

# Updating in Local Computation

Schneuwly Cesar<sup>1</sup>

**Abstract.** Local computation on covering join trees provides a solution for query answering in several different fields, such as relational databases, belief functions, constraint satisfaction, Gaussian potentials and many more. The algebraic structure behind is a generic framework for information processing known as valuation algebras [5].

In this paper we discuss how new information pieces can be added to the old information, i.e. how to use former computed results to answer the queries in the new situation. A task which will be referred to as updating. The domains of the new information pieces are possibly not covered by any node of the join tree. Since the construction of a new covering join tree may be computationally expensive and makes it harder or even impossible to reuse already available results, we introduce several methods to modify join trees locally. We will see that this enables updating approaches for all well-known architectures for local computation like Shenoy-Shafer, Lauritzen-Spiegelhalter and HUGIN.

## 1 INTRODUCTION

In valuation algebras information is represented by valuations. We normally start with a set of valuations  $\{\psi_1, \psi_2, \dots, \psi_n\}$ , a *knowledge base*. The problem of computing a marginal  $\phi^{\downarrow s}$  of the combined information  $\phi = \psi_1 \otimes \dots \otimes \psi_n$  is called *projection problem*; sometimes referred to as *query answering* with *query*  $s$ . Usually, several queries are of interest.

There are many different approaches based on join trees solving this task reasonably efficient, such as the Shenoy-Shafer [10], the Lauritzen-Spiegelhalter [6] and the HUGIN architectures [4], all described in terms of valuation algebras in [5]. A generalization to covering join trees is discussed in [9]. A common feature of these methods is that all operations take place within the domains of the nodes of the join tree. They are therefore often referred to as local computation methods. At the end we obtain the marginals of  $\phi$  to the labels of the nodes in the underlying join tree  $JT$ .

An important problem arises when a new piece of information  $\psi'$  arrives and has to be added. We then say that the overall information  $\phi$  is *updated* by  $\psi'$ . The new problem is to answer the already given and fixed queries relative to  $\phi \otimes \psi'$  and  $JT$ . Updating presupposes a former run of a local computation. The challenge is to reuse as many information as possible from this first run. There are two cases: If there is a node in  $JT$  which covers the domain of  $\psi'$ , we say that  $\psi'$  is *compatible* with  $JT$ . Otherwise, we call it *incompatible* [1] or *implicit* [13]. Since there are situations where several information pieces  $\mathcal{I} = \{\iota_1, \iota_2, \dots, \iota_m\}$  arrive simultaneously, we intend to treat

such a set  $\mathcal{I}$  directly at once. Every information piece in  $\mathcal{I}$  is then either compatible or not. A mixture is allowed.

One could imagine other forms of updating, like replacing or removing information pieces. Another issue is query answering to arbitrary domains after local computation, which has already been discussed in the domain of relational databases [15] and for belief functions [14]. The latter uses a *removal* operator [12] which is closely related to division in *regular* valuation algebras [5]. It turns out that these forms of updating can be transported to the level of valuation algebras as special cases of the theory developed here. But this goes beyond the scope of the article and we refer to [8] for the details.

For the case of incompatible updates we first discuss in section 2 several tree modification methods. The idea is to modify small parts of  $JT$  making the new information pieces  $\mathcal{I}$  compatible. A first approach is presented in subsection 2.1 which is in fact a generalization of tree modification methods introduced in [13, 1]. It considers just the structure of the join tree and not the valuations lying on it. This indicates a possible optimization which is discussed in subsection 2.2.

An important commonality of the different tree modification methods serves to introduce in section 3 so-called *updating bases*, a generic notion which will be used subsequently for updating procedures. We will see that new inward propagation phases on a modified tree  $JT'$  for the different local computation architectures can partially be skipped, if former results are accessible. Updating consists therefore of the *completion* of this new run followed by an outward propagation phase. We discuss further updating in algebras with partial marginalization where updating gets more involved.

## 2 TREE MODIFICATION METHODS

### 2.1 Modifying Regarding the Structure

We start with a join tree  $JT = (V, E, \lambda, D)$ , where  $V$  denote the nodes,  $E$  the edges,  $\lambda : V \rightarrow D$  the labeling function and  $D$  a powerset of variables. A set  $s' \in D$  such that there is no  $i \in V$  with  $s' \subseteq \lambda(i)$  is called *incompatible* [1] or *implicit* [13]. Consider a set  $s \subseteq D$  of implicit domains (relative to  $JT$ ) to be covered,

$$s = \{s_1, s_2, \dots, s_n\}, \quad s' = s_1 \cup s_2 \cup \dots \cup s_n.$$

We begin by determining a minimal set of *connected* nodes  $X$  such that the union of their labels covers  $s' \cap t$  where  $t = \bigcup_{j \in V} \lambda(j)$ . The next theorem (proven in [14]) guarantees that this can be done efficiently.

**Theorem 1** *The set  $X$  can be chosen in linear time such that it is minimal in the sense, that if we remove a node from it, it either does no more cover  $s' \cap t$ , or will make the vertices in  $X$  disconnected.*

After such a set  $X$  has been found, we remove the edges between the nodes in  $X$ . The join tree  $JT$  breaks into several fragments  $JT_i$ , each of which contains a single node  $i \in X$ . This is the starting point.

<sup>1</sup> University of Fribourg, Switzerland, <http://diuf.unifr.ch/tcs>  
Research supported by grant No. 200020-109510 of the Swiss National Foundation for Research.

