# Lifted Division for Lifted Hugin Belief Propagation

**Moritz P. Hoffmann**
singularIT GmbH
Lübeck, Germany

**Tanya Braun**
Computer Science Department,
University of Münster, Germany

**Ralf Möller**
Institute of Information Systems,
University of Lübeck, Germany

## Abstract

The lifted junction tree algorithm (LJT) is an inference algorithm that allows for tractable inference regarding domain sizes. To answer multiple queries efficiently, it decomposes a first-order input model into a first-order junction tree. During inference, degrees of belief are propagated through the tree. This propagation significantly contributes to the runtime complexity not just of LJT but of any tree-based inference algorithm. We present a lifted propagation scheme based on the so-called Hugin scheme whose runtime complexity is independent of the degree of the tree. Thereby, lifted Hugin can achieve asymptotic speed improvements over the existing lifted Shafer-Shenoy propagation. An empirical evaluation confirms these results.

## 1 INTRODUCTION

Lifted algorithms allow for tractable inference regarding domain sizes (Niepert and Van den Broeck 2014). With the lifted junction tree algorithm (LJT), Braun and Möller (2016) lift the propositional junction tree algorithm (JT) by Lauritzen and Spiegelhalter (1988) to efficiently handle multiple queries. Facing multiple queries is a problem that occurs during model learning as well as general query answering settings. With the advantage of tractability and the general applicability of the underlying modelling formalisms, advances in lifting may support applications in many areas such as healthcare (Gehrke et al. 2019).

A significant factor in the runtime of tree-based algorithms such as LJT is belief propagation, also called message passing, in the tree. The two best known propagation schemes for propositional junction trees (jtrees) are Shafer-Shenoy (Shafer and Shenoy 1990) and Hugin (Jensen et al. 1990). LJT implements a lifted version of the Shafer-Shenoy propagation on its first-order (FO) jtree. However, the advantage of Hugin over Shafer-Shenoy in terms of fewer calculations per message also applies to the lifted setting. Only, Hugin requires a lifted division operator that, to the best of our knowledge, does not exist.

Therefore, our contributions are twofold: (i) We solve the problem of first-order factor division by defining a lifted division operator. (ii) We present the lifted Hugin scheme as a means to efficiently organize message passing in FO-jtrees. Lifted operators are a fundamental building block of lifted inference algorithms such as lifted variable elimination (LVE) (Poole 2003) and LJT. As a lifted division operator has not yet been defined, we close a gap in the suite of lifted operators on probabilistic models. Our approach builds upon the formalism for lifted operators established by Taghipour, Davis, et al. (2013). With lifted Hugin, we achieve a belief propagation with a runtime polynomial in the domain size and per message, independent of the degree of the underlying FO-jtree. As such, LJT with lifted Hugin enables shorter answering times per query than lifted Shafer-Shenoy in different settings as evidenced by our empirical evaluation.

The paper is structured as follows: First, we look at related work. Then, we recap the modelling formalism and LJT including Shafer-Shenoy propagation. We follow with presenting our two main contributions, leading into a discussion and empirical evaluation. Last, we conclude and look at future work.

## 2 RELATED WORK

Zhang and Poole (1994) present variable elimination, which remains a cornerstone of probabilistic inference. Poole (2003) introduces the idea of exploiting symmetries for inference by proposing LVE. After the introduction of count conversion and counting random variables (CRVs, Milch et al. 2008) and other refine-

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. 

ments (Salvo Braz et al. 2005; Salvo Braz 2007; Friedman and Van den Broeck 2020), Taghipour, Fierens, et al. (2013) consolidate a standard LVE, using lifted operators decoupled from the model constraint language. Note: For lifted Hugin, we can treat a CRV as any other PRV but with special ranges in the form of histograms. For a definition of CRVs and count conversion, refer to Milch et al. (2008). Taghipour, Davis, et al. (2013) analyze LVE and its runtime complexity.

JT dates back to Lauritzen and Spiegelhalter (1988). The connection between variable elimination and jtrees through the clusters of a decomposition tree representing a variable elimination calculation is proposed by Darwiche (2001). Message passing in junction trees follows either the Shafer-Shenoy propagation scheme (Shafer and Shenoy 1990), the Lauritzen-Spiegelhalter scheme, or the Hugin scheme (Jensen et al. 1990). Lepar and Shenoy (1998) compare the three propagation schemes, providing first insights into where Hugin can effectively trade space requirements for runtime improvements. Braun and Möller (2016) lift JT to the first-order level. Their LJT uses LVE as a subroutine and relies on the Shafer-Shenoy message passing scheme (Braun 2020). Ahmadi et al. (2013) lift loopy belief propagation, sending messages directly on a model graph.

To the best of our knowledge, neither a lifted division operator nor a lifted Hugin scheme exists, which allow for further efficiency gains for inference problems.

## 3 PRELIMINARIES

In this section, we recap the modelling formalism and lifted query answering with LVE and LJT. We assume familiarity with operators of relational algebra.

### 3.1 Parameterised Probabilistic Models

Parameterised probabilistic models use logical variables (logvars) to represent indistinguishable random variables (randvars) (Poole 2003). Given a set of randvars $\mathbf{R}$, a *parameterised randvar* (PRV) $A(X_1, \ldots, X_n), n \geq 0$ is a syntactical construct that uses a finite set of logvars $\mathbf{X} = \{X_1, \ldots, X_n\}$ to compactly encode a finite set of indistinguishable randvars $\mathbf{A} \subseteq \mathbf{R}$. If $n = 0$, the PRV is a propositional randvar. The *range* of a PRV $A$, written as $\mathcal{R}(A)$, denotes possible values of $A$. Every logvar $X_i \in \mathbf{X}$ has a domain $\mathcal{D}(X_i)$. We assume here and in the following that all logvars are standardised apart. A *valuation* $\mathbf{a}$ of $\mathbf{A}$ is an assignment of values to all $A \in \mathbf{A}$. The set of all valuations of a set or sequence $\mathbf{A}$ of PRVs is written $val(\mathbf{A})$. A *constraint* $(\mathcal{X}, C_{\mathcal{X}})$ restricts the logvars in a sequence $\mathcal{X} = (X_1, \ldots, X_n)$ to the values given in $C_{\mathcal{X}} \subseteq \times_{i=1}^{n} \mathcal{D}(X_i)$. The symbol $\top$ denotes $C_{\mathcal{X}} = \times_{i=1}^{n} \mathcal{D}(X_i)$ and may be omitted. A PRV $A$, with its logvars constrained to $C$, is written as $A_{|C}$. For a sequence of PRVs $\mathcal{A} = (A_1, \ldots, A_n)$, constrained to $C$, and a potential function $\phi : \times_{i=1}^{n} \mathcal{R}(A_i) \to \mathbb{R}_{\geq 0}$ which is not the null function, a *parfactor* $g = \phi(\mathcal{A})|_C$ models the interrelation between the $A_i$. Given an element $P$, such as a PRV, a parfactor, or sets thereof, the term $lv(P)$ denotes the logvars in $P$, the term $rv(P)$ denotes the PRVs in $P$ with their constraints, and the term $gr(P_{|C})$ represents the set of ground instances of $P$, w. r. t. to constraint $C$. A set of parfactors $G = \{g_i\}_{i=1}^{n}$ is a *parameterised probabilistic model*. The semantics of a parameterised model is given in terms of its full joint distribution

$$P_G = \frac{1}{Z} \prod_{f \in gr(G)} f \tag{1}$$

with normalisation constant $Z$. A *query* in $G$ is a term $P(\mathbf{Q} \mid \mathbf{E})$ where $\mathbf{Q}$ is a set of grounded PRVs and $\mathbf{E}$ a set of valuations (evidence). Example 1 gives an intuition about such models.

