, and Yang Chen for comments and discussions.

REFERENCES

Almond, R. G. (1995), *Graphical Belief Modeling*, Chapman & Hall, London, UK.

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

Beeri, C., R. Fagin, D. Maier and M. Yannakakis (1983), "On the desirability of acyclic database schemes," *Journal of the Association for Computing Machinery*, **30**(3), 479–513.

Bertele, U. and F. Brioschi (1972), *Nonserial Dynamic Programming*, Academic Press, New York, NY.

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

Cano, J., M. Delgado and S. Moral (1993), "An axiomatic framework for propagating uncertainty in directed acyclic networks," *International Journal of Approximate Reasoning*, **8**(4), 253–280.

Dempster, A. P. (1967), "Upper and lower probabilities induced by a multivalued mapping," *Annals of Mathematical Statistics*, **38**, 325–339.

Dempster, A. P. and A. Kong (1988), "Uncertain evidence and artificial analysis," *Journal of Statistical Planning and Inference*, **20**, 355–368.

Jensen, F. V., S. L. Lauritzen and K. G. Olesen (1990), "Bayesian updating in causal probabilistic networks by local computation," *Computational Statistics Quarterly*, **4**, 269–282.

Kjærulff, U. (1990), "Triangulation of graphs—Algorithms giving small total state space," Technical Report R90-09, Institute for Electronic Systems, Aalborg University, Denmark.

Kong, A. (1986), "Multivariate belief functions and graphical models," Ph.D. dissertation, Department of Statistics, Harvard University, Cambridge, MA.

Lauritzen, S. L. and D. J. Spiegelhalter (1988), "Local computations with probabilities on graphical structures and their application to expert systems (with discussion)," *Journal of Royal Statistical Society, Series B*, **50**(2), 157–224.

Lauritzen, S. L. and F. V. Jensen (1996), "Local computation with valuations from a commutative semigroup," Technical Report # R-96-2028, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Lepar, V. and P. P. Shenoy (1996), "A comparison of architectures for exact computation of marginals," Working Paper, School of Business, University of Kansas, Lawrence, KS.

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," Ph.D. dissertation, School of Business, University of Kansas, Lawrence, KS.

Olmsted, S. M. (1983), "On representing and solving decision problems," Ph.D. thesis, Department of Engineering-Economic Systems, Stanford University, Stanford, CA.

Pearl, J. (1986), "Fusion, propagation and structuring in belief networks," *Artificial Intelligence*, **29**, 241–288.

Rose, D. J. (1970), "Symmetric elimination on sparse positive definite systems and the potential flow network problem," Ph.D. dissertation, Harvard University.

Shafer, G. (1991), "An axiomatic study of computation in hypertrees," Working Paper No. 232, School of Business, University of Kansas, Lawrence, KS.

Shafer, G. (1996), *Probabilistic Expert Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Shafer, G., P. P. Shenoy, and K. Mellouli (1987), "Propagating belief functions in qualitative Markov trees," *International Journal of Approximate Reasoning*, **1**(4), 349–400.

Shenoy, P. P. (1989), "A valuation-based language for expert systems," *International Journal of Approximate Reasoning*, **3**(5), 383-411.

Shenoy, P. P. (1991a), "On Spohn's rule for revision of beliefs," *International Journal of Approximate Reasoning*, **5**(2), 149–181.

Shenoy, P. P. (1991b), "Valuation-based systems for discrete optimization," in P. P. Bonissone, M. Henrion, L. N. Kanal and J. F. Lemmer (eds.), *Uncertainty in Artificial Intelligence*, **6**, 385–400, North-Holland, Amsterdam.

Shenoy, P. P. (1992a), "Valuation-based systems: A framework for managing uncertainty in expert systems," in Zadeh, L. A. and J. Kacprzyk (eds.), *Fuzzy Logic for the Management of Uncertainty*, 83–104, John Wiley & Sons, New York, NY.

Shenoy, P. P. (1992b), "Using possibility theory in expert systems," *Fuzzy Sets and Systems*, **52**(2), 129–142.

Shenoy, P. P. (1994a), "Conditional independence in valuation-based systems," *International Journal of Approximate Reasoning*, **10**(3), 203–234.

Shenoy, P. P. (1994b), "Representing conditional independence relations by valuation networks," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, **2**(2), 143–165.

Shenoy, P. P. (1994c), "Consistency in valuation-based systems," *ORSA Journal on Computing*, **6**(3), 281–291.

