# Morphing the Hugin and Shenoy–Shafer Architectures

James D. Park and Adnan Darwiche

UCLA, Los Angeles CA 90095, USA,
{jd,darwiche}@cs.ucla.edu

**Abstract.** The Hugin and Shenoy–Shafer architectures are two variations on the jointree algorithm, which exhibit different tradeoffs with respect to efficiency and query answering power. The Hugin architecture is more time–efficient on arbitrary jointrees, avoiding some redundant computations performed by the Shenoy–Shafer architecture. This efficiency, however, comes at the price of limiting the number of queries the Hugin architecture is capable of answering. In this paper, we present a simple algorithm which retains the efficiency of the Hugin architecture and enjoys the query answering power of the Shenoy–Shafer architecture.

## 1 Introduction

There are a number of algorithms for answering queries with respect to Bayesian networks. Among the most popular of these are the algorithms based on jointrees, of which the Shenoy–Shafer [10] and Hugin [4, 3] architectures represent two prominent variations. While superficially similar, these architectures differ in both efficiency and query answering power. Specifically, the Hugin architecture is faster on arbitrary jointrees, but the Shenoy–Shafer architecture results in more information and can answer more queries. This paper presents an architecture that combines the best of both algorithms. In particular, we show that a simple modification to the Hugin architecture which does not alter its time and space efficiency, allows it to attain the same query answering power exhibited by the Shenoy–Shafer.

This paper is structured as follows. Section 2 reviews the definition of jointrees, and details the Shenoy–Shafer and Hugin architectures. Section 3 introduces the main idea of the paper, and provides a corresponding algorithm which can be thought of as either a more efficient Shenoy–Shafer architecture or a more expressive Hugin architecture. It also details the semantics of messages and data maintained by the new algorithm. Section 4 closes the paper with some concluding remarks. Proofs of all theorems are delegated to the appendix.

## 2 Jointree Algorithms

We review the basics of jointrees and jointree algorithms in this section. Let $\mathcal{B}$ be a belief network. A jointree for $\mathcal{B}$ is a pair $(\mathcal{T}, \mathbf{C})$, where $\mathcal{T}$ is a tree and $\mathbf{C}$ is a function that assigns a label $\mathbf{C}_i$ to each node $i$ in tree $\mathcal{T}$. A jointree must satisfy three properties: (1) each label $\mathbf{C}_i$ is a set of variables in the belief network; (2) each network variable $X$ and its parents $\mathbf{U}$ (a family) must appear together in some label $\mathbf{C}_i$; (3) if a variable appears in the labels of nodes $i$ and $j$ in the jointree, it must also appear in the label of each node $k$ on the path connecting them. The label of edge $ij$ in tree $\mathcal{T}$ is defined as $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$. The nodes of a jointree and their labels are called *clusters.* Moreover, the edges of a jointree and their labels are called *separators.* The *width* of a jointree is defined as the number of variables in its largest cluster minus 1.

Jointree algorithms start by constructing a jointree for a given belief network [10, 4, 3]. They associate *tables* (also called *potentials*) with clusters and separators.[1] The *conditional probability table* (CPT) of each variable $X$ with parents $\mathbf{U}$, denoted $\theta_{X|\mathbf{U}}$, is assigned to a cluster that contains $X$ and $\mathbf{U}$. In addition, a table over each variable $X$, denoted $\lambda_X$ and called an *evidence table,* is assigned to a cluster that contains $X$. Evidence $\mathbf{e}$ is entered into a jointree by initializing evidence tables as follows: we set $\lambda_X(x)$ to 1 if $x$ is consistent with evidence $\mathbf{e}$, and we set $\lambda_X(x)$ to 0 otherwise.

Given some evidence $\mathbf{e}$, a jointree algorithm propagates messages between clusters. After passing two message per edge in the jointree, one can compute the marginals $\Pr(\mathbf{C}, \mathbf{e})$ for every cluster $\mathbf{C}$. There are two main methods for propagating messages in a jointree, known as the Shenoy–Shafer [10] and the Hugin [4] architectures, which we review next (see [5, 6] for a thorough introduction to each architecture).

### 2.1 The Shenoy–Shafer Architecture

Shenoy–Shafer propagation proceeds as follows. First, evidence $\mathbf{e}$ is entered into the jointree through evidence indicators. A cluster is then selected as the root and message propagation proceeds in two phases, inward and outward. In the *inward phase,* messages are passed toward the root. In the *outward phase,* messages are passed away from the root. Cluster $i$ sends a message to cluster $j$ only when it has received messages from all its other neighbors $k$. A message from cluster $i$ to cluster $j$ is a table $M_{ij}$ defined as follows:

$$M_{ij} = \sum_{\mathbf{C}_i \setminus \mathbf{S}_{ij}} \Phi_i \prod_{k \neq j} M_{ki}, \tag{1}$$

where $\Phi_i$ is the product of CPTs and evidence tables assigned to cluster $i$.