**Example 1.** A model $G_{ex} = \{g_i\}_{i=0}^{3}$ models the interrelation between an epidemic *Epid*, health conditions *Sick*$(X)$, travel behaviours *Travel*$(X)$, and treatments *Treat*$(X, M)$ of persons $X$ and treatment methods $M$, governmental regulations *Reg*$(G)$, human-made disasters *Man*$(W)$, and natural disasters *Nat*$(D)$, with

$$g_0 = \phi_0(\mathit{Epid}, \mathit{Treat}(X,M), \mathit{Sick}(X))$$
$$g_1 = \phi_1(\mathit{Epid}, \mathit{Travel}(X), \mathit{Sick}(X))$$
$$g_2 = \phi_2(\mathit{Epid}, \mathit{Reg}(G))$$
$$g_3 = \phi_3(\mathit{Epid}, \mathit{Man}(W), \mathit{Nat}(D))$$

Figure 1 shows a graphical representation with PRVs as ellipses and parfactors as boxes, connected if a PRV is an input to a parfactor. A query for $G_{ex}$ is $P(\mathit{Epid} \mid \mathit{Sick}(\mathit{bob}) = \mathit{true})$, asking for the conditional distribution of *Epid* given that *Sick*(*bob*) is true.

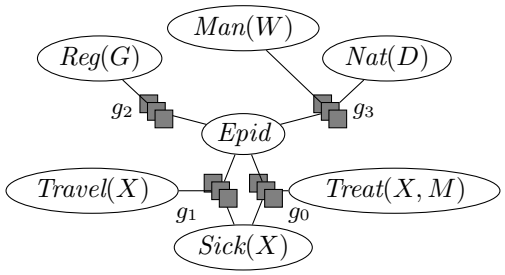


Figure 1: Parfactor graph of $G_{ex}$.

### 3.2 Lifted Query Answering

LVE and LJT answer queries for probability distributions. We briefly recap both approaches.

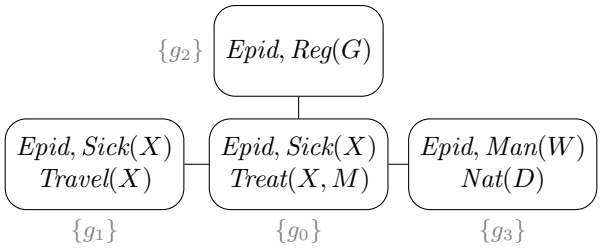


Figure 2: An FO-jtree for model $G_{ex}$ from Figure 1. The sets in grey denote the local models.

#### 3.2.1 Lifted Variable Elimination

LVE exploits symmetries leading to duplicate calculations. In essence, it computes a ground representative case and exponentiates its result for indistinguishable instances (lifted summing out). Taghipour, Davis, et al. (2013) implement LVE through an operator suite. Its main operator *sum-out* realises lifted summing out. An operator *absorb* handles evidence in a lifted way. The remaining operators (*count-convert*, *split*, *expand*, *count-normalise*, *multiply*, *ground-logvar*) aim at enabling a lifted summing out. All operators have pre- and postconditions to ensure computing a result equivalent to one computed on $gr(G)$. To answer a query $\mathbf{Q}$ in a model $G$ with evidence $\mathbf{E}$, LVE absorbs $\mathbf{E}$ and eliminates all non-query PRVs using the operators. For a new query, LVE starts over.

#### 3.2.2 Lifted Junction Tree Algorithm

For efficient repeated query answering, LJT decomposes a model into an FO-jtree $J = (\mathbf{V}, \mathbf{U})$ that consists of *parametric clusters* (parclusters) $\mathbf{C}_i \in \mathbf{V}$ of PRVs as nodes and a set of edges $\mathbf{U}$ between parclusters with at least one common PRV. Intuitively, parclusters are sets of PRVs that form a clique in the original parfactor graph. The set of parfactors that is covered by a parcluster is called the parcluster's *local model*. For details, see Braun and Möller (2016).

**Example 2.** Consider model $G_{ex}$ of Fig. 1. Its FO-jtree is given by Fig. 2.

LJT's main workflow to answer a set of queries $\{\mathbf{Q}_i\}_{i=1}^m$ given a model $G$ and evidence $\mathbf{E}$ is: (i) Construct an FO-jtree $J$ for $G$. (ii) Enter $\mathbf{E}$ into $J$. (iii) Pass messages in $J$. (iv) Compute answers for $\{\mathbf{Q}_i\}_{i=1}^m$. At the cost of some initial overhead (Steps 1 and 3), the tree decomposition takes away the need to perform LVE on the entire graph for every single query. The goal of message passing is to render any parcluster $\mathbf{C}_i$ independent of its neighbours $NB(\mathbf{C}_i)$ behind the *separators* $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$ it shares with its neighbouring parclusters $\mathbf{C}_j \in NB(\mathbf{C}_i)$. Then, each parcluster can answer queries about its PRVs locally, by eliminating all non query-variables from its local model and received messages. Message passing has the following procedure in general:

1. Parcluster $\mathbf{C}_i$, having received messages from all neighbours but $\mathbf{C}_j$, sends a message $M_{ij}$ to $\mathbf{C}_j$.
2. A parcluster, having received messages from all neighbours, sends messages to all its neighbours.

As Condition 1 is automatically true for leaf nodes, the scheme leads to two passes. In the first, inward pass, messages starting at the leaves transport information inward. At some point, Condition 2 becomes true for a central node in the tree, triggering an outward pass: Inner nodes that have collected all information send their messages back to the tree periphery.