Next, we build a new join tree  $JT^*$  which covers the domains in  $s$  together with the labels of the nodes in  $X$ .<sup>2</sup> The challenge is to connect properly every  $JT_i$  to  $JT^*$  such that the resulting tree  $JT'$  satisfies the running intersection property. This is described in algorithm 1, which is referred to as *generalized modification algorithm*. The crucial points are 5.(c) and 5.(d) where the edges of  $JT'$  are determined. For short, 5.(c) keeps all edges of  $JT$  which are not among the nodes in  $X$  and all edges of  $JT^*$ . This yields  $JT^*$  together with the  $JT_i$  for  $i \in X$  apart. 5.(d) connects each  $JT_i$  via  $i$  to a node in  $JT^*$ . The following theorem (proven in [8]) shows that  $JT'$  is a join tree.

**Theorem 2** *The resulting labeled graph  $JT' = (V', E', \lambda', D)$  of the generalized tree modification algorithm is a join tree and for every  $s_i \in s$  there is a node  $n'_i \in V'$  which covers  $s_i$ .*

**Algorithm 1: Generalized Tree Modification**

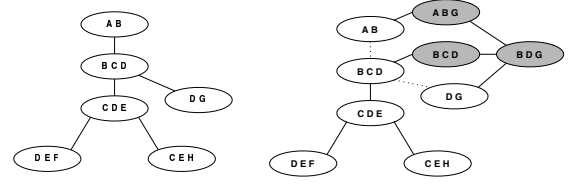
→ **input:**  $JT = (V, E, \lambda, D)$  and a set  $s = \{s_1, s_2, \dots, s_n\}$  of incompatible domains  
← **output:** A covering join tree  $JT'$  for all labels  $\lambda(i)$  and all domains  $s_j \in s$

1. Let  $t = \bigcup_{j \in V} \lambda(j)$  and  $s' = s_1 \cup s_2 \cup \dots \cup s_n$
2. Find a minimal set  $X$  of connected nodes with  $s' \cap t \subseteq \bigcup_{i \in X} \lambda(i)$
3. Determine the set of domains  $b' = s \cup \{\lambda(k) : k \in X\}$
4. Compute a covering join tree  $JT^* = (V^*, E^*, \lambda^*, D)$  for the domains  $b'$  without free variables (avoid name clashes of  $V^*$  and  $V$ ; “without free variables” means that each variable in the node labels must also occur somewhere in the domains contained in  $b'$ )
5. Generate the labeled graph  $JT' = (V', E', \lambda', D)$  according to
  - (a) Set  $V' := V \cup V^*$
  - (b) Set
$$\lambda'(i) := \begin{cases} \lambda(i), & \text{if } i \in V; \\ \lambda^*(i), & \text{if } i \in V^* \end{cases}$$
  - (c)  $E' := E^* \cup \{\{n, n'\} \in E : n \notin X\}$
  - (d) Find for every node  $i \in X$  a node  $j \in V^*$  with  $\lambda(i) \subseteq \lambda'(j)$  and add the edge  $\{i, j\}$  to  $E'$

**Example 1** *Given the join tree depicted on the left in figure 1. We are interested in covering the implicit domains  $s = \{\{A, G\}, \{B, G\}\}$ . For this purpose we determine  $X = \{\{A, B\}, \{B, C, D\}, \{D, G\}\}$  and get  $b' = \{\{A, B\}, \{B, G\}, \{B, C, D\}, \{D, G\}, \{A, G\}\}$ . A possible join tree  $JT'$  is depicted on the right in figure 1.*

If  $JT^*$  consists of a single node which covers  $s'$ , we essentially get Xu’s tree modification algorithm introduced in [13]. Similar generalizations of Xu’s method are discussed in [13, 1]. The first is based on the fusion algorithm for join tree construction [11] and takes, after having cut the edges among the nodes in  $X$ , the  $JT_j$  as an intermediate situation of fusion and completes the tree. The latter is close to the present approach, but “subsumes” several nodes during execution which makes the resulting tree harder to use for updating.

<sup>2</sup> for example with the fusion algorithm of Shenoy [11]. But not every join tree is equally suitable for computations, since the biggest label should have minimal cardinality. Finding such a covering join tree is known to be NP-hard [2]. An overview of heuristics can be found in [7] and a comparison of the different methods in [3].



**Figure 1.** Covering the implicit domains  $\{A, G\}$  and  $\{B, G\}$ .

It is worth noting that we can apply the generalized tree modification algorithm  $n$ -times consecutively for every  $s_i \in s$  instead of once for the whole set  $s$ . The question about the pros and cons cannot be answered here. It depends heavily on the domains in  $s$ , the structure of  $JT$  and one can imagine a combination of both. However, this question is not that important for our purposes. We will see later in this paper that if we are interested in updating, every modification of the tree provokes at least a complete new outward propagation phase which is computationally expensive. Therefore, we prefer one modification step for all domains in  $s$ .

