A Time and Space Efficient Junction Tree Architecture

Stephen Pasteris

Department of Computer Science
University College London
London WC1E 6BT, UK
s.pasteris@cs.ucl.ac.uk

May 11, 2014

Abstract

The Hugin and Shafer-Shenoy architectures are variations on the junction tree algorithm which tradeoff space and time complexities in different ways. The Hugin architecture is faster than that of Shafer-Shenoy but at the cost of a greater space complexity. This paper presents a new architecture which, in the cases where there exist relatively large vertices of the junction tree with high degree (or with many small corresponding factors), is significantly faster than Hugin propagation. The time complexity of the new architecture is never more than a logarithmic factor more than that of Hugin propagation. Like Shafer-Shenoy propagation, the space complexity of the new architecture is only exponential in the cardinality of the largest factor or separator, rather than being exponential in the cardinality of the largest vertex as in Hugin propagation.

1 Introduction

The junction tree algorithm is a way of computing marginal probabilities from graphical models that are stored in a factored form. The algorithm first converts the graph underlying the graphical model into a tree called a junction tree. The vertices of the junction tree are sets of vertices in the original graph. The algorithm then performs what is similar to belief propagation on the junction tree to compute the marginals. The Hugin and Shafer-Shenoy architectures are variations on the junction tree algorithm that differ in the way that they process messages. The Hugin architecture has a lower time complexity than the Shafer-Shenoy architecture but at the sake of a higher space complexity. In this paper we introduce a new junction tree architecture that is essentially at least as fast as Hugin propagation and takes essentially the same space as Shafer-Shenoy propagation (by “essentially” we mean up to a factor that is linear in the cardinality of the largest vertex of the junction tree). When the junction tree has some relatively large vertices of high degree (or with many small corresponding factors), the new architecture is significantly faster than both Hugin and Shafer-Shenoy propagation.

This paper is structured as follows: In section 2 we define the notation and terminology used by the paper. In section 3 we define the problem that the junction tree algorithms solve as well as comparing the complexity of the new architecture to that of Hugin and Shafer-Shenoy. In section 4 we give an overview of the new architecture. In section 5 we derive and detail the algorithms used in the architecture. In section 6 we show how to handle zeros.

2 Definitions

In this section we define the notation and terminology we will need to use.

2.1 Miscellaneous

The brackets ( ) and [ ], if they do not contain a comma or bar inside, are standard parenthesis. The notations ( , ), [ , ] and [ ] have a different meaning and are defined later.

Given a set $X$ we define $\mathcal{P}(X)$ to be the power set of $X$ (that is, the set of all possible subsets of $X$) and define $|X|$ to be the cardinality of $X$. A multiset is a set that may contain duplicate elements (we use the subset symbol, $S \subseteq X$, to denote that every element in the multiset $S$ is also contained in the set $X$).

The function $\exp$ is exponential base 2. The function $\log$ is logarithm base 2.

2.2 Trees

Given a rooted tree $T$, we define $\mathcal{V}(T)$, $\mathcal{E}(T)$ and $r(T)$ to be the vertex set, edge set, and root of $T$ respectively. We also denote the vertex set of $T$ by $T$ (where unambiguous).
We define $T^\bullet$ to be the leaves of $T$ and $T^\circ$ to be the internal vertices of $T$ (that is, $\nu(T) \setminus T^\bullet$).

Given a vertex $v$ in a rooted tree $T$ we define $\uparrow(v), \downarrow(v)$ and $\eta(v)$ to be the parent (if one exists), set of children and set of neighbours of $v$ respectively. We define $\downarrow'(v)$ to be the (rooted) subtree of $T$ induced by the descendants of $v$ and $\hat{\uparrow}(v)$ to be the set of ancestors of $v$. We also define $\deg(v)$ to be the degree of $v$.

If $T$ is a full binary tree (that is, a rooted binary tree in which every internal vertex has two children) we assume that it contains an orientation: That is, for every internal vertex $v \in T^\circ$, one child of $v$ is denoted as the left-child, $\triangledown(v)$, of $v$, and the other child of $v$ is denoted as the right-child, $\triangledown'(v)$, of $v$.

A depth first search of a full binary tree $T$ is as follows: We start at $r(T)$. Upon the first time we reach an internal vertex $v \in T^\circ$ we move next to $\triangledown(v)$ and upon the second time we reach an internal vertex $v \in T^\circ$ we move next to $\triangledown'(v)$. Upon the third time we reach an internal vertex $v \in (r(T))$ or a leaf $v \in T^\bullet$ we move next to $\uparrow(v)$. Upon the third time we reach $r(T)$ we terminate the depth first search.

### 2.3 Potentials

A potential on a set $X$ is a function from $\mathcal{P}(X)$ to $\mathbb{R}^\plus$. Note that a potential on a set $X$ represents a function from the set of all binary valued labelings of $X$ into $\mathbb{R}^\plus$ since each $Y \in \mathcal{P}(X)$ corresponds to the labelling $\mu$ of $X$ in which for every $v \in Y$, $\mu(v) := 1$ and for every $v \in X \setminus Y$, $\mu(v) := 0$.

Given a set $X$, the set of possible potentials on $X$ is denoted $\mathcal{T}(X)$. Given a set $X$ and a potential $\Psi \in \mathcal{T}(X)$ we define $\mid \Psi \mid := |X|$.

Given a set $X$, a subset $Y \subseteq X$ and a potential $\Psi \in \mathcal{T}(Y)$, the extension, $[\Psi, X]$, of $\Psi$ to $X$ is the potential in $\mathcal{T}(X)$ that satisfies, for every $Z \in \mathcal{P}(X)$, $[\Psi, X]|(Z) = \Psi(Z \cap Y)$.