For LJT, Braun lifts the Shafer-Shenoy scheme (Shafer and Shenoy 1990). In this scheme, clusters store all local parfactors and messages individually. The outbound message over separator $\mathbf{S}_{ij}$ is a set $m_{ij}$ of factors that is computed by summing out all non-separator PRVs from the submodel:

$$G_i' \equiv G_i \cup \bigcup_{\mathbf{C}_k \in NB(\mathbf{C}_i) \setminus \mathbf{C}_j} m_{ki}$$

where the union over $\mathbf{C}_k \in NB(\mathbf{C}_i) \setminus \mathbf{C}_j$ contains the already received messages from $\mathbf{C}_i$'s neighbours. As $\mathbf{C}_i$ receives more and more messages, the number of messages eventually increases to $O(d)$, where $d$ is the degree $d = \max_i |NB(\mathbf{C}_i)|$ of the FO-jtree. Shafer-Shenoy hands the set $G_i'$ to LVE which sums out all non-separator randvars. To enable summing out, LVE must first multiply the parfactors. These multiplications are performed for every message individually. However, the partaking parfactors are mostly the same in each message calculation (see Section 5.4.1). These multiplications can be saved. The propositional Hugin scheme (Jensen et al. 1990) trades space for runtime complexity and shows a polynomial runtime independent of the degree. Yet, in order to compute messages, a division operator is required. Introducing a lifted division operator as an enabler for lifted Hugin message passing is the first quest that this paper addresses.

## 4 PARFACTOR DIVISION

This section presents the lifted division operator and provides proof of its correctness and completeness with respect to known liftable classes. We conclude with an analysis of its runtime complexity. To get an intuition about division, we first look at its inverse operation of multiplication. As it is instructive to consider the ground case before lifting, we generalise propositional factor division and build the definition of lifted division on top of that foundation.

### 4.1 Multiplication

In propositional multiplication $f = f_1 \otimes f_2$, the potential of any valuation $\mathbf{a} \in val(\mathcal{A}_1 \cup \mathcal{A}_2)$ in the new factor $f = \phi(\mathcal{A}_1 \cup \mathcal{A}_2)$ is calculated as the product of the potentials of factors $f_1 = \phi_1(\mathcal{A}_1)$ and $f_2 = \phi_2(\mathcal{A}_2)$,

$$\phi(\mathbf{a}) = \phi_1(\pi_{\mathcal{A}_1}(\{\mathbf{a}\})) \cdot \phi_2(\pi_{\mathcal{A}_2}(\{\mathbf{a}\})) \tag{2}$$

The lifted multiplication operator takes two parfactors $g_1$ and $g_2$ and returns a new parfactor $g = g_1 \otimes g_2$ which represents the lifted product. A so-called *alignment*, i. e., a one-to-one renaming substitution, lets the operator know, which logvars between the parfactors correspond to each other. Formally, an alignment $\theta = \{\mathbf{Z}_1 \to \mathbf{Z}_2\}$ between parfactors $g_1 = \phi_1(\mathcal{A}_1)_{|C_1}$ and $g_2 = \phi(\mathcal{A}_2)_{|C_2}$ with $C_i = (\mathcal{X}_i, C_{\mathcal{X}_i})$ for $i \in \{1, 2\}$ is a mapping from $\mathbf{Z}_1 \subseteq lv(\mathcal{X}_1)$ to $\mathbf{Z}_2 \subseteq lv(\mathcal{X}_2)$ such that $(\pi_{\mathbf{Z}_1}(C_1))\theta = \pi_{\mathbf{Z}_2}(C_2)$ where $\pi$ is applied to the second component of constraint $C_i$.

In terms of groundings, the result of the lifted multiplication is equivalent to multiplying corresponding grounded factor potentials individually. Consider two parfactors $g_1 = \phi_1(\mathcal{A}_1)_{|C_1}$ and $g_2 = \phi_2(\mathcal{A}_2)_{|C_2}$ with an alignment $\theta = \{\mathbf{Z}_1 \to \mathbf{Z}_2\}$ where $\mathbf{Z}_i = lv(rv(g_1) \cap rv(g_2))$. If each grounding $f_1 \in gr(g_1)$ corresponds to exactly one $f_2 \in gr(g_2)$, then we have the same number of ground instances of $g_1$ as we have of $g_2$ and we can multiply two arbitrary representatives $f_1$ and $f_2$ and use the same result for all other groundings.

However, in general, the groundings of $g_1$ may interact with a number of groundings of $g_2$. In such a scenario, the factor potential resulting from the multiplication accounts for different numbers of groundings across parfactors and thereby requires scaling during multiplication. To scale potentials, the groundings of $g_1$ must be *count-normalised* w. r. t. the groundings of $g_2$ and vice versa, i. e., each grounding of one refers to the same number of groundings of the other. More precisely, count normalisation of a logvar $Y$ w. r. t. another logvar $Z$ ensures that for each possible grounding $z_i$ of $Z$, there is a natural number $r$ of groundings $y_{i_1}, \ldots, y_{i_r}$ of $Y$, and this number is the same for all $z_i \in \pi_{\mathbf{Z}}(C_{\mathcal{X}})$. Count normalization is usually achieved by splitting parfactors to establish disjoint or common constraints using one of the available lifted operators. For a constraint $C = (\mathcal{X}, C_{\mathcal{X}})$ with $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{Z} \subseteq (\mathbf{X} \setminus \mathbf{Y})$, the count function $\textsc{Count}_{\mathbf{Y}|\mathbf{Z}} : C_{\mathcal{X}} \to \mathbb{N}$ is defined for any tuple $t$ as

$$\textsc{Count}_{\mathbf{Y}|\mathbf{Z}}(t) = |\pi_{\mathbf{Y}}\left(C_{\mathcal{X}} \bowtie \pi_{\mathbf{Z}}(\{t\})\right)|$$

In other words, $\textsc{Count}_{\mathbf{Y}|\mathbf{Z}}(t)$ counts how many values in $\mathbf{Y}$ co-occur with values in $\mathbf{Z}$ in the tuple $t$ under constraint $C$. If $\mathbf{Y} = \emptyset$, then $\textsc{Count}_{\mathbf{Y}|\mathbf{Z}}(t) = 1$ by definition (Taghipour, Fierens, et al. 2013). The join operator $\bowtie$ acts on a constraint by joining on its set $C_{\mathcal{X}}$, so if $C_{\mathcal{X}} = \{(x_1, y_1), (x_2, y_1)\}$, then $C \bowtie \{(x_1)\} = \{(x_1, y_1)\}$. For $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{Z} \subseteq (\mathbf{X} \setminus \mathbf{Y})$, we call $\mathbf{Y}$ count-normalised with respect to $\mathbf{Z}$ in $C$ iff $\exists r \in \mathbb{N} : \forall t \in C_{\mathcal{X}} : \textsc{Count}_{\mathbf{Y}|\mathbf{Z}}(t) = r$ If such an $r$ exists, it is called the *conditional count* of $\mathbf{Y}$ given $\mathbf{Z}$ in $C$, written as $r = \textsc{Ncount}_{\mathbf{Y}|\mathbf{Z}}(C)$. As an example, consider the human parent relation. Every child $z \in \mathcal{D}(Z)$ has two biological parents $y_1, y_2 \in \mathcal{D}(Y)$, so logvar $Y$ is count-normalised w. r. t. $Z$ with $\textsc{Ncount}_{Y|Z}(\top) = 2$. The inverse is not automatically true, as parents can have more or fewer children.