## 2.2 Domain Based Approach

A local computation base for a projection problem

$$(\phi_1 \otimes \phi_2 \otimes \dots \otimes \phi_n)^{\perp x_j}$$

with queries  $x_j$  is a quintuple  $(V, E, \lambda, D, a)$ , where  $(V, E, \lambda, D)$  is a join tree which covers the factors above (together with the queries) and  $a : \{\phi_1, \dots, \phi_n\} \rightarrow V$  is a mapping which assigns uniquely every valuation  $\phi_i$  to a node in  $V$ . Every node  $i \in V$  has therefore a valuation  $\psi_i = \bigotimes_{a(j)=i} \phi_j$  assigned. Nodes  $k$  such that there is no  $l$  with  $a(l) = k$  get an (adjoined) identity valuation  $e$  assigned; this is indeed possible, see [9]. The  $\psi_i$  are thus either  $e$ , a single  $\phi_j$  or the result of the combination of several  $\phi_k$ . The overall idea of this subsection is that the domains of the valuations  $\psi_i$  do probably not entirely fill the label  $\lambda(i)$ . A further optimization can be achieved.

**Definition 1** *Let  $Y \subseteq V$  be a connected subset of nodes in  $JT$ . We call the nodes in  $Y$  having at least one neighbor in  $JT$ , which is not contained in  $Y$ , border nodes and denote the set of border nodes according to a set  $Y$  with  $bd(Y)$ .*

Consider  $Y$  as a minimal set of connected nodes in  $JT$  covering a domain  $s'$  (in the sense of theorem 1). For every  $i \in bd(Y)$

- introduce a new node  $j$  with label  $\lambda(i)$ ;
- remove for every  $k \in \text{neighbors}(i)$  with  $k \notin Y$  the edge  $\{k, i\}$  from  $E$ , but add  $\{k, j\}$ ;
- connect  $j$  to  $i$ , i.e. add an edge  $\{i, j\}$ ;
- assign  $e$  to the node  $j$ .

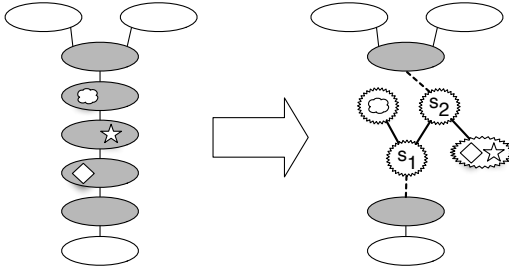
This procedure will be referred to as *expander*. Let then  $X$  contain  $Y$  together with the newly introduced nodes during the expander algorithm. Note that  $X$  is connected. The border nodes  $bd(X)$  are not used to cover real valuations. We may shrink their labels, that is, for every variable  $A \in \lambda(i)$ ,  $i \in bd(X)$ , we check if it can be removed such that the running intersection remains satisfied. The whole procedure (including shrinking) will be referred to as *normalization* on  $Y$  given  $s'$  relative to the assignment mapping  $a$ .

**Definition 2** Given a minimal subset  $X \subseteq V$  of connected nodes for covering a set  $s'$  of variables. We say that  $JT$  is normalized on  $X$  given  $s'$  relative to an assignment mapping  $a$ , if

- every border node  $i \in bd(X)$  has  $e$  assigned;
- removing any variable  $A$  in the label of any node in  $bd(X)$  violates the running intersection property.

We have seen above that if we normalize  $JT$  on  $Y$  given  $s'$  relative to  $a$ , then the result is a normalized tree on  $X$  (defined as above) given  $s'$  relative to the adapted assignment mapping.

We are now able to present the overall idea of the new modification algorithm. Let  $s = \{s_1, s_2, \dots, s_n\}$  be a set containing implicit domains to be covered with  $s' = s_1 \cup s_2 \cup \dots \cup s_n$ . Assume we have a join tree  $JT^\circ$  which is normalized on  $X$  given  $s' \cap t$  relative to an assignment mapping  $a$ , where  $t$  denotes the set of variables already used in the join tree. In  $JT^\circ$  we use the set  $X$  as the part which is concerned by the modification. Instead of cutting out the edges among the nodes in  $X$  as we did in the former approaches, we remove those in  $(X - bd(X))$  completely. Several connected parts  $JT_i^\circ$  of  $JT^\circ$  remain, all identifiable by a node  $i \in bd(X)$ . These “identifier” nodes serve later as connecting nodes to a newly built join tree  $JT^*$ , which is supposed to cover the domains in  $s$ , the labels  $\lambda^\circ(i)$  for every  $i \in bd(X)$  and the domains of the valuations assigned to nodes  $(X - bd(X))$  removed just before. Figure 2 depicts the idea. The crucial point is the connection of  $JT^*$  to the subtrees  $JT_j^\circ$  without violating the the running intersection property. Note that deleting the nodes implies that not all labels reappear. Further, the labels of nodes in  $X$  could represent a desired query. Until now we neglected this completely and concentrated on the domains of the factors only. It is clear that all desired queries  $x_i$  have to be covered in the resulting tree too. For simplicity, we assume throughout this subsection that they are represented by newly introduced valuations  $e_{x_i}$  with domains  $x_i$  and are added to the factorization as *placeholders*. After the modification process, they are replaced by the identity valuation  $e$ .



**Figure 2.** On the left-hand side we assume a normalized join tree, where the set  $X$  of nodes is grey-shaded. The idea is to remove the nodes in  $(X - bd(X))$  and build a completely new covering join tree for the domains of interest. This tree is then connected to the parts of the original join tree which arose by the remove operation.

The new tree modification approach, which will be referred to as *domain based tree modification*, is described in detail in algorithm 2.

**Theorem 3** At the end of the domain based tree modification algorithm, the graph  $JT' = (V', E', \lambda', D)$  is a covering join tree for the domains in  $s$  and for the domains of the factors  $\phi_1 \otimes \phi_2 \otimes \dots \otimes \phi_n$ .