Given a set $X$, a potential $\Psi \in \mathcal{T}(X)$, and a subset $Y \subseteq X$ we define the Y-marginal of $\Psi$, $(\Psi, Y)$, to be the potential in $\mathcal{T}(Y)$ that satisfies, for every subset $Z \subseteq Y$, $\Psi(Y \setminus Z) := \sum_{U \subseteq \mathcal{P}(X):U \cap \mathcal{P}(Y) = Z} \Psi(U)$.

Given a set $X$ and potentials $\Psi, \Theta \in \mathcal{T}(X)$ we define the product $\Psi \Theta$ to be the potential in $\mathcal{T}(X)$ that satisfies, for all $Y \in \mathcal{P}(X)$, $\Psi \Theta(Y) := \Psi(Y) \Theta(Y)$. We shall use the product symbol $\prod$ to denote the product of a set of potentials (e.g. Given a set $X$ and a set of potentials $\Lambda \subseteq \mathcal{T}(X)$ we have that, for all $Y \in \mathcal{P}(X)$, $\prod_{\Psi \in \Lambda} \Psi(Y) = \prod_{\Psi \in \Lambda} \Psi(Y)$).

Given a set $X$ and potentials $\Psi, \Theta \in \mathcal{T}(X)$ we define the quotient $\Psi / \Theta$ to be the potential in $\mathcal{T}(X)$ that satisfies, for all $Y \in \mathcal{P}(X)$, $[\Psi / \Theta](Y) := \Psi(Y) / \Theta(Y)$.

### 2.4 Formats

This paper utilises three types of format for potentials: standard format (SF), fast product format (FPF) and inclusion-exclusion format (IEF) [4]. The different formats are different ways of encoding a potential.

Given a set $X$ and potential $\Psi \in \mathcal{T}(X)$, the SF of $\Psi$ is simply the potential $\Psi$. The FPF, $\Psi^\triangledown$, and IEF, $\Psi^\triangledown$, of $\Psi$ are the potentials in $\mathcal{T}(X)$ that satisfy, for all $Y \in \mathcal{P}(X)$

\begin{align*}
\Psi^\triangledown(Y) &:= \prod_{Z \in \mathcal{P}(Y)} \Psi(Z)^{|\Psi(Z)|} \tag{1} \\
\Psi^\triangledown(Y) &:= \sum_{Z \in \mathcal{P}(X):Y \subseteq Z} \Psi(Z) \tag{2}
\end{align*}

### 2.5 Data-Structures

A mapped tree, $T$, is a full binary tree $D(T)$ with maps $\phi: D(T)^\circ \rightarrow \mathbb{N}$, and $\psi: D(T)^\bullet \rightarrow \mathbb{R}^\plus$ such that, for every $v \in D(T)^\circ$ with $\phi(v) \in D(T)^\circ$ (resp. $\psi(v) \in D(T)^\circ$) we have $\phi(\phi(v)) > \phi(v)$ (resp. $\psi(\phi(v)) > \psi(v)$).

Given mapped trees $S$ and $T$ a map $\pi: D(S) \rightarrow D(T)$ is called an isomorphism is structure from $S$ to $T$ iff it is an isomorphism from $D(S)$ to $D(T)$ that preserves left-child and right-child orientations and we have, for all $v \in D(S)^\circ$, $\phi(v) = \phi(\pi(v))$. If there exists an isomorphism in structure from $S$ to $T$ then $S$ and $T$ are said to be isomorphic in structure. A map $\pi: D(S) \rightarrow D(T)$ is called an isomorphism from $S$ to $T$ if it is an isomorphism in structure from $S$ to $T$ and for all $v \in S^\bullet$ we have $\psi(v) = \psi(\pi(v))$. If there exists an isomorphism from $S$ to $T$ then $S$ and $T$ are said to be isomorphic.

Given a mapped tree $T$ we denote (where unambiguous) the tree $D(T)$ as well as its vertex set by $T$.

Given a mapped tree $T$ we define its underlying set, $\mathcal{E}(T)$, as \{\phi(v) : v \in T^\circ\}. Given a leaf $l \in T^\bullet$ we define its corresponding set, $\mathcal{E}(l)$, as \{\phi(w) : w \in \uparrow(l)\}.

Given a mapped tree $T$ we define its corresponding potential, $\mathfrak{I}(T)$, to be the potential in $\mathcal{T}(\mathcal{E}(T))$ that satisfies, for all $l \in T^\bullet$, $\mathfrak{I}(T)(\mathcal{E}(l)) := \psi(l)$ and for all $Y \in \mathcal{P}(\mathcal{E}(T))$ for which there exists no $l \in T^\bullet$ with $\mathcal{E}(l) = Y$ we have $\mathfrak{I}(T)(Y) := 1$.

Given mapped trees $S$ and $T$ we say that $T$ is a compression of $S$ if and only if $\{\phi(v) : v \in T^\bullet\} \subseteq \{\phi(v) : v \in S^\bullet\}$. A mapped tree $T$ is called an info-tree if all $v \in T^\circ$ we have that $\downarrow'(\phi(v))$ is a compression of $\downarrow'(\phi(v))$.

Given a potential $\Psi$ we denote by $\Xi(\Psi)$ the unique (up to isomorphism) balanced info-tree (that is, an info-tree where each leaf is of the same depth) for which $\mathfrak{I}(\Xi(\Psi)) = \Psi$. 
Given a balanced info-tree $T$ (in which the map $\psi$ need not be defined), a potential $\Psi \in \mathcal{T}(\mathcal{C}(T))$, and a vertex $v \in T$, we define the restriction of $\Psi$ to $v$, $[\Psi|v]$ as the potential in $\mathcal{T}(\mathcal{C}(T)) \setminus \{\phi(w) : w \in \mathcal{E}(v), w \not\in v\}$ which satisfies, for all $Y \in \mathcal{P}(\mathcal{C}(T)) \setminus \{\phi(w) : w \in \mathcal{E}(v), w \not\in v\}$, $[\Psi|v](Y) := \Psi(Y \cup \{\phi(w) : \phi(w) \in \mathcal{E}(v)\})$.