So, prior to lifted multiplication $g_1 \otimes g_2$, it must hold that $r_i = \textsc{Ncount}_{lv(\mathcal{A}_i)\setminus\mathbf{Z}_i|\mathbf{Z}_i}(C_i)$ for $i = 1, 2$, i. e., the non-aligned logvars of each parfactor $g_i$ must be count-normalised w. r. t. the aligned logvars $\mathbf{Z}_i$ under constraints $C_i$. Then, the lifted product corresponding to the ground operations of Eq. (2) is

$$\phi(\mathbf{a}) = \phi_1^{1/r_2}(\mathbf{a}_1) \cdot \phi_2^{1/r_1}(\mathbf{a}_2).$$

### 4.2 Generalised Propositional Division

As a prelude to lifted division, we generalise propositional factor division (Darwiche 2009). The generalised division of two ground factors $f_1 = \phi_1(\mathcal{A}_1)$ and $f_2 = \phi_2(\mathcal{A}_2)$ over sequences $\mathcal{A}_2 \subseteq \mathcal{A}_1$ of randvars yields a new factor $f = \phi(\mathcal{A}_1)$ with factor potential

$$\phi(\mathbf{a}_1) = \begin{cases} \phi_1(\mathbf{a}_1)/\phi_2(\mathbf{a}_2) & \text{if } \phi_2(\mathbf{a}_2) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

for each valuation $\mathbf{a}_1 \in val(\mathcal{A}_1)$, $\mathbf{a}_2 \in val(\mathcal{A}_2)$. Note that $\mathcal{A}_2$ must be equal to or a subset of $\mathcal{A}_1$. Otherwise, the division is not well-defined. In the following, the division of any factor or parfactor $g_1$ by another factor or parfactor $g_2$ is denoted by $g_1 \oslash g_2$. Under certain conditions, factor division is the inverse of factor multiplication (see Section 4.4.1).

### 4.3 Lifted Division Operator

From the generalised factor division, we derive a lifted division of parfactors. Following the formalism of Taghipour (2013), the lifted division operator is shown as Operator 1. Given two parfactors $g_1 \in G$, $g_2 \notin G$ and an alignment $\theta = \{\mathbf{Z}_1 \to \mathbf{Z}_2\}$ with $\mathbf{Z}_i \subseteq lv(\mathcal{A}_i)$ for $i = 1, 2$ fulfilling the preconditions (1)–(3), the lifted division operator returns a new parfactor $g = g_1 \oslash g_2$ that corresponds to the generalised ground divisions. Precondition (1) is carried over from propositional factor division, namely the PRVs $\mathcal{A}_1$ of the divisor $g_2$ must be a subset of the PRVs $\mathcal{A}_2$ of the dividend. It ensures that the result is well-defined. Precondition (2) requires the non-aligned logvars to be count-

**Operator 1** DIVIDE

**Input:**
(1) $g_1 = \forall \mathbf{x}_1 \in \pi_{\mathbf{X}_1}(C_1) : \phi_1(\mathcal{A}_1)_{|C_1}, g_1 \in G$
(2) $g_2 = \forall \mathbf{x}_2 \in \pi_{\mathbf{X}_2}(C_2) : \phi_2(\mathcal{A}_2)_{|C_2}, g_2 \notin G$
(3) $\theta = \{\mathbf{Z}_1 \to \mathbf{Z}_2\}$, an alignment where $\mathbf{Z}_i \subseteq lv(\mathcal{A}_i)$ for $i = 1, 2$

**Preconditions:**
(1) $\mathcal{A}_2 \subseteq \mathcal{A}_1$
(2) $r = \text{NCOUNT}_{lv(\mathcal{A}_1)\setminus \mathbf{Z}_1 | \mathbf{Z}_1}(\pi_{\mathbf{X}_1}(C_1))$
(3) $\forall \mathbf{a}_2 \in val(\mathcal{A}_2) : \phi_2(\mathbf{a}_2) \neq 0$ or $g_1 = g_2 \otimes \prod_i g_i$

**Output:** $\forall \mathbf{x} \in \pi_{\mathbf{X}_1\theta \cup \mathbf{X}_2}(C) : \phi(\mathcal{A})_{|C}$ such that
$\mathcal{A} = \mathcal{A}_1\theta$
$C = C_1\theta \bowtie C_2$
for each $\mathbf{a} \in val(\mathcal{A})$,
$\mathbf{a}_1 = \pi_{\mathcal{A}_1\theta}(\{\mathbf{a}\})$,
$\mathbf{a}_2 = \pi_{\mathcal{A}_2}(\{\mathbf{a}\})$,

$$\phi(\mathbf{a}) = \begin{cases} \phi_1(\mathbf{a}_1)/\phi_2^{1/r}(\mathbf{a}_2) & \text{if } \phi_2(\mathbf{a}_2) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

**Postconditions:**
$G \equiv G \setminus \{g_1\} \cup \{\text{DIVIDE}(g_1, g_2, \theta)\} \cup \{g_2\}$

normalised w. r. t. the aligned logvars. Otherwise scaling can not be applied and lifted division is not possible. Precondition (3) states that one of the following is true: (a) there must be no zeros in the potential function of $g_2$, or (b) $g_2$ is part of the factorisation of $g_1$. In other words, $g_1$ must be expressible as a lifted product with $g_2$ as a partaking factor. Precondition (3) is ensured, for example, if $g_1$ is the result of a lifted multiplication of $g_2$ with other parfactors.

### 4.4 Analysis

Next, we prove some properties of Op. 1 that are important for the lifted Hugin scheme.

#### 4.4.1 Correctness

Correctness of Op. 1 guarantees that its postcondition holds given Preconditions (1), (2), and (3). To show correctness, we need the following lemma.