Shenoy, P. P. and G. Shafer (1990), "Axioms for probability and belief-function propagation," in Shachter, R. D., T. S. Levitt, J. F. Lemmer and L. N. Kanal (eds.), *Uncertainty in Artificial Intelligence*, **4**, 169–198, North-Holland, Amsterdam. Reprinted in: Shafer, G. and J. Pearl (eds.), *Readings in Uncertain Reasoning*, 575–610, 1990, Morgan Kaufmann, San Mateo, CA.

Xu, H. (1991), "An efficient implementation of belief function propagation," in B. D'Ambrosio, P. Smets and P. P. Bonissone (eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Seventh Conference*, 425–432, Morgan Kaufmann, San Mateo, CA.

Xu, H. and R. Kennes (1994), "Steps toward efficient implementation of Dempster-Shafer theory," in R. R. Yager, J. Kacprzyk and M. Fedrizzi (eds.), *Advances in the Dempster-Shafer Theory of Evidence*, 153–174, John Wiley & Sons, New York.

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

SELECTED WORKING PAPERS

Unpublished working papers are available via ftp from
Host: ftp://ftp.bschool.ukans.edu
User ID: (leave blank)
Password: (leave blank)
Directory: /data/pub/pshenoy/
File: wpxxx.ps (put appropriate Working Paper # in place of xxx)

No. 184. "Propagating Belief Functions with Local Computations," Prakash P. Shenoy and Glenn Shafer, February 1986. Appeared in *IEEE Expert*, **1**(3), 1986, 43–52.

No. 190. "Propagating Belief Functions in Qualitative Markov Trees," Glenn Shafer, Prakash P. Shenoy, and Khaled Mellouli, June 1987. Appeared in *International Journal of Approximate Reasoning*, **1**(4), 1987, 349–400.

No. 197. "AUDITOR'S ASSISTANT: A Knowledge Engineering Tool for Audit Decisions," Glenn Shafer, Prakash P. Shenoy, and Rajendra Srivastava, April 1988. Appeared in *Auditing Symposium IX: Proceedings of 1988 Touche Ors/University of Kansas Symposium on Auditing Problems*, 61–84, School of Business, University of Kansas, Lawrence, KS.

No. 200. "Probability Propagation," Glenn Shafer and Prakash P. Shenoy, August 1988. Appeared in *Annals of Mathematics and Artificial Intelligence*, **2**(1–4), 1990, 327–352.

No. 203. "A Valuation-Based Language for Expert Systems," Prakash P. Shenoy, August 1988. Appeared in *International Journal of Approximate Reasoning*, **3**(5), 1989, 383-411.

No. 209. "Axioms for Probability and Belief-Function Propagation," Prakash P. Shenoy and Glenn Shafer, November 1988. Appeared in Shachter, R. D., M. Henrion, L. N. Kanal, and J. F. Lemmer (eds.), *Uncertainty in Artificial Intelligence*, **4**, 1990, 169–198. Reprinted in Shafer, G. and J. Pearl (eds.), *Readings in Uncertain Reasoning*, 1990, 575–610, Morgan Kaufmann, San Mateo, CA.

No. 211. "MacEvidence: A Visual Evidential Language for Knowledge-Based Systems," Yen-Teh Hsia and Prakash P. Shenoy, March 1989. An 8-page summary of this paper appeared as "An evidential language for expert systems," in Ras, Z. W. (ed.), *Methodologies for Intelligent Systems*, **4**, 1989, 9–16, North-Holland, Amsterdam.

No. 213. "On Spohn's Rule for Revision of Beliefs," Prakash P. Shenoy, July 1989. Appeared in *International Journal of Approximate Reasoning*, **5**(2), 1991, 149–181.

No. 216. "Consistency in Valuation-Based Systems," Prakash P. Shenoy, February 1990, revised May 1991. Appeared in *ORSA Journal on Computing*, Vol. 6, No. 3, 1994, 281–291.

No. 220. "Valuation-Based Systems for Bayesian Decision Analysis," Prakash P. Shenoy, April 1990, revised May 1991. Appeared in *Operations Research*, **40**(3), 1992, 463–484.

No. 221. "Valuation-Based Systems for Discrete Optimization," Prakash P. Shenoy, June 1990. Appeared in Bonissone, P. P., M. Henrion, L. N. Kanal, and J. F. Lemmer, eds., *Uncertainty in Artificial Intelligence*, **6**, 1991, 385–400, North-Holland, Amsterdam.