2.6 Junction Trees

Given a some set $X$, a junction tree on $X$ is a rooted tree $J$ with the following properties:

1. $\mathcal{V}(J) \subseteq \mathcal{P}(X)$
2. $\bigcup \mathcal{V}(J) = X$
3. (Running intersection property) If $\Gamma$ and $\Delta$ are vertices of $J$ and there exists some $v \in X$ with $v \in \Gamma$ and $v \in \Delta$ then $v$ is contained in every vertex in the path from $\Gamma$ to $\Delta$.

Given a junction tree $J$ on a set $X$, a factorisation on $J$ is a function $F$ with domain $\mathcal{V}(J)$ that satisfies: For every $\Gamma \in \mathcal{V}(J)$, $F(\Gamma)$ is a set such that for every $\Psi \in F(\Gamma)$ there exists a subset $Y \subseteq \Gamma$ such that $\Psi \in \mathcal{T}(Y)$.

3 The Problem and Results

In this section we detail the problem that the junction tree algorithms solve as well as giving a comparison of the complexity of the new architecture to Hugin and Shafer-Shenoy propagation.

The general problem of this paper is as follows: We have, for some $n \in \mathbb{N}$, a junction tree $J$ on $\mathbb{N}$, and a factorisation $F$ on $J$. We assume that for every $v \in \mathbb{N}$ there exists a $\Gamma \in \mathcal{V}(J)$ and a potential $\Theta \in F(\Gamma)$ such that there exists a $\in \mathcal{P}(X)$ with $v \in \mathcal{Y}$ and $\Theta \in \mathcal{T}(\mathcal{Y})$. We wish to compute, for every element $v \in \mathbb{N}$, the quantity $\rho(v) := \left(\prod_{\Gamma \in \mathcal{V}(J)} \prod_{\Theta \in F(\Gamma)} |\Theta, \mathbb{N}|, \{v\}\right)$.

The time and space complexities $[\prod]$ of solving this problem under the different architectures are as follows:

- Hugin propagation has a time complexity of:
  $\Omega(\sum_{\Gamma \in \mathcal{V}(J)} (\deg \Gamma + |F(\Gamma)|) \exp |\Gamma|)$
  and a space complexity of:
  $\Omega(\sum_{\Gamma \in \mathcal{V}(J)} \exp |\Gamma| + \sum_{\Theta \in F(\Gamma)} \exp |\Theta|)$

- Shafer-Shenoy propagation has a time complexity of:
  $\Omega(\sum_{\Gamma \in \mathcal{V}(J)} \deg \Gamma (|F(\Gamma)| \exp |\Gamma|)$
  and a space complexity of:
  $\Omega((\sum_{\Gamma \in \mathcal{V}(J)} \sum_{\Theta \in F(\Gamma)} \exp |\Theta|) + (\sum_{(\Gamma, \mathcal{Y}) \in \mathcal{E}(J)} \exp |\Gamma \cap \mathcal{Y}|)$

- The new architecture has a time complexity of:
  $O(\sum_{\Gamma \in \mathcal{V}(J)} (|\Gamma| \exp |\Gamma| + \sum_{\Theta \in F(\Gamma)} |\Theta| \exp |\Theta|))$

Note that the time complexity of the new architecture is never more than a logarithmic factor more than that of Hugin (and hence also Shafer-Shenoy) propagation and, when we have relatively high cardinality vertices in $J$ of high degree then the new architecture is significantly faster than that of Hugin and Shafer-Shenoy. Note also that, like Shafer-Shenoy propagation, the space complexity of the new architecture is only exponential in the cardinality of the largest factor or separator, rather than being exponential in the cardinality of the largest vertex as in Hugin propagation. It is also true (although isn’t shown in the above bounds) that the space requirement of the new architecture is never more than a constant factor more than that of Hugin propagation.

We now give an example illustrating the speed increase of the new architecture over both Hugin and Shafer-Shenoy propagation:

Consider the following junction tree: The root, $\Gamma$ has cardinality $2 \log(m)$ and $\Gamma$ has $m$ children $\{\Gamma_1, \Gamma_2, ..., \Gamma_m\}$ which are all leaves. Each child of $\Gamma$ has cardinality $\log(m) + 1$. For all $i, j \in \mathbb{N}$ with $i \neq j$ we have that $|\Gamma_i \cap \Gamma_j| = \log(m)$ and $\Gamma_i \cap \Gamma_j \neq \emptyset$. We note that for $m$ large enough such a junction tree exists. Suppose that for each vertex $\mathcal{Y}$ in the junction tree we have a single factor at $\mathcal{Y}$ who’s underlying set is equal to $\mathcal{Y}$.

The times taken, on this family of junction trees, by the different architectures are as follows:

- Shafer-Shenoy: $\Omega(m^4)$
- Hugin: $\Omega(m^3)$
- New architecture: $O(m^2 \log(m))$.

4 Overview of the Architecture

In this section we give an overview of the new architecture for computing the problem of section 3.

The following algorithm solves the problem of section 3 (the correctness of the algorithm is seen by noting its equivalence to Hugin propagation). Like the other architectures the algorithm computes, for every edge $(\Gamma, \mathcal{Y}) \in \mathcal{E}(J)$ messages $M_{\Gamma \rightarrow \mathcal{Y}}$ and $M_{\mathcal{Y} \rightarrow \Gamma}$ which are both potentials in $\mathcal{T}(\Gamma \cap \mathcal{Y})$. The messages and potentials $\{\rho(v) : v \in \mathbb{N}\}$ are computed from other messages and the factors associated with a vertex in the junction tree.

Algorithm 1.