**Lemma 4.1.** *For parfactors $g_1, g_2$ fulfilling all prerequisites of Op. 1, it holds that*

$$g_1 = (g_1 \otimes g_2) \oslash g_2 \tag{4}$$

*Proof.* Rewriting Eq. (4) as follows

$$\phi_1(\mathbf{a}_1) = \frac{\phi_1^{1/r_2}(\mathbf{a}_1) \cdot \phi_2^{1/r_1}(\mathbf{a}_2)}{\phi_2^{1/r_1'}(\mathbf{a}_2)}.$$

where $r_1 = r_1'$ and $r_2 = 1$, enabling eliminating $\phi_2$, shows that Eq. (4) holds. □

**Theorem 4.1.** The lifted division operator is correct.

*Proof.* First, observe that for all models $G, G', G'' : G' \equiv G'' \Rightarrow G \cup G' \equiv G \cup G''$ (Taghipour, Fierens, et al. 2013). Now, let $G' \equiv G \setminus \{g_1\}$. It remains to show that $\{\text{DIVIDE}(g_1, g_2, \theta)\} \cup \{g_2\} \equiv \{g_1\}$, i. e., the union of the division result and the divisor must be equivalent to the original factor. Recall that the semantics is given by the full joint. With $g = \text{DIVIDE}(g_1, g_2, \theta)$, we have to show that the following are equivalent:

$$P_{\{g,g_2\}} = \frac{g \otimes g_2}{Z_1} \quad \text{and} \quad P_{\{g_1\}} = \frac{g_1}{Z_2}$$

With Lemma 4.1, it holds that if $g = g_1 \oslash g_2$, then $g \otimes g_2 = g_1$. Hence, the factorisation is $\prod_{g' \in \{g,g_2\}} g' = g \otimes g_2 = g_1$ with normalizing constants $Z_1 = Z_2$. □

#### 4.4.2 Complexity

The complexity of lifted division used within an FO-jtree depends on the number of parclusters as well as the largest intermediate parfactor size during calculation, i. e., the number of input to output mappings that the operator has to iterate through. Taghipour, Davis, et al. (2013) provide an upper bound for this size with the *node complexity*, as given by Definition 4.1. Braun (2020) use these results to express parcluster complexity. As any parfactor $g = \phi(\mathcal{A})_{|C}$ can be rewritten as parcluster $\mathbf{C} = \{A | A \in \mathcal{A}\}_{|C}$, node complexity can be used to characterize the complexity of individual parfactors. The *ground width* $w_g$ of an FO-jtree is the largest number of PRVs in any of the clusters. Analogously, the *counting width* $w_\#$ is the largest number of CRVs in any of the clusters.

**Definition 4.1.** The complexity of a set of PRVs $\mathbf{A}$ is

$$\mathcal{C}(\mathbf{A}) = \hat{r}^{w_g} \cdot \hat{n}_\#^{w_\# r_\#} \tag{5}$$

where $\hat{r}$ denotes the largest range size of any PRV, $\hat{r}_\#$ the largest range size in any CRV, $\hat{n}$ the largest domain size of any PRV, and $\hat{n}_\#$ the largest counted domain.

**Definition 4.2.** The complexity $\mathcal{C}(g)$ of parfactor $g = \phi(\mathcal{A})_{|C}$ is $\mathcal{C}(\mathbf{A}_{|C})$, $\mathbf{A} = \{A | A \in \mathcal{A}\}$.

Intuitively, these complexities provide an estimate of how many entries a table has for the potentials of all possible combinations of range values of the PRVs and CRVs. The term $\hat{r}^{w_g}$ is the size of a table that holds the potentials for all PRVs and the term $\hat{n}_\#^{w_\# r_\#}$ provides an upper bound for the range sizes of the $w_\#$ CRVs, as proposed by Taghipour, Davis, et al. (2013). In LVE, the same concept is used to describe the worst-case intermediate result, which most prominently influences LVE's runtime. With these preliminary considerations, we derive an expression for the runtime complexity of lifted division.

**Theorem 4.2.** Let $n_1 = \max_{Z_i \in \mathbf{Z}_1} |\mathcal{D}(Z_i)|$, the maximum number of instances covered by lifted division. The runtime complexity of lifted division $g_1 \oslash g_2$ is

$$O(\log(n_1) \cdot \mathcal{C}(g_1)) \tag{6}$$

*Proof.* From Precondition (1) in Op. 1, it must hold that $\mathcal{C}(g_1) \geq \mathcal{C}(g_2)$. A division of ground potential values is independent of domain or range sizes. Hence, the division of potentials requires at most $\mathcal{C}(g_1)$ operations. Scaling the resulting potential to the power of at most $n_1$ requires $O(\log(n_1))$. □

# 5 LIFTED HUGIN PROPAGATION

This section specifies a lifted Hugin message passing scheme for LJT, based on the definition of the lifted division operator. As a stepping stone to lifting Hugin, it identifies repeated operations in propositional Hugin given symmetries in the model. The elimination of these symmetries leads to the lifted Hugin scheme. This section concludes with an analysis and proof of lifted Hugin's runtime complexity.

## 5.1 Propositional Hugin Under Symmetries

Instead of multiplying local factors and received messages for each neighbour individually, as in the Shafer-Shenoy scheme, the Hugin scheme multiplies all local factors of a cluster $\mathbf{C}_i$ once, stored in a local Hugin factor $h_i$. Any incoming message is also multiplied into $h_i$ as well as stored separately. Outgoing messages are calculated by dividing the incoming message from the receiver out of the Hugin factor and summing out the non-separator randvars from the result.

Before lifting Hugin, it is instructive to look at propositional Hugin under symmetries. As an illustration, consider Fig. 3. Assume that the leaf nodes $\mathbf{C}_1$ are symmetric in that they share the same randvars and the same potential function for all $x_i, y_i$. At any cluster, the Hugin factor is initialised as the product of local factors, which are the same across all instances due to the symmetry. It follows that the Hugin factors $h_1$ are the same for all clusters $\mathbf{C}_1$. All inbound messages $m_{j0}$ are $\sum_{\mathbf{C}_1 \setminus \mathbf{S}_{j0}} h_1$. At the receiving cluster $\mathbf{C}_0$, the Hugin update is

$$h_0' = h_0 \otimes m_{10} \otimes m_{20} \otimes \cdots \otimes m_{n0}$$

As all $m_{j0}$ are identical, this expression reduces to

$$h_0' = h_0 \otimes m_{10}^n,$$

which essentially corresponds to a single lifted multiplication. So, lifting the Hugin initialisation and the inward pass is rather straightforward.

During the outward pass, shown in Fig. 3 by the arrows pointing away from $\mathbf{C}_0$, cluster $\mathbf{C}_0$ sends messages to a number of identical receiving clusters $\mathbf{C}_j$. It becomes clear that the outward messages $m_{0j} = \sum_{\mathbf{C}_0 \setminus \mathbf{S}_{j0}} (h_0 \oslash m_{j0})$ are the same for all $j$ if all of the identical inbound messages $m_{j0}$ from the leaf clusters are present. Lifting ensures this concurrency condition, as one lifted message covers all ground clusters. Therefore, it suffices to calculate the message $m_{0j}$ for one node $j$ and use the same result for all clusters.

After receiving $m_{0j}$, the Hugin factor in all of the leaf clusters is updated the same way: $h_j' = h_j \otimes m_{0j}$. Hence, all clusters $\mathbf{C}_1$ can be represented as one parcluster that sends and receives messages in a lifted manner. These considerations lead to a lifted Hugin message passing scheme.