Once message propagation is finished in the Shenoy–Shafer architecture, we have the following for each cluster $i$ in the jointree:

$$\Pr(\mathbf{C}_i, \mathbf{e}) = \Phi_i \prod_k M_{ki}. \tag{2}$$

[1] A table is an array which is indexed by variable instantiations. Specifically, a table $\phi$ over variables $\mathbf{X}$ is indexed by the instantiations $\mathbf{x}$ of $\mathbf{X}$. Its entries $\phi(\mathbf{x})$ are in $[0, 1]$. We assume familiarity with table operations, such as multiplication, division and marginalization [3].

Let us now look at the time and space requirements of the Shenoy–Shafer architecture. The space requirements are simply those needed to store the messages computed by Equation 1. That is, we need two tables for each separator $\mathbf{S}_{ij}$, one table stores the message from cluster $i$ to cluster $j$, and the other stores the message from $j$ to $i$. We will assume in our time analysis below the availability of the table $\Phi_i$, which represents the product of all CPT and evidence tables assigned to cluster $i$. This is meant to simplify our time analysis, but we stress that one of the attractive aspects of the Shenoy–Shafer architecture is that one can afford to keep this table in factored form, therefore, avoiding the need to allocate space for this table which may be significant.

As for time requirements, suppose that we have a jointree with $n$ clusters and width $w$. Suppose further that the table $\Phi_i$ is already available for each cluster $i$, and let us bound the amount of work performed by the inward and outward passes of the Shenoy–Shafer architecture, i.e., the work needed to evaluate Equations 1 and 2. We first note that for each cluster $i$, Equation 1 has to be evaluated $n_i$ times and Equation 2 has to be evaluated once, where $n_i$ is the number of neighbors for cluster $i$. Each evaluation of Equation 1 leads to multiplying $n_i$ tables, whose variables are all in cluster $\mathbf{C}_i$. Moreover, each evaluation of Equation 2 leads to multiplying $n_i + 1$ tables, whose variables are also all in cluster $\mathbf{C}_i$. The total complexity is then:

$$\sum_i O(n_i(n_i-1)\exp(|\mathbf{C}_i|) + n_i \exp(|\mathbf{C}_i|)),$$

(since multiplying $n$ elements requires $n-1$ multiplications) which reduces to $\sum_i O(n_i^2 \exp(w))$ where $w$ is the jointree width. This further reduces to $O(\alpha \exp(w))$, where $\alpha = \sum_i n_i^2$ is a term that ranges from $O(n)$ to $O(n^2)$ depending on the jointree struture. For example, we may have what is known as a *binary jointree* in which each cluster has at most three neighbors, leading to $\alpha \leq 6(n-1)$. Or we may have a jointree with one cluster having the other $n-1$ clusters as its neighbors, leading to $\alpha = n^2 - n$.

Given a Bayesian network with $n$ variables, and an elimination order of width $w$, we can construct a binary jointree for the network with the following properties: the jointree has width $\leq w$, and no more than $2n-1$ clusters. Hence, we can avoid the quadratic complexity suggested above by a careful construction of the jointree, although this can dramatically increase the space requirements.

## 2.2 The Hugin Architecture

We now discuss the Hugin architecture, which tends to take less time but uses more space. We first consider Figure 1 which provides an abstraction of the difference between the Hugin and Shenoy–Shafer architectures. Here, node $r$ has neighbors $1, \ldots, n$, where each edge between $r$ and its neighbor $i$ is labeled with a number $x_i$. Node $r$ is also labeled with a number $x_r$. Suppose now that node $r$ wants to send a message $m_i$ to each of its neighbors $i$, where the content of this message is: $m_i = x_r \prod_{j \neq i} x_j$. One way to do this is to compute the above product