1. Perform a depth first search of $J$. Upon the last time a
vertex $\Gamma \in V(J) \setminus \{r(J)\}$ is encountered perform the following:

(a) $M_{\Gamma \rightarrow (\uparrow \Gamma)} \leftarrow \left(\prod_{\Theta \in F(\Gamma)} \Theta, \Gamma\right) \prod_{\Lambda \in \Lambda(\Gamma)} [M_{\Lambda \rightarrow \Gamma}, \Gamma], \Gamma \cap \uparrow(\Gamma)

2. Perform a depth first search of $J$. Upon the first time some vertex $\Gamma$ is encountered perform the following:

(a) For every $Y \in \downarrow(\Gamma)$ set $N_Y \leftarrow \left(\prod_{\Theta \in F(\Gamma)} \Theta, \Gamma\right) \prod_{\Lambda \in \Lambda(Y)} [M_{\Lambda \rightarrow \Gamma}, \Gamma], \Gamma \cap Y

For every $v \in \Gamma$ set $\rho(v) \leftarrow \left(\prod_{\Theta \in F(\Gamma)} \Theta, \Gamma\right) \prod_{\Lambda \in \Lambda(v)} [M_{\Lambda \rightarrow \Gamma}, \Gamma], \{v\}

(b) For every $Y \in \downarrow(\Gamma)$ set $M_{\Gamma \rightarrow Y} \leftarrow N_Y/M_{\Gamma \rightarrow \Gamma}$

Note that in stage \[\text{Ia}\] of algorithm \[\text{I}\] we have that $\prod_{\Lambda \in \Lambda(\Gamma)} [M_{\Lambda \rightarrow \Gamma}, \Gamma] = \{1, \Gamma\} \prod_{\Lambda \in \Lambda(\Gamma)} [M_{\Lambda \rightarrow \Gamma}, \Gamma]$ where $1$ is the potential in $T((\uparrow \Gamma) \cap \Gamma)$ which satisfies, for every $Y \in P((\uparrow \Gamma) \cap \Gamma), 1(Y) := 1$. Note also that in stage \[\text{IIa}\] of algorithm \[\text{II}\] we have that $\prod_{\Lambda \in \Lambda(\Gamma)} [M_{\Lambda \rightarrow \Gamma}, \Gamma] = \left(\prod_{\Theta \in F(\Gamma)} \Theta, \Gamma\right) \prod_{\Lambda \in \Lambda(Y)} [M_{\Lambda \rightarrow \Gamma}, \Gamma]$ where $1$ is the potential in $T((\Gamma \cap v)$ that satisfies, for every $Y \in P(\{v\}), 1(y(Y) := 1$.

Stages \[\text{Ia}\] and \[\text{IIa}\] of algorithm \[\text{I}\] can hence be performed with instances of the following operation:

**Operation 2.** We have a set $\Gamma \subseteq N_n$, a multiset of subsets $\{\Delta_i : i \in N_k\} \subseteq P(\Gamma)$ with $\bigcup_{i \in N_k} \Delta_i = \Gamma$, and a multiset of potentials $\{\Phi_i : i \in N_k\}$ such that, for every $i \in N_k$, we have $\Phi_i \in T(\Delta_i)$. We wish to compute, for every $j \in N_k$, the potential $\prod_{i \in N_k} [\Phi_i, \Gamma], \Delta_j$.

We now give an overview of an algorithm for performing operation \[\text{II}\] in a time of $O\left(|\Gamma| \exp |\Gamma| + \sum_{i=1}^k |\Delta_i| \exp |\Delta_i|\right)$ and taking a space of $O\left(|\Gamma| \left(\sum_{i=1}^k \exp |\Delta_i|\right)\right)$ hence giving, for algorithm \[\text{I}\] the time and space complexities stated in section \[\text{II}\].

**Algorithm 3.** Let $\Psi := \prod_{i \in N_k} [\Phi_i, \Gamma]$.  

1. For every $i \in N_k$ compute $\Xi(\Phi_i)$ from $\Xi(\Phi_i)$. (See section \[\text{III}\].)

2. From the multiset $\{\Xi(\Phi_i) : i \in N_k\}$ compute the info-tree $S$ that satisfies the following properties:

(a) For each leaf $l \in S^*$ there exists an $i \in N_k$ such that $\Xi(l) \in P(\Delta_i)$.

(b) For each $i \in N_k$ and each set $Y \in P(\Delta_i)$ there exists a leaf $l \in S^*$ such that $\Xi(l) = Y$.

(c) $\Xi(S) = \Psi$. (See section \[\text{III}\].)

3. From $S$ compute the info-tree $T$ that is isomorphic to structure $S$ and such that satisfies, for each leaf $l \in T^*$, $\psi(l) = \Psi^*(\Xi(l))$. (See section \[\text{III}\].)

4. From $T$ compute the multiset $\{\Xi(\Psi, \Delta_i) : i \in N_k\}$. (See section \[\text{IV}\].)

5. For every $i \in N_k$ compute $\Xi(\Psi, \Delta_i)$ from $\Xi(\Psi, \Delta_i)$. (See section \[\text{IV}\].)

5 Details of the Architecture

In this section we derive efficient algorithms for computing each step of algorithm \[\text{III}\]. This section uses the notation (i.e., $\Gamma, \Psi, \Phi_i, S$ and $T$) given in operation \[\text{II}\] and algorithm \[\text{III}\]. Some of the algorithms in this section rely on the following lemma:

**Lemma 4.** Given a potential $\Theta \in T(\emptyset)$ we have that $\Theta = \Theta' = \Theta^*$.

5.1 From SF to FP

In this subsection we derive an efficient algorithm for computing stage \[\text{II}\] of algorithm \[\text{III}\]. We start with the following lemma:

**Lemma 5.** Suppose we have a set $X$ and a potential $\Theta \in T(X)$. Suppose we have some element $v \in X$. Let $\Theta^*$ be the potential in $T(X \setminus \{v\})$ that satisfies, for all $Y \in P(X \setminus \{v\}), \Theta^*(Y) := \Theta(Y)$ and let $\Theta^+$ be the potential in $T(X \setminus \{v\})$ that satisfies, for all $Y \in P(X \setminus \{v\}), \Theta^+(Y) := \Theta(Y \cup \{v\})$. Then for all $Y \in P(X \setminus \{v\})$ we have $\Theta^*(Y) = \Theta^*(Y) \cdot \Theta^+(Y \cup \{v\})^{-1}$

Lemma \[\text{V}\] readily gives the following algorithm for computing, for some potential $\Phi$ and some vertex $v \in \Xi(\Phi)$, $\Xi(\Phi|v)$ from $\Xi(\Phi|v)$ and $\Xi(\Phi|v)$.  