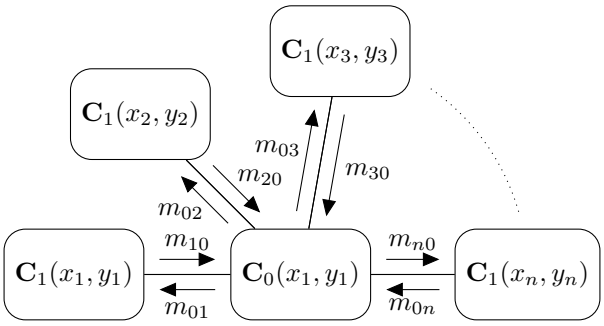


Figure 3: Message passing in a ground jtree with exposed symmetries. The notation $\mathbf{C}_i(x_j, y_j)$ represents the groundings obtained from $\mathbf{C}_i$ when constants $x_j$ and $y_j$ are used as arguments in the PRVs in $\mathbf{C}_i$.

## 5.2 Formal Specification

Formally, for each parcluster $\mathbf{C}_i$, lifted Hugin maintains a local parfactor $h_i$ which is initialised as the lifted product of all parfactors of the local model $G_i$,

$$h_i = \prod_{g \in G_i} g \tag{7}$$

For an incoming message $m_{ji}$, the Hugin factor is updated through lifted multiplication $h_i' = h_i \otimes m_{ji}$. After having received all required messages, according to the message passing conditions, parcluster $\mathbf{C}_i$ computes its outbound message to parcluster $\mathbf{C}_j$ in the following way: First, lifted Hugin divides the Hugin parfactor $h_i$ by the inbound message of parfactor $\mathbf{C}_j$, to remove the information already present at $\mathbf{C}_i$. This division leads to a new parfactor

$$g_{ij} = \begin{cases} h_i \oslash m_{ji} & \text{if } m_{ji} \text{ exists} \\ h_i & \text{otherwise} \end{cases} \tag{8}$$

If $\mathbf{C}_j$ is the only neighbour of $\mathbf{C}_i$ that has not yet sent its message to $\mathbf{C}_i$, the message $m_{ij}$ can nevertheless be computed. In this case, $m_{ij}$ is not yet contained in the local factor $h_i$ and the message parfactor simplifies to $g_{ij} = h_i$. Afterwards, the outbound message $m_{ij}$ is obtained by eliminating all non-separator PRVs in $g_{ij}$.

### 5.3 Complexity Results

Let $n = \max\{\hat{n}, \hat{n}_{\#}\}$, where $\hat{n}$ and $\hat{n}_{\#}$ are the largest PRV and CRV domain sizes as in Section 4.4.2. Given an FO-jtree $J = (\mathbf{V}, \mathbf{U})$ with parclusters $\mathbf{C}_i \in \mathbf{V}$, its complexity is $\mathcal{C}(J) = \max_{\mathbf{C}_i \in \mathbf{V}}(\mathcal{C}(\mathbf{C}_i))$.

**Theorem 5.1.** Given an FO-jtree $J = (\mathbf{V}, \mathbf{U})$, lifted Hugin propagation in $J$ has a runtime complexity of

$$O(|\mathbf{V}| \cdot \log(n) \cdot \mathcal{C}(J)) \tag{9}$$

*Proof.* As described above, one complete cycle of Hugin propagation consists of the initial calculation of the Hugin factors, and the two passes of message passing. We look at the initialisation first.

At each parcluster in an FO-jtree, the Hugin factor $h_i$ is initialised by multiplying all parfactors $g \in G_i$ of local model $G_i$ (Eq. (7)). Together with scaling the result in $\log(n)$, lifted multiplication has a complexity of $O(\log(n) \cdot \mathcal{C}(h_i))$. Observe that the complexity of the Hugin factor of cluster $\mathbf{C}_i$ is the complexity of the cluster itself, i. e., $\mathcal{C}(h_i) = \mathcal{C}(\mathbf{C}_i)$. Thus, the Hugin initialisation for any parcluster $\mathbf{C}_i$ has a complexity of $O(\log(n) \cdot \mathcal{C}(\mathbf{C}_i))$. As $\mathcal{C}(J) = \max_{\mathbf{C}_i \in \mathbf{C}} \mathcal{C}(\mathbf{C}_i)$ and there are $|\mathbf{V}|$ parclusters in an FO-jtree, the complexity for Hugin initialisation in $J = (\mathbf{V}, \mathbf{U})$ is given by Eq. (9).

The message passes are two executions of the same procedure, namely, the calculation of a message by lifted division, followed by summing out, and updating the Hugin factor through lifted multiplication. This happens at every cluster in the FO-jtree. Lifted division of $g_{ij} = h_i \oslash m_{ji}$ has a complexity of $O(\log(n) \cdot \mathcal{C}(h_i))$, which does not add to the asymptotic complexity. Summing out the non-separator PRVs from the message parfactor $g_{ij}$ takes at most $O(\log(n) \cdot \mathcal{C}(g_{ij}))$ operations (Taghipour, Davis, et al. 2013), when performed using LVE. As $\mathcal{C}(g_{ij}) \leq \mathcal{C}(J)$, summing-out does not add to the asymptotic complexity either. Observe that any lifted multiplication $h_i \otimes m_{ji}$ is also bounded by $O(\log(n) \cdot \mathcal{C}(h_i))$. Consequently, Eq. (9) holds for the entire message passing procedure. □

### 5.4 Discussion

This section compares the lifted Hugin and the lifted Shafer-Shenoy propagation and describes how to use lifted Hugin for adaptive inference.

#### 5.4.1 Comparison with Shafer-Shenoy

To compare the lifted Hugin scheme to the lifted Shafer-Shenoy scheme, recall the FO-jtree from Fig. 2. One complete procedure of lifted Hugin message passing consists of the following steps: First, local Hugin parfactors are initialised. As every local model consists of exactly one parfactor, namely $G_i = \{g_i\}$ we have $h_i = g_i$ for $i \in \{0, 1, 2, 3\}$. Next, the inbound messages are prepared. From Eq. (8), the inbound messages are trivially computed by eliminating all PRVs but $\mathbf{S}_{0i} = \{\mathit{Epid}\}$ from intermediate parfactors $g_{i0} = h_i$. In Shafer-Shenoy, the set of local parfactors making up the local model are handed over to LVE, which in turn handles the lifted multiplication and summing out. In the Hugin scheme, only the Hugin factor is handed over to LVE, where only the summing out operation remains to be done. In our example, there is no difference between Hugin and Shafer-Shenoy in terms of the number of lifted operations. However, when the local model consists of a multitude of parfactors, LVE has more work to do per neighbour under the Shafer-Shenoy propagation scheme. More differences emerge during the outward pass. To compute the intermediate parfactor $g_{01}$ for message $m_{01}$, Shafer-Shenoy would perform the following lifted multiplications

$$g_{01} = g_0 \otimes m_{20} \otimes m_{30} \otimes m_{40}$$

before eliminating $\mathit{Treat}(X, M)$ and $\mathit{Sick}(X)$ from the result $g_{01}$. This factorisation is computed for each message, with one of the partaking parfactors exchanged. E.g., for $g_{02}$, $m_{10}$ replaces $m_{20}$. Hugin on the other hand, requires computing a similar product only once:

$$h_0' = h_0 \otimes m_{10} \otimes m_{20} \otimes m_{30} \otimes m_{40} \tag{10}$$