*Proof.* See [8]. □

**Algorithm 2: Domain Based Tree Modification**

<b>input:</b>	Local computation base $(V, E, \lambda, D, a)$ for $\phi_1 \otimes \dots \otimes \phi_n$ together with a related set of implicit domains $s = \{s_1, s_2, \dots, s_n\}$
<b>output:</b>	Covering join tree for the factors $\phi_i$ and the domains in $s$

1. Let  $t = \bigcup_{i \in V} \lambda(i)$  and  $s' = s_1 \cup s_2 \cup \dots \cup s_n$ .
2. Find a minimal set  $Y$  of connected nodes (see theorem 1) such that  $s' \cap t \subseteq \bigcup_{i \in Y} \lambda(i)$ .
3. Normalize the given join tree on  $Y$  given  $s' \cap t$  relative to  $a$  and get a new join tree  $JT^\circ = (V^\circ, E^\circ, \lambda^\circ, D)$  together with a new assignment mapping  $a^\circ$ .
4. Let  $X \subseteq V^\circ$  contain  $Y$  and all new nodes in  $JT^\circ$  relative to  $JT$  introduced in the last step for normalization, i.e. the ones in  $(V^\circ - V)$ . Formally  $X = Y \cup \{i : i \in (V^\circ - V)\}$ .
5. Remove the nodes  $(X - bd(X))$  from  $JT^\circ$ . This yields several subtrees  $JT_i^\circ$  which can be identified by a unique node  $i \in bd(X)$ .
6. Let  $b' = s \cup \bigcup_{j: \exists i \in X, a^\circ(j)=i} \{d(\phi_j)\} \cup \bigcup_{i \in bd(X)} \{\lambda^\circ(i)\}$ .
7. Generate a covering join tree  $JT^*$  based on  $b'$  without free variables (avoid name clashes of  $V^*$  with  $V^\circ$ ).
8. Find for every subtree  $JT_i^\circ$  a node  $k$  in  $JT^*$  with  $\lambda^\circ(i) \subseteq \lambda^*(k)$  and add an edge  $(i, k)$  to  $E^\circ$ .
9. This connects the  $JT_i^\circ$  and  $JT^*$  together. The resulting labeled graph is the final  $JT' = (V', E', \lambda', D)$ .

**Example 2** Given the join tree depicted in figure 3 (a) and let  $s = \{\{E, F\}, \{D, F\}\}$  and consider the factorization

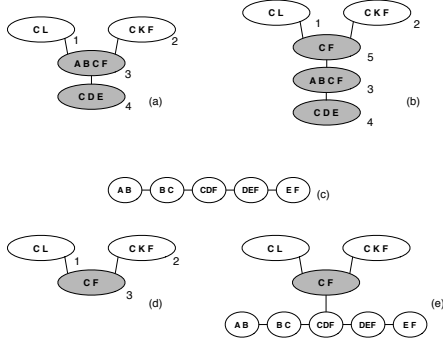
$$\phi = \phi_1 \otimes \phi_2 \otimes \dots \otimes \phi_7$$

with  $d(\phi_1) = \{C, L\}$ ,  $d(\phi_2) = \{C, F, K\}$ ,  $d(\phi_3) = \{C, F\}$ ,  $d(\phi_4) = \{B, C\}$ ,  $d(\phi_5) = \{A, B\}$ ,  $d(\phi_6) = \{C, D\}$ ,  $d(\phi_7) = \{D, E\}$  and  $a(1) = 1$ ,  $a(2) = 2$ ,  $a(3) = a(4) = a(5) = 3$ ,  $a(6) = a(7) = 4$ . We follow the algorithm step by step:

1. The set  $t = \{A, B, C, D, E, F, K, L\}$  and  $s' = \{E, F, D\}$ .
2.  $Y = \{3, 4\}$ .
3. The normalized join tree  $JT^\circ$  is depicted in fig. 3 (b).
4. With  $JT^\circ$  we get  $X = \{3, 4, 5\}$ . Note that  $bd(X) = \{3\}$ .
5. See figure 3 (d). The single subtree  $JT_3$  remains in this example.
6.  $b' = \{\{C, F\}, \{B, C\}, \{A, B\}, \{C, D\}, \{D, E\}, \{E, F\}, \{D, F\}\}$ .
7. A possible covering join tree without free variables is in fig. 3 (c).
8. The only possible choice is the node in  $JT^*$  with label  $\{C, D, F\}$ .  
If we add an edge between the two nodes we get the final  $JT'$  (see figure 3 (e)).

*It is interesting that there is no node of cardinality 4 in the resulting  $JT'$ .*

The idea of the domain based modification algorithm is first mentioned in [1], but not worked out. There it is further limited to the case  $|s| = 1$ . Since the new  $JT^*$  is based on the variables really needed, we can say that  $JT'$  is locally optimal on  $JT^*$ . Taking the result of the last example we see that it is even possible in some cases that the biggest label of  $JT'$  has a smaller cardinality than the biggest of  $JT$ . So  $JT'$  does in this case not only cover the domains in  $s$  in addition, but is even more suitable for computational purposes.



**Figure 3.** The new tree modification approach takes the domains of the valuations lying on the nodes into account (see example 2).