**Algorithm 6.** Input $P \leftarrow \Xi(\Phi|v)$, $Q \leftarrow \Xi(\Phi|v)$. Let $\pi$ be the isomorphism in structure from $P$ to $Q$. Perform simultaneous depth-first searches of $P$ and $Q$ (i.e. when we are at a vertex $w$ in $P$ we are at the vertex $\pi(w)$ in $Q$). When we encounter a leaf $l \in P$ we set $\psi(\pi(l)) \leftarrow \psi(l) \psi(\pi(l))$. After the simultaneous depth-first searches we construct the info-tree $R$ such that $\phi(r(R)) = \phi(v)$ and $\Xi(\Phi|v)$ is isomorphic to $P$ (resp. $Q$). Output $\Xi(\Phi|v) \leftarrow R$.

We now give an algorithm for converting, for some potential $\Phi$, $\Xi(\Phi)$ to $\Xi(\Phi)$ in time of $O(|\Phi| \exp |\Phi|)$ and a space of $O(|\Phi| \exp |\Phi|)$ (this algorithm is used to perform stage \[\text{II}\] of algorithm \[\text{III}\].)

**Algorithm 7.** Input $P \leftarrow \Xi(\Phi)$. Perform a depth-first search of $P$. Upon reaching a leaf $l \in P^*$, construct (and store at $l$) $\Xi(\Phi|v)$ as follows: $\Xi(\Phi|v)$ has a single vertex $r$ and $\psi(r) = \psi(l)$ (by lemma \[\text{II}\]). Upon reaching a vertex $v \in P^*$ for the final time, construct (and store at vertex $v$) $\Xi(\Phi|v)$ from $\Xi(\Phi|v)$ and $\Xi(\Phi|v)$ using algorithm \[\text{VI}\] and then delete $\Xi(\Phi|v)$ and $\Xi(\Phi|v)$.
Upon termination of the depth first search output $\Xi(\Phi') \leftarrow \Xi'(\Phi r(P))$

5.2 Products of Extensions in FPF

In this subsection we derive an efficient algorithm for computing stage 2 of algorithm 3. We start with the following two lemmas:

**Lemma 8.** Given a set $\Gamma$, a subset $\Delta \subseteq \Gamma$ and a potential $\Phi \in \mathcal{T}(\Delta)$ we have, for every $Z \in \mathcal{P}(\Gamma)$ that $[\Phi, \Gamma]'(Z) = \Phi'(Z)$, and for every $Z \in \mathcal{P}(\Gamma) \setminus \mathcal{P}(\Delta)$ that $[\Phi, \Gamma]'(Z) = 1$.

**Lemma 9.** Given a set $\Gamma$, and a multiset of potentials $\{\Phi_i : i \in N_k\} \subseteq \mathcal{T}(\Gamma)$, we have that $\prod_{i=0}^{k}[\Phi_i, \Gamma]' = \prod_{i=0}^{k}[\Phi_i, \Gamma]$.

Lemmas 8 and 9 combine to give the following lemma:

**Lemma 10.** Suppose we have a set $\Gamma$, a multiset of subsets $\{\Delta_i : i \in N_k\} \subseteq \mathcal{P}(\Gamma)$, and a multiset of potentials $\{\Phi_i : i \in N_k\}$ where each $\Phi_i$ is in $\mathcal{T}(\Delta_i)$. Then given any $Y \in \mathcal{P}(\Gamma)$:

$$\prod_{i=0}^{k}[\Phi_i, \Gamma]'(Y) = \prod_{i \in N_k: Y \subseteq \Delta_i}[\Phi_i, \Gamma]'(Y) \tag{3}$$

Noting that, by definition, $[\Phi, \Gamma]' = \prod_{i=0}^{k}[\Phi_i, \Gamma]'$, lemma 10 readily gives the following algorithm for computing stage 2 of algorithm 3 in a time of $O\left(\exp |\Gamma| + \sum_{i=1}^{k} \exp |\Delta_i|\right)$ and a space of $O\left(\sum_{i=1}^{k} \exp |\Delta_i|\right)$ (noting that $|S| < 2 \sum_{i=1}^{k} \exp |\Delta_i|$).

**Algorithm 11.** We maintain an array $A$ of length $n$ (note that the array adds nothing to the space complexity of algorithm 1 and is hence not included in the space complexity of this operation) where each element of $A$ is a set of (pointers to) vertices. The only elements of $A$ that are utilised in this algorithm are those $A(i)$ where $i \in \Gamma$. Initially we have, for all $i \in \Gamma$, $A(i) \leftarrow \emptyset$. We also maintain a set $L$ of (pointers to) vertices, initially empty. At any point in the algorithm we define, for any $i \in \Gamma$, the value $\mu_i$ as $\min\{\{j : i \in \Gamma, A(j) \neq \emptyset\}\}$, where $\min(\emptyset)$ is defined to be equal to $\infty$ (we compute $\mu_i$ by moving, in ascending order, from $i$, through the elements of $\Gamma$, checking, at each $j$ in $\Gamma$ we encounter, to see if the set $A(j)$ is empty). We take as input, for every $i \in N_k$, $R_i \leftarrow [\Phi_i, \Gamma]$. Throughout the algorithm we build the info-tree $S$, which is initialised to a single vertex $r(S)$, designated an internal vertex with $\phi(r(S)) = \min(\Gamma)$. Firstly, for every $i \in N_k$ we add (a pointer to) $r(R_i)$ to $A(\phi(r(R_i)))$. We then run the following algorithm, based on a depth first search through the vertices of $S$ as it is constructed:

Start at vertex $r(S)$.