Then, for all subsequent messages $i \in \{1, 2, 3\}$, the computation is reduced to $g_{0i} = h_0 \oslash m_{i0}$. With Eq. (6), we have already established that the complexity of any such lifted division is $O(\log(n) \cdot \mathcal{C}(\mathbf{C}))$, where $n$ is the largest domain size covered by the operation. Taghipour, Davis, et al. (2013) show the same complexity result for lifted multiplication. Observe that $n_1$ is constant for any parcluster and $\mathcal{C}(\mathbf{C})$ is determined by the range values and domain sizes of the PRVs involved. Despite the larger factorisation in Eq. (10), the size of intermediate results, and thereby the node complexity is not increased by the operation. This is due to the fact that the messages contain only PRVs from the separators. So, the node complexities influencing the lifted operations are the same for both propagation schemes. However, as argued above, in terms of the degree $d$ of the FO-jtree, the Shafer-Shenoy scheme depends on $d$ per message, i. e., $O(d)$, whereas for lifted Hugin, only one the sum-out operations are needed for every message, which does not depend on $d$, i. e., $O(1)$.

These runtime improvements however have a cost in terms of space complexity. We have seen that Shafer-Shenoy stores the parfactors as well as all messages individually. So its space complexity is given by the complexity of the largest parfactor in the model $O(\max_{g \in G}(\mathcal{C}(g)))$. As Hugin requires the space to store the Hugin factors of each parcluster, its space requirements are $O(\mathcal{C}(J))$, which is exponential in the ground width, the counting width and the largest PRV range size of the model, as can be seen from Eq. (5).

### 5.4.2 Adaptive Inference

Adaptive inference aims to reuse as many calculations as possible when changes, e.g., in evidence, occur, which an FO-jtree with its messages easily supports. For message passing, we only want to update those messages affected by changes. The lifted division operator plays an important role in an adaptive message passing as it allows for information to be removed from any parfactor by dividing out the respective messages (or old parfactors). When new parfactors are added to the model or parfactors are altered, it suffices to divide out the obsolete messages of the Hugin factors and multiply them with the new message, before triggering a partial message pass, instead of running the entire procedure all over.

For example, assume that in the FO-jtree from Fig. 2, parfactor $g_1$ is altered. To update the model, $\mathbf{C}_1$ needs to resend a message $m'_{10}$ to $\mathbf{C}_0$, where $\mathbf{C}_0$ merely divides out the old message $m_{10}$, which is now obsolete, of its local Hugin parfactor $h_i$. Finally, $\mathbf{C}_0$ updates its local Hugin factor to $h'_0 = h_0 \otimes m_{10}$ with the new message. It then sends out outbound messages to its neighbours which receive the new information.

## 6 EVALUATION

We have evaluated the lifted Hugin propagation scheme in a number of settings based on the LJT implementation available[1]. Our empirical results are shown in the following figures. All runtimes are given in milliseconds. All values are averages over five runs. We decided not to add confidence intervals, as they are so small, they complicate reading the plots while providing only little added value.

Figure 4 shows a comparison of the runtimes of ground and lifted variants of the Shafer-Shenoy and Hugin message passing schemes. The model used is $G_{ex}$ as shown in Fig. 1, where the domain size for $X$ is varied between 1 and $10^3$. In this scenario, we do not expect Hugin and Shafer-Shenoy to exhibit major differences in either their propositional or lifted variant. The results show the effect of lifting however: The runtimes for lifted schemes are polynomial in domain sizes.

1. https://www.ifis.uni-luebeck.de/index.php?id=518

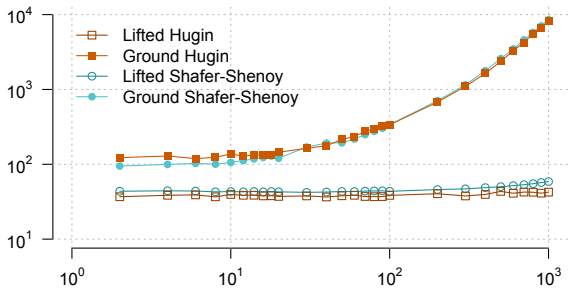


Figure 4: Influence of the domain size on runtime of different propagation schemes. Times [ms] shown are total runtimes, consisting of tree construction, message passing, and query answering. Model used is $G_{ex}$ from Fig. 1, which is a typical model for lifted probabilistic inference.

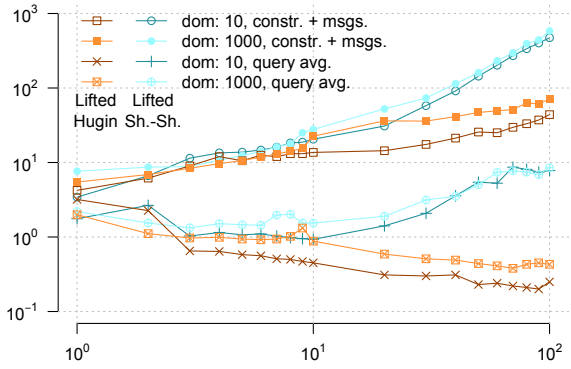


Figure 5: Influence of the degree of the FO-jtree on runtime [ms] of different propagation schemes in a graph with a star topology.