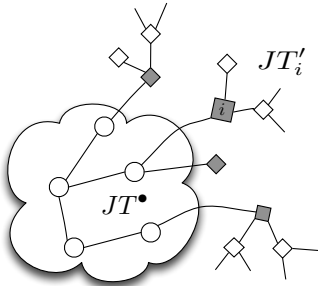
### 3 ADDING NEW INFORMATION

Consider a covering join tree  $JT = (V, E, \lambda, D)$  for an arbitrary knowledge base  $\{\phi_1, \phi_2, \dots, \phi_n\}$  which can be used for answering queries  $x_i$ . We assume that  $\psi_i, 1 \leq i \leq m$ , is the valuation assigned to node  $i \in V$  by an assignment mapping  $a$ , such that

$$\phi = \bigotimes_{i=1}^n \phi_i = \bigotimes_{i=1}^m \psi_i.$$

Let  $\mathcal{I}$  be a set of valuations  $\mathcal{I} = \{\iota_1, \iota_2, \dots, \iota_p\}$  and  $s$  be the set of domains  $s = \{d(\iota) : \iota \in \mathcal{I}; d(\iota) \text{ is incompatible with } JT\}$ . This section examines how the queries  $x_i$  according to the updated  $\phi' = \phi \otimes \bigotimes_{\iota \in \mathcal{I}} \iota$  knowledge base can (efficiently) be computed using a previous local computation. The challenge is to answer queries by reusing as many results as possible of the former run.

Let  $JT^\bullet$  be an arbitrary subtree of a join tree  $JT'$ . This induces a decomposition (see figure 4): A node  $j$  is either in  $V^\bullet$  or in  $(V' - V^\bullet)$ . But  $(V' - V^\bullet)$  consists in general of several subtrees, where



**Figure 4.** Every subtree  $JT^\bullet$  of a tree  $JT'$  induces a decomposition into several parts  $JT_i^prime$ , each of which connected via a node  $i$  to  $JT^\bullet$ . The set  $X$  of connecting nodes is grey shaded.

each can be identified by a (connecting) node  $i$  which is a neighbor of a node in  $JT^\bullet$ . We refer to the subtrees corresponding to  $(V' - V^\bullet)$  as  $JT_i^prime$ . Let then  $X$  denote the set of all connecting nodes.  $JT^\circ$  is finally the tree consisting of  $JT^\bullet$  together with the nodes in  $X$ .

In a first step of updating we modify the given join tree  $JT$  such that the implicit domains  $s$  become compatible. Xu's tree modification, the generalized and the domain based approaches have one property in common: Subtrees of  $JT$  reappear in the resulting  $JT'$

and are connected to a newly build join tree (see the algorithms). Now let  $JT^\bullet$  be a subtree of  $JT'$  such that

- $V^\bullet$  contains all new nodes introduced during tree modification;
- for every  $\iota \in \mathcal{I}$  there is a node  $j \in V^\bullet$  which covers  $d(\iota)$ .

Note that we may choose a minimal  $JT^\bullet$  in the sense that removing any node of  $JT^\bullet$  does either no more cover the valuations  $\mathcal{I}$ , contains no more all new nodes introduced for the modification or is disconnected. This can be done by successively check every node. Based on the subtree  $JT^\bullet$ , we may determine  $X$ ,  $JT_i^prime$  and  $JT^\circ$  as described above. We call the quintuple  $UB = (JT, JT', JT^\bullet, JT_i^prime, JT^\circ)$  an *updating base*. If  $JT^\bullet$  is minimal, we add the prefix *normalized* to the name. Note that the  $JT_i^prime$  in an updating base are subtrees of  $JT$ ; they are identical to  $JT_i$ . Updating bases allow to introduce updating *procedures* for the different architectures. This is the topic of the following subsections.

### 3.1 Updating Procedures

#### 3.1.1 In the Shenoy-Shafer Architecture

In the Shenoy-Shafer architecture updating is rather simple. We begin by determining an assignment mapping for the factors of  $\phi'$  on  $JT'$ : Every node  $j$  in a subtree  $JT_i^prime$  of  $JT'$  gets again the valuation  $\psi_j$  assigned. Note that  $\psi_j$  is either a single factor  $\phi_l$ , the result of the combination of several such factors or the identity element  $e$ . It is important to remember that in the domain based modification approach not all nodes of  $JT$  reappear and therefore not all information pieces  $\phi_k$  are distributed by this process. However, it is an immediate consequence of the domain based modification approach that the remaining factors  $\phi_l$  to be distributed are covered by the nodes in  $JT^\bullet$ . And so are the new information pieces in  $\mathcal{I}$ . We distribute both freely to covering nodes in  $JT^\bullet$ .

Next we start a new inward propagation phase towards a node in  $JT^\bullet$ . But we may skip the subtrees  $JT_i^prime$ . Indeed, inward propagation produces messages identical to the first run until the connecting nodes are reached. These messages are already available, since the Shenoy-Shafer architecture explicitly requires a caching. An additional outward propagation phase completes the updating process.

We conclude that inward propagation can partially be skipped during updating procedures if the Shenoy-Shafer architectures has been applied earlier. A result which can partly be transported to the HUGIN architecture, see the following subsection.