Upon the first (resp. second) time we encounter a vertex $v \in S$ that is designated as an internal vertex we perform the following: For every $w \in A(\phi(v))$, if $\phi(w) \neq \phi(v)$ is an internal vertex add (a pointer to) $\phi(w)$ to $A(\phi(w))$. If $\phi(w) = \phi(v)$ and $\phi(v)$ is a leaf add (a pointer to) $\phi(w)$ to $L$. Add a vertex $z$ to $S$ as the left-child (resp. right-child) of $v$. If $\mu_z \neq \infty$ then $z$ is designated as an initial vertex with $\phi(z) = \mu_z$. If $\mu_z = \infty$ then $z$ is designated as a leaf. Move next to $z$.

Upon the third and final time we encounter a vertex $v \in S$ that is designated as an internal vertex we set $A(\phi(v)) \leftarrow \emptyset$ and move next to $\uparrow(v)$ (unless $v = r(S)$ in which case the algorithm terminates, outputting $S$).

Upon encountering a vertex $v \in S$ that is designated as a leaf we set $\psi(v) \leftarrow \prod_{w \in L}[\Phi, \Gamma]$ and then $L \leftarrow \emptyset$. We move next to $\uparrow(v)$.

5.3 From FPF to IEF

In this subsection we derive an efficient algorithm for computing stage 3 of algorithm 3. We start with the following definitions:

Note first that if $l$ is the least leaf of $S$ then $\{\phi(w) : w \in \bigcap\{\{l\}\}\} = \Gamma$.

Given $i \in \Gamma$ we define the info-tree $S_i$ as follows: Let $l$ be the least leaf of $S$. Let $v$ be the unique vertex in $\bigcap\{\{l\}\}$ for which $\phi(v) = i$. Let $S_i := \Psi(v)$. We compute $S_i$ by first climbing $S$ from the left-most leaf, $l$, until we reach the vertex $v$ with $\phi(v) = i$.

Given some $i \in \Gamma$ and a potential $\Phi \in \mathcal{T}(\{j : j \in \Gamma, j \geq i\})$ we define $\Xi_S(\Phi)$ to be an info-tree that is isomorphic in structure to $S_i$ and satisfies, for all $\ell \in \Xi_S(\Phi^*)$, $\psi(\ell) = \Phi(\Xi(l))$. Let $B$ be the info-tree that is isomorphic in structure to $\Xi(\Psi)$ (the map $\psi$ on $B^*$ is arbitrary). $B$ is regarded as purely a mathematical object - never being stored in full in the memory.

The first part of this subsection relies on the following lemma:

**Lemma 12.** Suppose we have a set $X$ and a potential $\Theta \in \mathcal{T}(X)$. Suppose we have some element $v \in X$. Let $[\Theta_+]$ be the potential in $\mathcal{T}(X \setminus \{v\})$ that satisfies, for all $Y \in \mathcal{P}(X \setminus \{v\})$, $[\Theta_+](Y) := \Theta(Y)$ and let $[\Theta_-]$ be the potential in $\mathcal{T}(X \setminus \{v\})$ that satisfies, for all $Y \in \mathcal{P}(X \setminus \{v\})$, $[\Theta_-](Y) := \Theta(Y \cup \{v\})$. We have $[\Theta_-](Y) = [\Theta_+](Y)$ and $[\Theta_+](Y)' = [\Theta_-'(Y)\Theta'(Y \cup \{v\})^{-1}$

**Lemma 12** gives us the following lemma, proved by induction on the depth of $v$:

**Lemma 13.** For all $v \in B$ we have $\Xi(\Xi_S([\Psi[v]]) = \Psi[v])$.

**Lemmas 12 and 13** readily give the following algorithm for computing, for some vertex $v \in B^*$, $\Xi_S([\Psi[v]])$ and $\Xi_S([\Psi[v]])$ from $\Xi_S([\Psi[v]])$: 

- **Algorithm 13.** We maintain an array $A$ of length $n$ (note that the array adds nothing to the space complexity of algorithm 1 and is hence not included in the space complexity of this operation) where each element of $A$ is a set of (pointers to) vertices. The only elements of $A$ that are utilised in this algorithm are those $A(i)$ where $i \in \Gamma$. Initially we have, for all $i \in \Gamma$, $A(i) \leftarrow \emptyset$. We also maintain a set $L$ of (pointers to) vertices, initially empty. At any point in the algorithm we define, for any $i \in \Gamma$, the value $\mu_i$ as $\min\{\{j : i \in \Gamma, A(j) \neq \emptyset\}\}$, where $\min(\emptyset)$ is defined to be equal to $\infty$ (we compute $\mu_i$ by moving, in ascending order, from $i$, through the elements of $\Gamma$, checking, at each $j$ in $\Gamma$ we encounter, to see if the set $A(j)$ is empty). We take as input, for every $i \in N_k$, $R_i \leftarrow [\Phi_i, \Gamma]$. Throughout the algorithm we build the info-tree $S$, which is initialised to a single vertex $r(S)$, designated an internal vertex with $\phi(r(S)) = \min(\Gamma)$. Firstly, for every $i \in N_k$ we add (a pointer to) $r(R_i)$ to $A(\phi(r(R_i)))$. We then run the following algorithm, based on a depth first search through the vertices of $S$ as it is constructed:

Start at vertex $r(S)$.
Algorithm 14. Input \( P \leftarrow \downarrow(\psi(r(S_{\Psi|v}^\ast))) \), \( Q \leftarrow \downarrow(\phi(r(S_{\Psi|v}^\ast))) \). Construct an info-tree \( R \) that is isomorphic to \( P \).

