Linköping Electronic Articles in
Computer and Information Science
Vol. (): nr

Dynamic Propagation with Hierarchical Junction Trees

Roberto O. Puch
Jim Q. Smith

Department of Statistics
University of Warwick
Coventry, UK

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis///

Published on , by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in Computer and Information Science, Vol. (): nr .
http://www.ep.liu.se/ea/cis///. , .

This URL will also contain a link to the author's home page.

Abstract

Motivated by applications in the development of decision support software for emergency planning under uncertainty, we introduce a new framework for probability propagation. The hierarchical junction tree (HJT) provides a platform for forecasting with Bayesian networks (BNs) for dynamic models with a heterogeneous evolution. In this setting new descendant nodes need to be added sequentially to the BN whilst nodes earlier in the BN are sequentially learned. Interest focuses on the calculation of distributions of the most recently added nodes given the observations so far. Thus probability propagation is restricted to calculation of distributions of a subset of the terminal nodes given a subset of their ancestors.

The junction tree provides a framework for computing distributions of any set of variables given any other set of variables. In the dynamic setting this generality requires on-line triangulation: known to be computationally expensive. The HJT preserves the time consistent edge direction of the BN. As a consequence, the HJT is constructed without conditional independence loss when the ancestral graph of the conditioning variables is decomposable. If this is not the case, only the ancestral graph of the conditioning variables needs to be triangulated.

1 Introduction

Dawid [3], Jensen [5] and Cowell *et al.*[2] describe an algorithm which takes a Bayesian network (BN) and transforms it into a junction tree (JT). This JT then takes the conditional independence statements encoded in the BN and uses them to construct a framework for fast probability propagation algorithms. This structure enables the computation of probabilities of a random vector of interest, subsequently called target variables, given the learned variables whose values are now known, subsequently called conditioning variables. These algorithms have the advantage of being always valid whatever the set of target and conditioning variables.

However for BNs that do not have a decomposable structure, this level of generality can be achieved only by ignoring some of the conditional independence in the generating BN. The types of conditional independence statements that need to be ignored are those involving variables listed earlier in the DAG and having a dependence relationship with "collider variables", a term defined in the next section, occurring later in the BN. The BN then needs to be modified by the operations of moralisation and triangulation before a JT can be drawn, see [4, 2].

There have been recent important articles which address this loss of efficiency. Madsen & Jensen [11] noted that for the JT to cover any kind of propagation task it needs to loose the direction of the BN. They proposed the Lazy Propagation methodology to recover some of the directionality while keeping the generality of the JT. They also noted that the generality of the JT also required the loss of some conditional independence. Again, to keep the generality of the graph, they recover some of the conditional independence by keeping the potentials on the nodes of the JT factorised.

Kjærulff [10] also noted the trade off between loss of conditional independence and generality of the JT for any kind of propagation task and proposed the Nested JTs method to recover some of the conditional independence lost during the construction of the JT. He suggested that JT representations be constructed for some of the nodes of the JT that would be suitable for the direction of the propagation and the factorisation of the potential of that clique. It is important to note that Kjærulff [10] introduced the idea of JTs having nodes represented by JTs. This feature is one of the components of the HJT.

In dynamic models where the primary interest is in forecasting, the level of generality that the JT provides is often not required. Examples of this setting arise from the development of decision support systems for emergency planning under uncertainty [14, 15]. In decision support systems in the case of a nuclear accident, counter-measures cannot affect the past contamination levels but only future received radiation doses. In addition the information that is usually received is on contamination levels in the past. Although forecasting is a less complex computational task, in this type of problems forecasting is required on-line and as soon as possible so that counter-measures are implemented. In contrast, smoothing is done off-line.

In most dynamic settings, it is natural to construct the BN so that variables appearing earlier in the BN tend to be learned before the variables appearing later in the network. In this setting we also have new descendant variables added with time, and we forecast the future in the light of observing the past. Thus it is typical for the target variables to be descendants of the conditioning variables. Furthermore the use of JT algorithms in settings where new descendant nodes are continuously added to the net has some inherent problems. First, the standard JT algorithms, that is, the JT with the Hugin architecture [1], provide no facility to move variables out of the system as they are learned because this is not required for static networks. So, the number of variables in the system/cliques in the tree steadily increase. Lazy propagation and Nested JTs inherit this feature as they act on a standard JT. Second, as new variables arrive with time, on-line triangulation is required. This tends to progressively increase the average clique size and further undermines efficiency.

In this article we take a different approach. We define the hierarchical junction tree (HJT): a framework that addresses efficiency issues explicitly arising from forecasting. These issues were first addressed in pioneering articles for time series with homogeneous structure (time slices) by Kjærulff [9] and Kanazawa *et al.* [6]. However, in the methodology described below we do not assume time homogeneity of structure. The HJT is constructed from the BN without loss of conditional independence when the ancestral graph of the conditioning variables is decomposable [12]. Only this ancestral graph needs to be triangulated when it is not decomposable. In contrast, the whole graph needs to be triangulated in the JT framework. In addition the HJT decreases its size as variables are learned to offset any increase in the descendants.