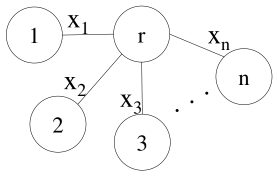


**Fig. 1.** A simplified jointree message example

for each neighbor $i$. This is the approach taken by the Shenoy–Shafer architecture and leads to a quadratic complexity in the number of neighbors $n$. Alternatively, we can compute the product $p = x_r \prod_{j=1}^{n} x_j$ only once, and then use it to compute the message to each neighbor $i$ as $m_i = p/x_i$. This is the approach taken by the Hugin architecture. It is clearly more efficient as it only requires one division for each message, while the first method requires $n$ multiplications per message. However, it requires that $x_i \neq 0$, otherwise, $p/x_i$ is not defined. But if the message is going to be later multiplied by an expression of the form $x_i\alpha$, then we can define $p/0$ to be 0, or any other number for that matter, and our computations will be correct since $(x_r \prod_{j \neq i} x_j)x_i\alpha = 0$ regardless of the message value. This is exactly what happens in the Hugin architecture when computing joint marginals and, hence, the division by zero does not pose a problem. Yet, for some other queries which we discuss later, the quantity $\prod_{i=1}^{n} x_i$ is needed when $x_r = 0$, in which case it cannot be recovered by dividing $p$ by $x_r$. Moreover, as we see next, since the Hugin architecture does not save the numbers $x_i$, it cannot compute the product $\prod_{i=1}^{n} x_i$ through an explicit multiplication of the terms appearing in this product. This is basically the main difference between the Hugin and Shenoy–Shafer architectures except that the above analysis is applied to tables instead of numbers.

Hugin propagation proceeds similarly to Shenoy–Shafer by entering evidence $\mathbf{e}$ using evidence tables; selecting a cluster as root; and propagating messages in two phases, inward and outward. The Hugin method, however, differs in some major ways. First, it maintains a table $\Phi_{ij}$ with each separator, whose entries are initialized to 1s. It also maintains a table $\Phi_i$ with each cluster $i$, initialized to the product of all CPTs and evidence tables assigned to cluster $i$; see Figure 2.

Cluster $i$ passes a message to neighboring cluster $j$ only when $i$ has received messages from all its other neighbors $k$. When cluster $i$ is ready to send a message to cluster $j$, it does the following:

- it saves the separator table $\Phi_{ij}$ into $\Phi_{ij}^{old}$
- it computes a new separator table $\Phi_{ij} = \sum_{\mathbf{C}_i \setminus \mathbf{S}_{ij}} \Phi_i$
- it computes a message to cluster $j$: $M_{ij} = \Phi_{ij}/\Phi_{ij}^{old}$
- it multiplies the computed message into the table of cluster $j$: $\Phi_j = \Phi_j M_{ij}$

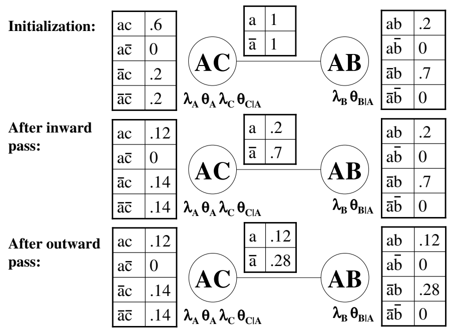


**Fig. 2.** Hugin propagation on a jointree under evidence $b$. The jointree is for network $A \rightarrow B$, where $\theta_a = .6$, $\theta_{b|a} = .2$, $\theta_{b|\bar{a}} = .7$, $\theta_{c|a} = 1$, and $\theta_{c|\bar{a}} = .5$.

After the inward and outward–passes of Hugin propagation are completed, we have the following for each cluster $i$ in the jointree: $\Pr(\mathbf{C}_i, \mathbf{e}) = \Phi_i$. The space requirements for the Hugin architecture are those needed to store cluster and separator tables: one table for each cluster and one table for each separator. Note that the Hugin architecture *does not save* the message $M_{ij} = \Phi_{ij}/\Phi_{ij}^{old}$ sent from cluster $i$ to cluster $j$.

As for time requirements, suppose that we have a jointree with $n$ clusters and width $w$. Suppose further that the initial tables $\Phi_i$ and $\Phi_{ij}$ are already available for each cluster $i$ and separator $ij$. Let us now bound the amount of work performed by the inward and outward passes of the Hugin architecture, i.e., the work needed to pass a message from each cluster $i$ to each of its neighbors $j$. Saving the old separator table takes $O(\exp(|\mathbf{S}_{ij}|))$; computing the message takes $O(\exp(|\mathbf{C}_i|) + \exp(|\mathbf{S}_{ij}|))$, and multiplying the message into the table of cluster $j$ takes $O(\exp(|\mathbf{C}_j|))$. Hence, if each cluster $i$ has $n_i$ neighbors, the total complexity is:

$$\sum_i \sum_j O(\exp(|\mathbf{C}_i|) + 2\exp(|\mathbf{S}_{ij}|) + \exp(|\mathbf{C}_j|)),$$

which reduces to $O(n \exp(w))$, where $w$ is the jointree width. Note that this result holds regardless of the jointree structure. Hence, the linear complexity in $n$ is obtained for any jointree, without a need to use a special jointree as in the Shenoy–Shafer architecture.