Perform a depth-first search of \( P \). During the depth first search we have a single vertex \( f \in Q \) that is flagged. We initialise with \( f \leftarrow r(Q) \). Upon the first time we reach an internal vertex \( v \in P^o \), if \( f \in Q^o \) and \( \phi(f) = \phi(v) \) then set \( f \leftarrow \psi(f) \). Upon the second time we reach an internal vertex \( v \in P^o \), if \( f \in Q^o \) and \( \phi(f) = \phi(v) \) then set \( f \leftarrow \psi(f) \). Upon the third and final time we reach an internal vertex \( v \in P^o \), if \( f \in Q^o \) and \( \phi(f) = \phi(v) \) then set \( f \leftarrow \psi(f) \) (if \( f \neq r(Q) \) - else terminate the depth first search of \( P \)). When we encounter a leaf \( l \in P^* \), if \( f \in Q^* \) then set \( \psi(l) \leftarrow \psi(l)/\psi(f) \) and set \( f \leftarrow \psi(f) \) (if \( f \neq r(Q) \) - else terminate the depth first search of \( P \).

Upon termination of the depth first search output \( S_{\Psi|v}^\ast \leftarrow R \), \( S_{\Psi|v}^\ast \leftarrow L \).

The following lemma regards the recursive construction of the IEF of a potential:

Lemma 15. Suppose we have a set \( X \) and a potential \( \Theta \in T(X) \). Suppose we have some element \( v \in X \). Let \( \Theta \) be the potential in \( T(X \setminus \{v\}) \) that satisfies, for all \( Y \in P(X \setminus \{v\}) \), \( \Theta(Y) := \Theta(Y) \) and \( \Theta \) be the potential in \( T(X \setminus \{v\}) \) that satisfies, for all \( Y \in P(X \setminus \{v\}) \), \( \Theta(Y) := \Theta(Y) \). Then for all \( Y \in P(X \setminus \{v\}) \) we have \( \Theta(Y) = \Theta(Y \setminus \{v\}) + \Theta(Y \setminus \{v\}) \) and \( \Theta(Y) = \Theta(Y \setminus \{v\}) \).

Algorithm 16. Input \( P \leftarrow S_{\Psi|v}^\ast \), \( Q \leftarrow S_{\Psi|v}^\ast \). Let \( \pi \) be the isomorphism in structure from \( P \) to \( Q \). Perform simultaneous depth-first searches of \( P \) and \( Q \) (i.e. when we are at a vertex in \( P \) we are at the vertex \( \pi(v) \) in \( Q \)). When we encounter a leaf \( l \in P^* \) we set \( \psi(l) \leftarrow \psi(l) \). After the simultaneous depth first searches construct \( R \) as an info-tree isomorphic to \( \downarrow(\psi(r(S_{\Psi|v}))) \).

Perform a depth-first search of \( Q \). During the depth first search we have a single vertex \( f \in R \) that is flagged. We initialise with \( f \leftarrow r(R) \). Upon the first time we reach an internal vertex \( v \in Q^o \), if \( f \in R^o \) and \( \phi(f) = \phi(v) \) then set \( f \leftarrow \psi(f) \). Upon the second time we reach an internal vertex \( v \in Q^o \), if \( f \in R^o \) and \( \phi(f) = \phi(v) \) then set \( f \leftarrow \psi(f) \). Upon the third and final time we reach an internal vertex \( v \in Q^o \), if \( f \in R^o \) and \( \phi(f) = \phi(v) \) then set \( f \leftarrow \psi(f) \) (if \( f \neq r(R) \) - else terminate the depth first search of \( Q \)). When we encounter a leaf \( l \in Q^* \), if \( f \in R^* \) then set \( \psi(f) \leftarrow \psi(l) \) and set \( f \leftarrow \psi(f) \) (if \( f \neq r(R) \) - else terminate the depth first search of \( Q \).

After the depth first search we construct the info-tree \( U \) such that \( \phi(r(U)) = \phi(v) \) and \( \psi(\phi(r(U))) \) (resp. \( \psi(\phi(r(U))) \)) is isomorphic to \( P \) (resp. \( R \). Output \( S_{\Psi|v}^\ast \leftarrow U \).

We now define a ghost search of \( B \):

Algorithm 17. (Ghost search) We maintain throughout, a (dynamic) subtree, \( C \), of \( B \), initially containing \( r(B) \) as a single vertex. We start the ghost search at \( r(B) \).

When we encounter a vertex \( v \in B^o \) for the first time we add \( \psi(v) \) and \( \phi(v) \) to \( C \) and then move to \( \phi(v) \). When we encounter a vertex \( v \in B^o \) for the second time we move to \( \psi(v) \). When we encounter a vertex \( v \in B^o \) for the first time we move to \( \phi(v) \) and \( \psi(v) \) from \( C \) and then move to \( \phi(v) \) (unless \( v = r(B) \) in which case the ghost search terminates). When we encounter a leaf \( l \in B^* \) we move next to \( l \).

We now give an algorithm for computing stage 3 of algorithm 3 in a time of \( O(\exp |\Gamma| \exp |\Gamma|) \) and with a space of \( O(|\Gamma| \sum_{i=1}^k \exp |\Delta_i|) \) (noting that, for every \( i \), \( |\Delta_i| \in O \left( \sum_{i=1}^k \exp |\Delta_i| \right) \)).

Algorithm 18. Perform a ghost search of \( B \) and let \( C \) be the (dynamic) subtree of \( B \) maintained throughout the ghost search. Initially we compute \( S_{\Psi|v}^\ast \) as isomorphic to \( S \) and store it at \( r(B) \).

Upon the first time we reach an internal vertex \( v \in B^o \) (and after its children are added to \( C \)) we compute \( S_{\Psi|v}^\ast \) and store it at \( \phi(v) \) and \( \psi(v) \) respectively. We then delete \( S_{\Psi|v}^\ast \).