#### 3.1.2 In Algebras with Division

Assume now a regular valuation algebra  $(\Phi, D)$  with a well-defined division operator [5]. Projection problems can then be solved using the HUGIN or the Lauritzen-Spiegelhalter (LSA) architectures. Unfortunately we can not carry out updating as for the Shenoy-Shafer architecture, since the messages sent among the nodes are not cached. Nevertheless, a comparable procedure as for Shenoy-Shafer exist by exploiting an important property based on division: The valuation  $\phi = \psi_1 \otimes \dots \otimes \psi_m \in \Phi$  with  $\psi_i \in \Phi$  also factorizes in

$$\phi = \bigotimes_{i \in V} \phi^{\downarrow \lambda(i)} \otimes \bigotimes_{j \in S} (\phi^{\downarrow \sigma(j)})^{-1}, \quad (1)$$

if  $S$  denotes the set of separator nodes in  $JT$  and  $\sigma$  their labels,<sup>3</sup> see [5, 9]. The factors above fit on the nodes of  $JT$ . A possible

<sup>3</sup> if  $j \in S$  is the separator between  $i$  and  $ch(i)$ , then  $\sigma(j) = \lambda(i) \cap \lambda(ch(i))$

assignment mapping  $a$  which presupposes a choice of an arbitrary root node  $r$  is given in the algorithm 3. Note that if the valuations  $\eta$  are determined by this algorithm we get by equation (1) indeed  $\phi = \bigotimes_{t \in V} \eta_t$ .

**Algorithm 3: Stabilizer**

**input:** A factorization of  $\phi$  like in equation (1) relative to a given join tree  $JT = (V, E, \lambda, D)$  rooted at  $r$   
**output:** An assignment of the factors to covering nodes in  $JT$

1. Assign the valuations  $\eta_i = \phi^{\downarrow \lambda(i)}$  to every leaf node  $i$
2. Let  $s(k)$  denote separator nodes between  $k$  and its parents  $pa(k)$
3. Assign to every non-leaf  $j$  the valuation

$$\eta_j = \phi^{\downarrow \lambda(j)} \otimes \bigotimes_{q \in s(j)} (\phi^{\downarrow \sigma(q)})^{-1}$$

**Lemma 1** *At the end of a new inward propagation phase on  $JT$  towards  $r$  using the assignments determined by algorithm 3, every node  $i$  contains  $\phi^{\downarrow \lambda(i)}$  and every separator  $j$  stores  $\phi^{\downarrow \sigma(j)}$ .*

*Proof.* See [8] □

It is important to note that we require the valuations  $\phi^{\downarrow \sigma(k)}$  on every separator  $k \in S$ . Remember that they are available at the end of the HUGIN architecture. The LSA provokes however the computation of the additional marginals  $\phi^{\downarrow \sigma(k)}$  though.<sup>4</sup> And if they are available (either newly computed or conserved beforehand), we may proceed as for HUGIN. This is actually recommended, since in HUGIN divisions take place on nodes with smaller labels. We skip therefore the LSA.

Let us investigate updating. First we concentrate on the generalized modification algorithm in order to get a modified  $JT'$ .  $X$  should be the set consisting of the connecting nodes  $i$  of the subtrees  $JT_i$ . Clearly,  $X$  is a subset of the nodes in  $JT$ . We distribute then the valuations  $\eta$  on  $JT$  according to the stabilizer algorithm using a root node  $r \in X$ . The above lemma 1 implies that new runs of inward propagation on the original  $JT$  can be skipped, since we know already the results. The generalized modification algorithm guarantees that the assigned valuations  $\eta$  to nodes in  $JT$  can fully be transported towards the subtrees  $JT_i$  of  $JT'$ . If the new information pieces  $\mathcal{I}$  are distributed to covering nodes in  $JT^\bullet$  (this is indeed possible), we easily see that the  $JT_i$  can be skipped during inward propagation on the modified  $JT'$  towards a node in  $V^\bullet$  until the connecting nodes  $i$  are reached. Hence, updating consists of inward propagation beginning at  $JT^\circ$  followed by an outward propagation phase on the whole  $JT'$ .

A word of warning: We start updating where every node  $j \in JT_i$ ,  $j \neq i$ , stores a valuation of the form  $\phi^{\downarrow \lambda(j)}$ . It might be conjectured that this remains correct for the connecting nodes  $i \in X$ . This is only incidentally the case because the edges between nodes in  $X$  were cut in order to get  $JT'$ . As a consequence, every parent  $j \in X$  of a node  $i \in X$  in the original tree  $JT$  does not send the message  $\mu_{j \rightarrow i} = \phi^{\downarrow \lambda(i) \cap \lambda(j)}$  towards  $i$  during inward propagation on  $JT'$ , since the edge  $(j, i)$  does no more exist. In other words, the inverse  $(\phi^{\downarrow \lambda(i) \cap \lambda(j)})^{-1}$  does not vanish in the node store of  $i$  as it would on

<sup>4</sup> if they are not conserved during outward propagation (which is not explicitly required by the algorithm)

$JT$ . We conclude formally that every  $i \in X$ , i.e. every leaf of  $JT^\circ$ , stores at the beginning of inward propagation on  $JT^\circ$

$$\phi^{\downarrow \lambda(i)} \otimes \bigotimes_{k, i: (k, i) \in E, (k, i) \notin E'} (\phi^{\downarrow \lambda(i) \cap \lambda(k)})^{-1},$$

where the edges  $(k, i)$  are directed<sup>5</sup> according to the root node  $r$  chosen above for the stabilizer algorithm.

If the domain based tree modification was applied, then the assignments by the stabilizer algorithm can only partially be transported from  $JT$  towards  $JT_i$ . This is due to the fact that not all nodes of  $JT$  reappear. So there are probably some factors of (1) which are not assigned to nodes of the  $JT_i$  and which have still to be distributed. But in difference to the new information pieces  $\mathcal{I}$ , these factors do typically not fit on  $JT^\bullet$ .