## 2.3 Beyond Joint Marginals

The Hugin architecture gains efficiency over the Shenoy–Shafer architecture on arbitrary jointrees by employing division. Moreover, although the use of division does not prevent the architecture from producing joint marginals, it does prevent it from producing answers to some other queries which are useful for a variety of applications including sensitivity analysis [1], local optimization problems like parameter learning [9], and MAP approximation [7].

To explain these additional queries, suppose that we just finished jointree propagation using evidence $\mathbf{e}$. This gives us the probability of evidence $\mathbf{e}$, since for any cluster $\mathbf{C}$, we have $\Pr(\mathbf{e}) = \sum_{\mathbf{c}} \Pr(\mathbf{c}, \mathbf{e})$. Suppose now that we need the probability of some new evidence which results from erasing the value of variable $X$ from $\mathbf{e}$, denoted $\mathbf{e} - X$. More generally, suppose that we need the probability of evidence $\mathbf{e} - X, x$, where $x$ is a value of variable $X$ which is different from the one appearing in evidence $\mathbf{e}$. Both of these probabilities can be obtained locally using the Shenoy–Shafer architecture without further propagation, but cannot in general be computed locally using the Hugin architecture (see [2] for some special cases). The other type of query which falls in this category is that of computing the derivative $\partial \Pr(\mathbf{e}) / \partial \theta_{x|\mathbf{u}}$ of the likelihood $\Pr(\mathbf{e})$ with respect to a network parameter $\theta_{x|\mathbf{u}} = 0$. We will now show how these queries can be answered locally using the Shenoy–Shafer architecture. We later show how they can be computed using the modification we suggest to the Hugin architecture.

To compute the probabilities $\Pr(\mathbf{e} - X, x)$ for a variable $X$ using the Shenoy–Shafer architecture, we first need to identify the cluster $i$ which contains the evidence table $\lambda_X$. The probabilities $\Pr(\mathbf{e} - X, x)$, for each value $x$, are then available in the following table which is defined over variable $X$:

$$\sum_{\mathbf{C}_i \setminus X} \prod_k \phi_k \prod_j M_{ji}.$$

Here, $\phi_k$ ranges over all CPTs and evidence tables assigned to cluster $i$, excluding the evidence table $\lambda_X$ [2, 8].

Similarly, to compute the derivatives $\partial \Pr(\mathbf{e}) / \partial \theta_{x|\mathbf{u}}$, we need to identify the cluster $i$ which is assigned the CPT of variable $X$. The derivatives for all instantiations $x\mathbf{u}$ of variable $X$ and its parents $\mathbf{U}$ are then available in the following table, which is defined over family $X\mathbf{U}$:

$$\sum_{\mathbf{C}_i \setminus X \cup \mathbf{U}} \prod_k \phi_k \prod_j M_{ji}.$$

Here, $\phi_k$ ranges over all CPTs and evidence tables assigned to cluster $i$, excluding the CPT $\theta_{X|\mathbf{U}}$ [8].

Hugin is not able to handle these queries in general because it does not save messages that are exchanged between clusters, and because table division may lead to a division by zero (see [2] for a special case where Hugin can handle some of these queries).

## 3 Getting the Best of Both Worlds

We now present a jointree propagation algorithm that combines the query answering power of Shenoy–Shafer propagation with the efficiency of Hugin propagation. The messages sent between clusters are the same as Shenoy–Shafer messages, but the tables stored at each cluster represent the product of assigned tables and incoming messages in a manner similar to the Hugin approach.

As discussed earlier, division is a key to Hugin efficiency, but it also produces a loss of information. The problem is that multiplication by zero is noninvertible. In Section 3.1 we discuss the problem in detail and introduce a simple and efficient technique to circumvent it. In Section 3.2 we describe the new propagation algorithm. Section 3.3 details the semantics of messages in the new architecture, as well as the content of cluster and separator tables.

## 3.1 Handling Zeros

This section introduces the notion of a *zero conscious number:* a pair $(z, b)$, where $z$ is a scalar and $b$ is a bit. It also defines various operations on these numbers. We then show in the following section that by employing such numbers in a variation on the Hugin architecture, we can attain the same query answering power exhibited by the Shenoy–Shafer architecture.

To motivate zero conscious numbers, consider a set of numbers $x_1, \ldots, x_n$ and suppose that for each $i = 1, \ldots, n$, our goal is to compute the product $m_i = \prod_{j \neq i} x_j$. We distinguish between three cases:

**Case 1:** $x_1, \ldots, x_n$ contain no zeros. Then $m_i = p/x_i$, where $p = \prod_j x_j$.
**Case 2:** $x_1, \ldots, x_n$ contain a single zero $x_k$. Then $m_k = \prod_{j \neq k} x_j$ and $m_i = 0$ for all $i \neq k$.
**Case 3:** $x_1, \ldots, x_n$ contain more than one zero. Then $m_i = 0$ for all $i$.