Upon encountering a leaf \( l \in B^* \) we compute \( S_{\Psi|v}^\ast \) as isomorphic to \( S_{\Psi|v}^\ast \) (by lemma 3). We then delete \( S_{\Psi|v}^\ast \).

Upon the third and final time we reach an internal vertex \( v \in B^o \) (and before its children are removed from \( C \)) we compute \( S_{\Psi|v}^\ast \) and store it at \( \phi(v) \) and \( \psi(v) \) respectively.

When the ghost search terminates output \( T \leftarrow S_{\Psi|v}^\ast \).

5.4 Marginals in IEF

In this subsection we derive an efficient algorithm for computing stage 3 of algorithm 3. We start with the following lemma:

Lemma 19. Given a set \( \Gamma \), a potential \( \Psi \in T(\Gamma) \) and a subset \( \Delta \subseteq \Gamma \), then for all subsets \( Z \in P(\Delta) \) we have:

\[
(\Psi, \Delta)^*(Z) = \Psi^*(Z)
\]

(4)

Lemma 19 readily gives the following algorithm for computing stage 3 of algorithm 4 in a time of \( O(\exp |\Gamma| + \sum_{i=1}^k \exp |\Delta_i|) \) and a space of \( O\left( \sum_{i=1}^k \exp |\Delta_i| \right) \) (noting that \( |\Gamma| < 2 \sum_{i=1}^k \exp |\Delta_i| \)).

Algorithm 20. We maintain an array \( A \) of length \( n \) (note that the array adds nothing to the space complexity of algorithm 1) and is hence not included in the space complexity of
this operation) where each element of A is a set of (pointers to) vertices. The only elements of A that are utilised in this algorithm are those A(i) where i ∈ Γ. Initially we have, for all i ∈ Γ, A(i) ← ∅. We also maintain a set L of (pointers to) vertices, initially empty.

Initially, we construct, for every i ∈ Nk an info-tree Ri as isomorphic in structure to Ξ((Ξ), Δi)) (the map ψ is initially arbitrary).

Firstly, for every i ∈ Nk we add (a pointer to) r(Ri) to A(ϕ(r(Ri))). We then perform a depth-first search of T:

Upon the first (resp. second) time we encounter a vertex v ∈ T ∞ we perform the following: For every w ∈ A(ϕ(v)), if ϕ(w) (resp. ψ(w)) is an internal vertex add (a pointer to) ϕ(w) (resp. ψ(w)) to A(ϕ(ϕ(w))) and if ϕ(w) (resp. ψ(w)) is a leaf add (a pointer to) ϕ(w) (resp. ψ(w)) to L.

Upon the third and final time we encounter a vertex v ∈ T ∞ we set A(ϕ(v)) ← ∅.

Upon encountering a leaf l ∈ T ∞ we set, for every w ∈ L, ψ(w) ← ψ(l) and then set L ← ∅.

Output, for every i ∈ Nk, Ξ((Ξ), Δi)) ← Ri.

5.5 From IEF to SF

In this subsection we derive an efficient algorithm for computing stage 5 of algorithm 3. We start with the following lemma:

Lemma 21. Suppose we have a set X and a potential Θ ∈ T(X). Suppose we have element ψ ∈ X. Let [Θ−] be the potential in T(X \ {ψ}) that satisfies, for all Y ∈ P(X \ {ψ}), [Θ−](Y) := Θ(Y) and let [Θ+] be the potential in T(X \ {ψ}) that satisfies, for all Y ∈ P(X \ {ψ}), [Θ+](Y) := Θ(Y). We have [Θ−](Y) = Θ+(Y \ {ψ}) and [Θ+] = Θ+(Y \ {ψ}) + Θ−(Y \ {ψ}).

Lemma readily gives the following algorithm for computing, for some set X and some vertex v ∈ Ξ(Ξ), Ξ((Ξ), v) and Ξ((Ξ), v) from Ξ((Ξ), v):

Algorithm 22. Input P ← ψ(ϕ(Ξ((Ξ), v))) Q ← ψ(ϕ(Ξ((Ξ), v))). Let π be the isomorphism in structure from P to Q.

Perform simultaneous depth-first searches of P and Q (i.e. when we are at a vertex v in P we are at the vertex π(v) in Q). Upon encountering a leaf l ∈ P ∞ we set ψ(l) ← ψ(l) − ψ(π(l)).

Upon termination of the simultaneous depth-first searches we output Ξ((Ξ), v) ← P, Ξ((Ξ), v) ← Q.

We now give an algorithm for converting, for some potential Ξ, Ξ(Ξ) to Ξ(Ξ) in a time of O(|Ξ| exp |Ξ|) and a space of O(exp |Ξ|) (this algorithm is used to perform stage 5 of algorithm 3):

Algorithm 23. Construct P as an info-tree that is isomorphic in structure to Ξ(Ξ). Initialise with Ξ((Ξ), v) ← Ξ(Ξ).

6 A Note on Zeros

In the previous sections we assumed that each potential was a map into R+. We now describe how to deal with potentials which map into R+ ∪ {0}:

We have some algebraic value ǫ. For every factor we replace any zero in the factor with ǫ. We then run the algorithm. Throughout the algorithm we maintain the limit ǫ → 0 (i.e. Any number is stored as ae−m for some a ∈ R+ and m ∈ N. (e.g. the value 4e−2 + 7e−3, upon the limit ǫ → 0 is equal to, and hence stored as, 4e−2)).

References:

1. J. Park and A. Darwiche: Morphing the Hugin and Shenoy-Shafer Architectures.
2. F. V. Jensen, S. Lauritzen, and K. Olesen: Bayesian updating in recursive graphical models by local computation.
3. G. Shafer and P. Shenoy: Probability Propagation.
4. D. Smith and V. Gogate: The Inclusion-Exclusion Rule and its Application to the Junction Tree Algorithm.
5. J. Pearl: Reverand Bayes on Inference Engines: a Distributed Hierarchical Approach.