No. 223. "A New Method for Representing and Solving Bayesian Decision Problems," Prakash P. Shenoy, September 1990. Appeared in: Hand, D. J. (ed.), *Artificial Intelligence Frontiers in Statistics: AI and Statistics III*, 1993, 119–138, Chapman & Hall, London, England.

No. 226. "Valuation-Based Systems: A Framework for Managing Uncertainty in Expert Systems," Prakash P. Shenoy, March, 1991. Appeared in: Zadeh, L. A. and J. Kacprzyk (eds.), *Fuzzy Logic for the Management of Uncertainty*, 1992, 83–104, John Wiley and Sons, New York, NY.

No. 227. "Valuation Networks, Decision Trees, and Influence Diagrams: A Comparison," Prakash P. Shenoy, June 1991. Appeared as: "A Comparison of Graphical Techniques for Decision

Analysis" in *European Journal of Operational Research*, Vol. 78, No. 1, 1994, 1–21.

No. 233. "Using Possibility Theory in Expert Systems," Prakash P. Shenoy, September 1991. Appeared in *Fuzzy Sets and Systems*, **52**(2), 1992, 129–142.

No. 236. "Conditional Independence in Valuation-Based Systems," Prakash P. Shenoy, September 1991. Appeared in *International Journal of Approximate Reasoning*, **10**(3), 1994, 203–234.

No. 238. "Valuation Networks and Conditional Independence," Prakash P. Shenoy, September 1992. Appeared as "Representing Conditional Independence Relations by Valuation Networks" in *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, **2**(2), 1994, 143–165.

No. 239. "Game Trees for Decision Analysis," Prakash P. Shenoy, February 1993. Revised February 1994. An 8-page summary titled "Information Sets in Decision Theory" appeared in Clarke, M., R. Kruse and S. Moral (eds.), *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, Lecture Notes in Computer Science No. 747, 1993, 318–325, Springer-Verlag, Berlin.

No. 242. "A Theory of Coarse Utility," Liping Liu and Prakash P. Shenoy, February 1993. Revised September 1993. Appeared in *Journal of Risk and Uncertainty*, Vol. 11, 1995, pp. 17–49.

No. 245. "Modeling Ignorance in Uncertainty Theories," Prakash P. Shenoy, April 1993. Appeared in Gammerman, A. (ed.), *Probabilistic Reasoning and Bayesian Belief Networks*, 1995, 71–96, Alfred Waller, Henley-on-Thames, UK.

No. 246. "Valuation Network Representation and Solution of Asymmetric Decision Problems," Prakash P. Shenoy, April 1993. Revised September 1995. A 10-page summary of this paper appeared as "Representing and Solving Asymmetric Decision problems Using Valuation Networks" in Fisher, D. and H.-J. Lenz (eds.), *Artificial Intelligence and Statistics V*, Lecture Notes in Statistics, **112**, 99–108, Springer-Verlag, New York, 1996.

No. 247. "Inducing Attitude Formation Models Using TETRAD," Sanjay Mishra and Prakash P. Shenoy, May 1993. Revised October 1993. Appeared as "Attitude Formation Models: Insights from TETRAD" in Cheeseman, P. and R. W. Oldford (eds.), *Selecting Models from Data: Artificial Intelligence and Statistics IV*, Lecture Notes in Statistics No. 89, 1994, 223–232, Springer-Verlag, Berlin.

No. 258. "A Note on Kirkwood's Algebraic Method for Decision Problems," Rui Guo and Prakash P. Shenoy, November 1993. Revised May 1994. To appear in *European Journal of Operational Research*, 1996.

No. 261. "A New Pruning Method for Solving Decision Trees and Game Trees," Prakash P. Shenoy, March 1994. Appeared in: Besnard, P. And S. Hanks (eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, 1995, 482–490, Morgan Kaufmann, San Francisco, CA.

No. 267. "Computing Marginals Using Local Computation," Steffen L. Lauritzen and Prakash P. Shenoy, July 1995, revised May 1996.

No. 270. "Binary Join Trees for Computing Marginals in the Shenoy-Shafer Architecture," Prakash P. Shenoy, December 1995. An 8-pp summary titled "Binary Join Trees" appeared in: Horvitz, E. and F. V. Jensen (eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, 1996, 492–499, Morgan Kaufmann, San Francisco, CA.

No. 271. "A Comparison of Graphical Techniques for Asymmetric Decision Problems," Concha Bielza and Prakash P. Shenoy, February 1996, revised June 1996.

No. 273. "A Forward Monte Carlo Method for Solving Influence Diagrams Using Local Computation," John M. Charnes and Prakash P. Shenoy, February 1996.