Note that in Case 3, we have $m_i = 0 = \prod_{j \neq k} x_j$ for any $k$ since we have more than one zero. Hence, Case 2 and Case 3 can be merged together. Using these cases, the messages $m_i$ can be computed efficiently by first computing a pair $(z, b)$ such that:

- $b$ is a bit which indicates whether any of the elements $x_i$ is a zero (Cases 2,3).
- $z$ is the product of all elements $x_i$, excluding the single zero if one exists.

For example, if the elements $x_i$ are $1, 2, 3, 4, 5$, we would compute $(1 * 2 * 3 * 4 * 5, f) = (120, f)$ since no zero was withheld. For elements $1, 2, 0, 4, 5$, we would compute $(40, t)$. Finally, for elements $1, 2, 0, 4, 0$, we would compute $(0, t)$.

Then each message $m_i$ can be computed from the pair $(z, b)$ as follows:

$$m_i = \begin{cases} z/x_i & \text{if } b = f \\ 0 & \text{if } b = t \text{ and } x_i \neq 0 \\ z & \text{if } b = t \text{ and } x_i = 0 \end{cases}$$

This can be thought of as dividing the pair $(z, b)$ by $x_i$. In fact, we will call the pair $(z, b)$ a *zero conscious number* and define division by a scalar as given above. Two more operations on zero conscious numbers will be needed. In particular, multiplying a zero conscious number $(z, b)$ by a scalar $c$ is defined as:

$$(z, b) * c = \begin{cases} (z, t) & \text{if } b = f \text{ and } c = 0 \\ (c * z, b) & \text{otherwise.} \end{cases}$$

Moreover, the addition of two zero conscious numbers is defined as:

$$(z_1, b_1) + (z_2, b_2) = \begin{cases} (z_1, b_1) & \text{if } b_1 = f \text{ and } b_2 = t \\ (z_2, b_2) & \text{if } b_1 = t \text{ and } b_2 = f \\ (z_1 + z_2, b_1) & \text{otherwise.} \end{cases}$$

Finally, we define

$$real(z, b) = \begin{cases} z \text{ if } b = f \\ 0 \text{ otherwise.} \end{cases}$$

Note that if the zero conscious number $(z, b)$ was computed for elements $x_1, \ldots, x_n$, then $real(z, b)$ recovers the product $\prod_j x_j$ of these elements.

We can also defined *zero conscious tables* which map variable instantiations to zero conscious numbers. Now let $\Psi$ be a zero conscious table and $\Phi$ be a standard table. The marginal $\sum_{\mathbf{X}} \Psi$, product $\Psi\Phi$, and division $\Psi/\Phi$ can then be defined in the obvious way. Moreover, $real(\Psi)$ is defined as a standard table which results from applying the *real* operation to each entry of $\Psi$.

## 3.2 Algorithmic Description

The algorithm we propose is very similar to Hugin propagation. Like the Hugin architecture, it maintains a table $\Phi_{ij}$ for each separator $ij$. A table is also maintained for each cluster. Unlike for Hugin, the table $\Psi_i$ associated with cluster $i$ is a zero conscious table. From here on, we will use $\Psi$ to denote a zero conscious table and $\Phi$ to denote a standard table.

**Initialization** The separator table entries are initialized to 1, and the cluster table entries are initialized to $(1, f)$. The CPTs and evidence tables assigned to a cluster are multiplied into the corresponding cluster table.

**Message Propagation** This algorithm requires the same obedience to message ordering that the other jointree algorithms do. That is, messages are sent from the leaves, toward some root, then back from the root to the leaves. A message from cluster $i$ to cluster $j$ is computed as follows:

- $\Psi_{temp} \leftarrow \sum_{\mathbf{C}_i \setminus \mathbf{S}_{ij}} \Psi_i$
- $\Psi_j \leftarrow \Psi_j(\Psi_{temp}/\Phi_{ij})$
- $\Phi_{ij} \leftarrow real(\Psi_{temp})$

Figure 3 illustrates an example of this propagation scheme. This algorithm basically mirrors Hugin propagation, but with zero conscious tables for clusters. It has some minor time and space overhead over what the Hugin algorithm requires. The time overhead consists of an additional logical test per operation. The storage requirement is also fairly insignificant. Single precision floating point numbers require 32 bits and double precision numbers require 64 bits, so an extra bit (or even byte, if the processor can't efficiently manipulate bits) per number will increase the space requirements only slightly.