Figure 5 shows the runtimes of lifted Hugin compared to lifted Shafer-Shenoy for FO-jtrees with increasing degree. The underlying FO-jtrees have a star topology, consisting of a central parcluster with a number $n \in [1, 100]$ of leaf nodes. Each node has one parfactor with one 1-ary PRV with its own logvar per parcluster and one separator PRV. The values are shown for domain sizes of 10 and $10^3$ respectively. The model is queried once per PRV. More parclusters lead to more queries and a larger FO-jtree, which results in increased runtimes. Thereby, for comparability, the average time per query is also shown for Hugin and Shafer-Shenoy, next to FO-jtree construction and message passing time. As expected, the average time per query in the Hugin scheme decreases with the number of queries issued, whereas for Shafer-Shenoy, the influence of the number of neighbours leads to a steeper increase in runtime. Of course, this scenario is somewhat artificial, as FO-jtrees almost never exhibit such a perfect star topology. However, the gathered data

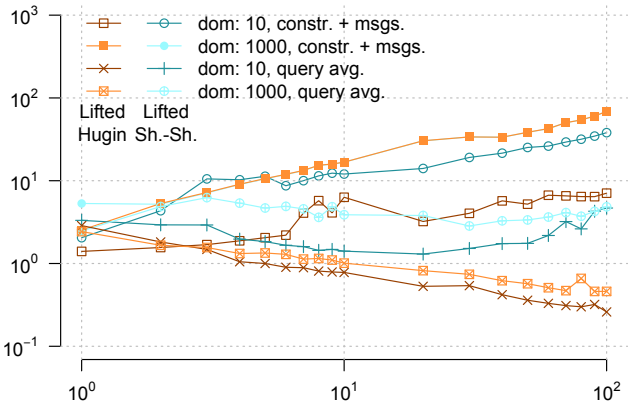


Figure 6: Influence of the domain size on runtime of different propagation schemes. Times shown are total runtimes [ms], consisting of tree construction, message passing and query answering.

nicely illustrate the main differences between Hugin and Shafer-Shenoy message propagation.

For reference, consider Fig. 6. It represents a similar setup, but with a chain graph, instead of a star graph. A chain graph favours Shafer-Shenoy propagation to the maximum extent. However, the data illustrates that while Hugin provides no asymptotic advantage over Shafer-Shenoy, its runtime is also not asymptotically worse than that of Shafer-Shenoy.

Overall, the evaluation again showcases the advantages of lifted inference for models that exhibit symmetries and exhibits that a lifted Hugin propagation easily outperforms lifted Shafer-Shenoy propagation in settings that favour Hugin while not performing asymptotically worse than Shafer-Shenoy in settings that favour Shafer-Shenoy. Therefore, the lifted Hugin propagation presents itself as a true alternative to the lifted Shafer-Shenoy propagation in terms of efficiency.

## 7 CONCLUSIONS

With the definition of the lifted division operator and an analysis of its main properties, we fill a missing link in the suite of lifted operators. Lifted division enables a lifted Hugin message passing scheme, for which we provide a specification and a theoretical analysis, revealing a runtime complexity that is independent of the degree of the FO-jtree. With its topology-agnostic runtime that outperforms lifted Shafer-Shenoy in a number of scenarios, the lifted Hugin scheme is a further step towards real-time query answering.

Another application of the Hugin scheme lies in dynamic inference, such as in the Lifted Dynamic Junction Tree (Gehrke et al. 2018), which we expect to benefit from the additional adaptability through lifted division as well as from Hugin's degree-independent runtime. Beyond that, the ability to split parfactors through lifted division could possibly contribute to the reification of parfactors for approximate inference.

## References

Niepert, Mathias, and Guy Van den Broeck. 2014. "Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference." In *AAAI-14 Proceedings of the 28th AAAI Conference on Artificial Intelligence,* 2467–2475. AAAI Press.

Braun, Tanya, and Ralf Möller. 2016. "Lifted Junction Tree Algorithm." In *Proceedings of KI 2016: Advances in Artificial Intelligence,* 30–42. Springer.

Lauritzen, S. L., and D. J. Spiegelhalter. 1988. "Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems." *Journal of the Royal Statistical Society. Series B (Methodological)* 50 (2): 157–224. ISSN: 00359246.

Gehrke, Marcel, Tanya Braun, Ralf Möller, Alexander Waschkau, Christoph Strumann, and Jost Steinhäuser. 2019. "Lifted Maximum Expected Utility." In *Artificial Intelligence in Health,* 131–141. Springer.

Shafer, Glenn, and Prakash Shenoy. 1990. "Probability propagation." *Ann Math Artif Intell* 2 (March): 327–351. https://doi.org/10.1007/BF01531015.

Jensen, Finn V., Steffen L. Lauritzen, and Kristian G. Olesen. 1990. "Bayesian updating in causal probabilistic networks by local computations" [in English]. *Computational Statistics Quarterly* 4:269–282. ISSN: 0723-712X.

Poole, David. 2003. "First-order probabilistic inference." In *Proc. 18th International Joint Conf. on Artificial Intelligence,* 985–991.

Taghipour, Nima, Jesse Davis, and Hendrik Blockeel. 2013. "First-order Decomposition Trees." In *Advances in Neural Information Processing Systems,* edited by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, vol. 26. Curran Associates, Inc.

Zhang, Nevin Lianwen, and David Poole. 1994. "A simple approach to Bayesian network computations."

Milch, Brian, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. 2008. "Lifted Probabilistic Inference with Counting Formulas." In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2,* 1062–1068. AAAI'08. Chicago, Illinois: AAAI Press. ISBN: 9781577353683.

Salvo Braz, Rodrigo de, Eyal Amir, and Dan Roth. 2005. "Lifted First-Order Probabilistic Inference." In *Proceedings of the 19th International Joint Conference on Artificial Intelligence,* 1319–1325. IJCAI'05. Edinburgh, Scotland: Morgan Kaufmann Publishers Inc.

Salvo Braz, Rodrigo de. 2007. "Lifted First-Order Probabilistic Inference." PhD diss., University of Illinois at Urbana-Champaign.

Friedman, Tal, and Guy Van den Broeck. 2020. "Symbolic Querying of Vector Spaces: Probabilistic Databases Meets Relational Embeddings." *arXiv:2002.10029 [cs]* (June 27, 2020). Accessed November 1, 2020.

Taghipour, Nima, Daan Fierens, Jesse Davis, and Hendrik Blockeel. 2013. "Lifted Variable Elimination: Decoupling the Operators from the Constraint Language." *Journal of Artificial Intelligence Research* 47 (1): 393–439.

Darwiche, Adnan. 2001. "Recursive conditioning." Tradeoffs under Bounded Resources, *Artificial Intelligence* 126 (1): 5–41. ISSN: 0004-3702.

Lepar, Vasilica, and Prakash P. Shenoy. 1998. "A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions." In *UAI-98 Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence,* 328–337. AUAI Press.

Braun, Tanya. 2020. "Rescued from a Sea of Queries: Exact Inference in Probabilistic Relational Models." PhD diss., University of Lübeck.

Ahmadi, Babak, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. 2013. "Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training." *Machine Learning* 92 (1): 91–132.

Darwiche, Adnan. 2009. *Modeling and reasoning with Bayesian networks.* Cambridge; New York: Cambridge University Press. ISBN: 978-0-521-88438-9.

Taghipour, Nima. 2013. "Lifted Probabilistic Inference by Variable Elimination." PhD diss., KU Leuven.

Gehrke, Marcel, Tanya Braun, and Ralf Möller. 2018. "Lifted Dynamic Junction Tree Algorithm." In *Proceedings of the International Conference on Conceptual Structures,* 55–69. Springer.