**Example 3** *Reconsider example 2. The factorization is*

$$\phi = \phi^{\downarrow \{C, L\}} \otimes \phi^{\downarrow \{C, K, F\}} \otimes \phi^{\downarrow \{A, B, C, F\}} \otimes \phi^{\downarrow \{C, D, E\}} \otimes (\phi^{\downarrow \{C\}})^{-1} \otimes (\phi^{\downarrow \{C, F\}})^{-1} \otimes (\phi^{\downarrow \{C\}})^{-1},$$

but neither  $\phi^{\downarrow \{A, B, C, F\}}$ , nor  $\phi^{\downarrow \{C, D, E\}}$  are covered by  $JT'$ .

We avoid therefore domain based tree modifications in architectures with division.

We conclude that updating in the HUGIN architecture works for the generalized modification algorithm, but usually not in the more general domain based case. The stabilizer algorithm determines assignments for the  $JT_i$  such that they can be skipped during a new inward propagation phase. The new information pieces can be freely distributed to covering nodes in  $JT^\bullet$ . A totally new outward propagation phase completes the updating process.

Next we assume a separative valuation algebra  $(\Phi, D)$  where division can be carried out only in an embedding algebra  $(\Phi^*, D)$  and is only partially defined, see [5, 9]. It is by no means guaranteed that all marginals are well-defined during the new inward propagation phase towards a node in  $JT^\bullet$  (nevertheless they are at least on the subtrees  $JT_i$ , see above). It is however well-known that if inward propagation works, then so does outward propagation [9].

We impose that the updated valuation  $\phi'$  has to be an element of  $\Phi$ , but the valuations in  $\mathcal{I}$  may be only in the embedding algebra  $(\Phi^*, D)$ . Hence there is a big difference between the addition of  $p = |\mathcal{I}|$  information pieces consecutively (piece by piece) and the full set  $\mathcal{I}$  at once. For the first case, there must be a permutation  $\pi$  such that the sequence  $\iota_{\pi(1)}, \iota_{\pi(2)}, \dots, \iota_{\pi(p)}$  of the valuations in  $\mathcal{I}$  fulfills  $\phi \otimes \iota_{\pi(1)} \in \Phi$ ,  $\phi \otimes \iota_{\pi(1)} \otimes \iota_{\pi(2)} \in \Phi$  and so on until  $\phi \otimes \iota_{\pi(1)} \otimes \iota_{\pi(2)} \otimes \dots \otimes \iota_{\pi(p)} \in \Phi$ . Respecting the sequence  $\pi(1), \pi(2), \dots, \pi(p)$  in the updating procedure, we can proceed as usual but add single information pieces. It is interesting to note that if the  $\iota_i$  are all compatible, then inward propagation always works since for every step a single node  $r$  is concerned in the updating (and can thus be reached in the new inward propagation phase). No new marginals need to exist. Remember that adding  $p$  single pieces apart requires  $p$  outward propagation phases. This is computationally expensive. The situation gets more involved if we add all compatible valuations simultaneously. An example is depicted in the figure 5. If we add  $\iota_1, \iota_2$  and  $\iota_3$  at the same time, it is not clear if the messages  $\mu_{2 \rightarrow 1}, \mu_{1 \rightarrow 3}, \mu_{3 \rightarrow 1}$  or  $\mu_{1 \rightarrow 2}$  exist.

But even if the  $\mathcal{I}$  are incompatible with  $JT$ , the existence of a permutation  $\pi$  as described above may guarantee a successful updating. For this purpose reconsider Xu's tree modification approach because it has the following interesting consequence proven in [8]:

<sup>5</sup>  $k$  is the parent of  $i$

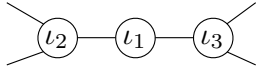


Figure 5. Adding information pieces simultaneously.

**Theorem 4** *If we add a single, incompatible information piece  $\iota \in \Phi^*$  to  $\phi$  such that  $\iota \otimes \phi \in \Phi$  and  $JT$  in the used  $UB$  is modified according to Xu’s tree modification algorithm, then all marginals exist during the generic updating procedure.*

### 3.1.3 In Idempotent Algebras

We stated above that we can not use factorization (1) if  $JT' = (V', E', \lambda', D)$  results from the domain based tree modification approach.  $JT'$  is normally not a covering join tree for these factors. But if we concentrate on idempotent valuation algebras, which are regular too, we find yet another factorization of  $\phi$ . It consists of a mixture of the computed  $\phi^{\downarrow \lambda(i)}$  and the original factors  $\phi_j$ .

**Lemma 2** *Let  $Y$  denote a set of connected nodes in  $JT$ . The valuation  $\phi \in \Phi$  factorizes into  $\phi = \bigotimes_{i \in V, i \notin Y} \phi^{\downarrow \lambda(i)} \otimes \bigotimes_{i \in Y \subseteq V} \phi_i$ , if  $(\Phi, D)$  is an idempotent valuation algebra.*