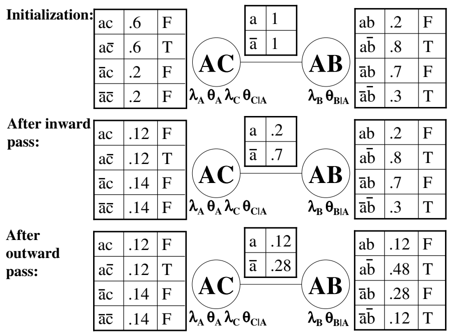


**Fig. 3.** Zero conscious propagation illustrated on a simple jointree under evidence $b$, where the left cluster is root. The jointree is for network $C \leftarrow A \rightarrow B$, where $\theta_a = .6$, $\theta_{b|a} = .2$, $\theta_{b|\bar{a}} = .7$, $\theta_{c|a} = 1$, and $\theta_{c|\bar{a}} = .5$.

## 3.3 The Semantics

The message passing semantics is the same as for Shenoy–Shafer.

**Theorem 1.** *The message passed from cluster $i$ to cluster $j$ is the same as the message passed using Shenoy–Shafer propagation. That is, if $\Phi_{ij}$ is the product of all tables assigned to clusters on the $i$–side of edge $ij$, and if $\mathbf{X}$ are the variables appearing in these tables, then the message $M_{ij} = \sum_{\mathbf{X}\setminus\mathbf{S}_{ij}} \Phi_{ij}$.*

The cluster and separator table semantics closely resemble the corresponding Hugin semantics.

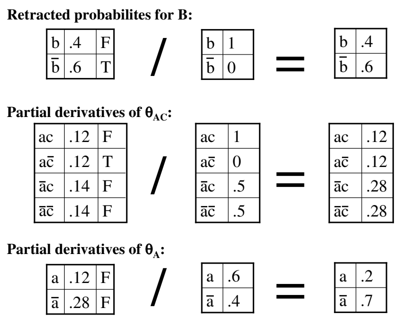
**Theorem 2.** *After all the messages have been passed, real$(\Psi_i) = \Pr(\mathbf{C}_i, \mathbf{e})$ and $\Phi_{ij} = \Pr(\mathbf{S}_{ij}, \mathbf{e})$ for all neighboring clusters $i$ and $j$.*

Although very similar to the Hugin semantics, the difference in the cluster table makes these semantics significantly more powerful.

**Theorem 3.** *Let $\phi_{i1} ... \phi_{in}$ be the CPTs and evidence tables assigned to cluster $i$, and let $\mathbf{X}_m$ be the set of variables of $\phi_{im}$. Then after message passing is complete,*

$$\sum_{\mathbf{C}_i\setminus\mathbf{X}_k} \prod_{m\neq k} \phi_{im} \prod_j M_{ji} = \left( \sum_{\mathbf{C}_i\setminus\mathbf{X}_k} \Psi_i \right) / \phi_{ik}.$$

This theorem shows that we can use zero conscious division to perform the same local computations permitted by the Shenoy–Shafer architecture. Consider Figure 3 for an example, which depicts an example of zero conscious propagation under evidence $\mathbf{e} = b$. Given the table associated with the left cluster, we have $\Pr(\mathbf{e}) = .12+.14+.14 = .4$. Suppose now that we want to compute the probability of $(\mathbf{e} - B, \bar{b}) = \bar{b}$. We can do this by identifying the cluster $AB$ which contains



**Retracted probabilites for B:**

| | | |
|---|---|---|
| b | .4 | F |
| $\bar{b}$ | .6 | T |

/

| | |
|---|---|
| b | 1 |
| $\bar{b}$ | 0 |

=

| | |
|---|---|
| b | .4 |
| $\bar{b}$ | .6 |

**Partial derivatives of $\theta_{AC}$:**

| | | |
|---|---|---|
| ac | .12 | F |
| a$\bar{c}$ | .12 | T |
| $\bar{a}$c | .14 | F |
| $\bar{a}\bar{c}$ | .14 | F |

/

| | |
|---|---|
| ac | 1 |
| a$\bar{c}$ | 0 |
| $\bar{a}$c | .5 |
| $\bar{a}\bar{c}$ | .5 |

=

| | |
|---|---|
| ac | .12 |
| a$\bar{c}$ | .12 |
| $\bar{a}$c | .28 |
| $\bar{a}\bar{c}$ | .28 |

**Partial derivatives of $\theta_A$:**

| | | |
|---|---|---|
| a | .12 | F |
| $\bar{a}$ | .28 | F |

/

| | |
|---|---|
| a | .6 |
| $\bar{a}$ | .4 |

=

| | |
|---|---|
| a | .2 |
| $\bar{a}$ | .7 |

**Fig. 4.** Evidence retraction and partial derivative operations for the jointree in Figure 3.

the evidence table $\lambda_B$ and then computing: $(\sum_A \Psi_{AB})/\lambda_B$. This leads to the first division shown in Figure 4, showing that the probability of $\bar{b} = .6$.

Similarly, to compute the partial derivatives of $\Pr(\mathbf{e})$ with respect to parameters $\theta_{C|A}$, we need $\Psi_{AC}/\theta_{C|A}$ which is also shown in Figure 4. According to this computation, for example, we have $\partial\Pr(\mathbf{e})/\partial\theta_{c|\bar{a}} = .28$. Finally, the partial derivatives $\partial\Pr(\mathbf{e})/\partial\theta_a$ are obtained from $(\sum_C \Psi_{AC})/\theta_A$, which is also shown in Figure 4. According to this computation, $\partial\Pr(\mathbf{e})/\partial\theta_{\bar{a}} = .7$.

## 4 Conclusion

We proposed a combination of the Shenoy–Shafer and Hugin architectures, in which we use zero conscious tables/potentials. The use of these tables provide a simple way to exploit the efficiency of the Hugin method, while extending the set of queries that can be answered efficiently. For the price of a single bit per cluster entry, and some minimal logic operations, all queries answerable using Shenoy–Shafer propagation can now be answered using Hugin type operations. For applications that require more than just marginal probabilities, such as local search methods for MAP and sensitivity analysis, this can produce a significant speed up over the use of Shenoy–Shafer architecture.

## References

1. H. Chan and A. Darwiche. When do numbers really matter? In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 65–74, San Francisco, California, 2001. Morgan Kaufmann Publishers, Inc.
2. R. Cowell, A. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.
3. C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

4. F. V. Jensen, S. Lauritzen, and K. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.
5. S. L. Lauritzen and F. V. Jensen. Local computation with valuations from a commutative semigroup. *Annals of Mathematics and Artificial Intelligence*, 21(1):51–69, 1997.
6. V. Lepar and P. P. Shenoy. A comparison of Lauritzen-Spiegelhalter, Hugin and Shenoy-Shafer architectures for computing marginals of probability distributions. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 328–337, 1998.
7. J. Park and A. Darwiche. Approximating map using local search. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 403–410, San Francisco, California, 2001. Morgan Kaufmann Publishers, Inc.
8. J. Park and A. Darwiche. A differential semantics for jointree algorithms. In *Neural Information Processing Systems (NIPS) 15*, 2003.
9. S. Russell, J. Binder, D. Koller, and K. Kanazawa. Local learning in probabilistic networks with hidden variables. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 1146–1152, 1995.
10. P. P. Shenoy and G. Shafer. Propagating belief functions with local computations. *IEEE Expert*, 1(3):43–52, 1986.

## A Proof of Theorems

We first introduce a few lemmas about zero conscious potentials that we will need to prove the theorems.

**Lemma 1.** *Let $\Psi$ be the zero conscious product of potentials $\Phi_1, ..., \Phi_n$. Then $\Psi/\Phi_i = \prod_{j \neq i} \Phi_j$.*

*Proof.* Consider an entry $\psi$ of $\Psi$, and the unique compatible entry $\phi_i$ in from each table. Then, based on the division property of zero conscious numbers, $\psi/\phi_i = \prod_{j \neq i} \phi_j$. This is true for all entries $\psi$, thus proving the result.

**Lemma 2.** *Let $c$ be a factor of zero conscious numbers $a = (n_a, z_a)$ and $b = (n_b, z_b)$. Then $(a + b)/c = a/c + b/c$*

*Proof.* We simply break it into cases and show that the definitions of the operations require that they agree.

**Case 1** $(c = 0)$ Then $z_a = z_b = t$. Thus $(a + b)/c = (n_a + n_b, t)/c = n_a + n_b = a/c + b/c$ since $a/c = n_a$ and $b/c = n_b$.
**Case 2** $(c \neq 0, z_a = f, z_b = f)$ Then $(a + b)/c = (n_a + n_b, f)/c = (n_a + n_b)/c = n_a/c + n_b/c = a/c + b/c$.
**Case 3** $(c \neq 0, z_a = f, z_b = t)$ Then $(a + b)/c = a/c = a/c + b/c$ since $b/c = 0$.
**Case 4** $(c \neq 0, z_a = t, z_b = f)$ Then $(a + b)/c = b/c = a/c + b/c$ since $a/c = 0$.
**Case 5** $(c \neq 0, z_a = t, z_b = t)$ Then $(a + b)/c = 0 = a/c + b/c$.