*Proof.* See [8]. □

Consider a (normalized) updating base  $UB = (JT, JT', JT^\bullet, JT'_i, JT^\circ)$ . The nodes  $k \in V'$  such that  $k \notin V^\bullet$  coincide with the reappearing nodes in the subtrees  $JT_i$  and we may reassign the factors  $\phi^{\downarrow \lambda(j)}$  to each of them. Let then  $Y$  contain the nodes in  $(V - V')$ , i.e. the ones removed by the tree modification process (if any). Since  $Y$  is a connected set of nodes, we know by the above lemma that  $\phi = \bigotimes_{i \in V, i \notin Y} \phi^{\downarrow \lambda(i)} \otimes \bigotimes_{i \in Y \subseteq V} \phi_i$ , hence the factors  $\phi_k$  with  $k \in Y$  have still to be distributed. But they are covered by the nodes in  $JT^\bullet$ .<sup>6</sup> We distribute them freely, together with the new information pieces in  $\mathcal{I}$ , to covering nodes in  $V^\bullet$ . Using these assignments, a new inward propagation phase can again be skipped on the  $JT_i$  since the messages vanish due to idempotency. Indeed, if a node stores  $\phi^{\downarrow \lambda(i)}$  its message sent to its neighbor  $j$  is  $\mu_{i \rightarrow j} = \phi^{\downarrow \lambda(i) \cap \lambda(j)}$  and we get at node  $j$ ,

$$\phi^{\downarrow \lambda(j)} \otimes \mu_{i \rightarrow j} = \phi^{\downarrow \lambda(j)}.$$

Updating in idempotent algebras consists therefore, as for the Shenoy-Shafer and for the HUGIN architectures, of a new inward propagation phase beginning at the nodes in  $JT^\circ$  followed by an outward propagation phase.

## 4 CONCLUSION

Updating and query answering are important tasks for the treatment of information. Once a join tree is fixed and a local computation architecture executed, information processing leads to these two operations. We introduced in this paper several different tree modification methods for avoiding a complete new join tree construction if incompatible information pieces are added. Of particular interest is then the generic notion of an updating base which proves sufficient for updating procedures. We have seen that for all well-known architecture

for local computation updating is (efficiently) possible since the departing inference problem does not cause computational load on the unmodified parts of the join tree. Former results can be reused.

The presented theory is based on already established techniques of both, tree modification and local computation. This guarantees that the presented algorithms can be applied in many different research fields.

## ACKNOWLEDGEMENTS

I would like to thank Prof. J. Kohlas, Dr. N. Lehmann and the anonymous referee for their valuable comments.

Research supported by grant No. 200020-109510 of the Swiss National Foundation for Research.

## REFERENCES

- [1] B. Anrig, *Probabilistic Model-Based Diagnostics*, Ph.D. dissertation, Institute of Informatics, University of Fribourg, 2000.
- [2] S. Arnborg, D. Corneil, and A. Proskurowski, ‘Complexity of finding embeddings in a k-tree’, *SIAM J. of Algebraic and Discrete Methods*, **38**, 277–284, (1987).
- [3] A. Cano and S. Moral, ‘Heuristic algorithms for the triangulation of graphs’, in *Proceedings of the Fifth IPMU Conference*, eds., R.R. Yager Bouchon-Meunier and L.A. Zadeh, pp. 166–171. Springer., (1995).
- [4] F.V. Jensen, S.L. Lautitzen, and K.G. Olesen, ‘Bayesian updating in causal probabilistic networks by local computation’, *Computational Statistics Quarterly*, **4**, 269–282, (1990).
- [5] J. Kohlas, *Information Algebras: Generic Structures for Inference*, Springer-Verlag, 2003.
- [6] S. L. Lauritzen and D. J. Spiegelhalter, ‘Local computations with probabilities on graphical structures and their application to expert systems’, *J. Royal Statist. Soc. B*, **50**, 157–224, (1988).
- [7] N. Lehmann, *Argumentation System and Belief Functions*, Ph.D. dissertation, Department of Informatics, University of Fribourg, 2001.
- [8] C. Schneuwly and J. Kohlas, ‘Local computation in covering join trees, part #2, updating in local computation’, Technical Report 06-08, Department of Informatics, University of Fribourg, (2006).
- [9] C. Schneuwly, M. Pouly, and J. Kohlas, ‘Local computation in covering join trees’, Technical Report 04-16, Department of Informatics, University of Fribourg, (2004).
- [10] P. P. Shenoy and G. Shafer, ‘Axioms for probability and belief-function propagation’, in *Uncertainty in Artificial Intelligence 4*, eds., Ross D. Shachter, Tod S. Levitt, Laveen N. Kanal, and John F. Lemmer, volume 9 of *Machine intelligence and pattern recognition*, pp. 169–198, Amsterdam, (1990). Elsevier.
- [11] P.P. Shenoy, ‘Valuation-based systems: A framework for managing uncertainty in expert systems’, in *Fuzzy Logic for the Management of Uncertainty*, eds., L.A. Zadeh and J. Kacprzyk, 83–104, John Wiley & Sons, (1992).
- [12] P.P. Shenoy, ‘Using dempster-shafer’s belief function theory in expert systems’, in *Advances in The Dempster-Shafer Theory of Evidence*, eds., J. Kacprzyk R.R. Yager and M. Fedrizzi, 395–414, John Wiley & Sons, (1994).
- [13] H. Xu, ‘Computing marginals for arbitrary subsets from marginal representation in markov trees’, *Artificial Intelligence*, **74**(1), 177–189, (1995).
- [14] H. Xu, *Uncertain Reasoning and Decision Analysis using Belief Functions in the Valuation-based Systems*, Ph.D. dissertation, Faculty of Applied Sciences, Universite libre de Bruxelles, Belgium., 1995.
- [15] M. Yannakakis, ‘Computing the minimum fill-in is np-complete’, *SIAM J. of Algebraic and Discrete Methods*, **2**, 77–79, (1981).

<sup>6</sup> this is trivially guaranteed by the tree modification algorithms