**Lemma 3.** *Let $\Psi$ be the zero conscious product of potentials $\Phi_1, ..., \Phi_n$. Then $(\sum_{\mathbf{C} \setminus \mathbf{S}} \Psi)/\Phi_i = \sum_{\mathbf{C} \setminus \mathbf{S}} (\Psi/\Phi_i)$ where $\mathbf{C}$ are the variables of $\Psi$, and $\mathbf{S}$ are the variables of $\Phi_i$.*

*Proof.* Consider the instances $\mathbf{c}$ of $\mathbf{C}$ compatible with instance $\mathbf{s}$ of $\mathbf{S}$. Repeated application of Lemma 2 implies that summing them, then dividing is the same as dividing them summing. This is true for each element, and so for the table as a whole.

**Proof of Theorem 1**

We will first prove the theorem for messages sent towards the root. We will then prove it for messages away from the root.

Consider a leaf node $i$ sending a message to its neighbor $j$. The cluster potential $\Psi_i$ consists of the product of all tables $\phi_{i1} ... \phi_{in}$ assigned to it. The message sent is $(\sum_{\mathbf{C} \setminus \mathbf{S}} \Psi_i)/\mathbf{1} = \sum_{\mathbf{C} \setminus \mathbf{S}} \prod_k \phi_{ik} = \sum_{\mathbf{C} \setminus \mathbf{S}} \Phi_i$ which is the Shenoy–Shafer message from a leaf node.

Now, assume by way of induction that all messages toward the root have been received for cluster $i$. The cluster $\Psi_i$ contains the product of the assigned tables $\phi_{i1}, ..., \phi_{in}$ and the incoming upward messages $M_{ki}$ from neighbors $k$. Then the upward message is $(\sum_{\mathbf{C} \setminus \mathbf{S}} \Psi_i)/\mathbf{1} = \sum_{\mathbf{C} \setminus \mathbf{S}} \prod_m \phi_{im} \prod_k M_{ki} = \sum_{\mathbf{C} \setminus \mathbf{S}} \Phi_i \prod_k M_{ki}$ which again is the Shenoy–Shafer message.

So, for the upward pass the messages sent equal the corresponding Shenoy–Shafer messages.

Now, consider a message sent from the root $r$. The cluster potential $\Psi_r$ consists of the product of the assigned tables, and the incoming messages of all neighbors. The message sent to neighbor $j$ is $(\sum_{\mathbf{C} \setminus \mathbf{S}} \Psi_r)/M_{jr}$ which by application of Lemma 3 followed by Lemma 1 yields $\sum_{\mathbf{C} \setminus \mathbf{S}} \Phi_r \prod_{i \neq j} M_{ir}$ which is again the appropriate Shenoy–Shafer message.

Now, assume by way of induction that a cluster $i$ has received messages which equal the corresponding Shenoy–Shafer messages from each of its neighbors. Then, the message sent to neighbor $j$ away from the root is $(\sum_{\mathbf{C} \setminus \mathbf{S}} \Psi_i)/M_{ji}$ which again appealing to Lemmas 3 and 1 equals $\sum_{\mathbf{C} \setminus \mathbf{S}} \Phi_i \prod_{k \neq j} M_{ki}$ which is the same as the Shenoy–Shafer message.

**Proof of Theorem 2**

After propagation completes, $\Psi_i$ contains the product of the locally assigned tables $\phi_{i1} ... \phi_{in}$, and the incoming messages. Thus $real(\Psi_i) = \prod_k \phi_{ik} \prod_j M_{ji} = \Phi_i \prod_j M_{ji}$. Since the messages are the same as the Shenoy–Shafer messages, and in the Shenoy–Shafer architecture $\Pr(\mathbf{C}_i, \mathbf{e}) = \Phi_i \prod_j M_{ji}$, $real(\Psi_i) = \Pr(\mathbf{C}_i, \mathbf{e})$. For separator $ij$, where $i$ is closer to the root, after propagation completes, $\Phi_{ij} = real(\sum_{\mathbf{C}_i \setminus \mathbf{S}_{ij}} \Psi_i) = \sum_{\mathbf{C}_i \setminus \mathbf{S}_{ij}} real(\Psi_i) = \sum_{\mathbf{C}_i \setminus \mathbf{S}_{ij}} \Pr(\mathbf{C}_i, \mathbf{e}) = \Pr(\mathbf{S}_{ij}, \mathbf{e})$.

**Proof of Theorem 3**

After propagation completes, $\Psi_i$ consists of the product of the tables $\phi_{i1} ... \phi_{in}$ assigned to cluster $i$, and the incoming messages $M_{ji}$. Then $(\sum_{\mathbf{C}_i \setminus \mathbf{X}_k} \Psi_i)/\phi_{ik} = \sum_{\mathbf{C}_i \setminus \mathbf{X}_k} \prod_{m \neq k} \phi_{im} \prod_j M_{ji}$, which contains the partial derivatives of $\Pr(\mathbf{e})$ with respect to the parameters of $\phi_{ik}$ [8].