The HJT organises a collection of directed JTs into hierarchies through a directed tree. Its construction is achieved by identifying sections of the BN that are decomposable and organising them with the time consistent edge directions of the original BN. The computational complexity for propagation in HJTs is determined by the number of layers active at any given time. In typical examples this can be kept small because of the relative order of the conditioning and target variables in the BN.

In Section 2 we revise the definition of a BN and introduce a new concept, the ancestrality partition, which is the basic structure for the construction of the HJT. In Section 3 we revise the construction of JTs, and in Section 4 we demonstrate how the JT can be used for forecasting in a dynamic setting. In Section 5 we define the HJT and illustrate its construction from a BN. In Section 6 we demonstrate how the HJT is used for forecasting.

2 Bayesian Networks

We recall that a *Bayesian network* is a pair (G, P) consisting of a directed acyclic graph (DAG) $G = (V, E)$ and a probability distribution P. The set V is the index of a set of variables $\{X_v\}_{v \in V}$. The probability distribution P factorises according to G, that is, $P = \prod_{v \in V} P(X_v | X_{Pa(v)})$ where $Pa(v)$ is the set of parents of the node v in the node set V of G.

In the transition from the Bayesian network (BN) to the hierarchical junction tree (HJT), we divide the BN into layers. This division relies on the following concept. A *collider* in a BN is a node having at least two parents which are not joined by an edge. The layers of the BN is then based on the configuration of the colliders. A *decomposable* BN is a BN that has no colliders. Some of the nodes of the HJT will be *families* of the nodes of a BN, that is, sets of the form $Fa(v) = \{v\} \cup Pa(v)$. In fact they will be *maximal families* where family $Fa(v)$ is maximal if it is not contained in another family. An undirected path in a BN is a sequence of nodes $(v_1, \ldots, v_s)$ where v_i is either a parent or a child of v_{i+1}, $i = 1, \ldots, s-1$. A connected component W of a BN is a maximal subset of nodes that are joined with an undirected path. A directed path in a BN is a sequence of nodes $(v_1, \ldots, v_s)$ where v_i is a parent of v_{i+1}, $i = 1, \ldots, s-1$. Node v is an ancestor of node w or w is a descendant of node v if there is a directed path from v to w. The ancestral set of a subset of nodes W is the set W together with all ancestors of the nodes in W. The ancestral graph of W is the BN whose node set is the ancestral set of W and whose edges are inherited from the BN.

The first step in the transition from the BN to the HJT is the identification of the ancestrality partition of the BN. In the following we will introduce and discuss this new concept.

Let $B = (G, P)$ be a BN where $G = (V, E)$. The *ancestrality partition of* B is the partition of V,

$$\{I(1), H(1), I(2), \ldots, I(m-1), H(m-1), I(m)\}.$$

The elements of this partition are defined as follows,

$I(1)$ $=\{v : v$ is not a collider and has no collider ancestors$\}$,

$H(1)$ $=\{v : v$ is a collider and has no collider ancestors$\}$,

$I(i)$ $=\{v : v$ is not a collider, has at least one collider ancestor in $H(i-1)$ and has no collider ancestors in $V \setminus [H(1), \ldots, H(i-1)]\}$. This set is defined if $H(i-1)$ exists.

$H(i)$ $=\{v : v$ is a collider, has at least one collider ancestor in $H(i-1)$ and has no collider ancestors in $V \setminus [H(1), \ldots, H(i-1)]\}$. This set is defined if $H(i-1)$ exists and if it is not empty.

The existence and non-emptiness conditions guarantee that the sets in the ancestrality partition are well-defined, pairwise-disjoint and non-empty for finitely many i, see [12].

The ancestrality partition organises the BN into layers determined by the pattern of colliders. The first layer consists of $I(1)$, the second layer consists of $H(1) \cup I(2)$. In general, the i-th layer consists of $H(i-1) \cup I(i)$, $i = 2, \ldots, m$. These layers help to identify decomposable sections of the original BN from which the JTs of the HJT are constructed, see Section 5.

In the dynamic setting that we are considering here, new descendants are added sequentially to the BN as time passes. This sequential addition gives a total order of the nodes which is consistent with the partial order induced by the BN. We can construct the ancestrality partition of a BN given a total order by organising the nodes in layers. Let $(v(1), v(2), \ldots, v(n))$ be a total order and $L(v(i))$ denote the layer where $v(i)$ is. Starting with $v(1)$ the sequential rule is:

1) If $Pa(v(i)) = \emptyset$, set $L(v(i)) = 1$
2) If $Pa(v(i)) \neq \emptyset$ and $v(i)$ is not a collider, set $L(v(i)) = max\{L(u) : u \in Pa(v(i))\}$
3) If $Pa(v(i)) \neq \emptyset$ and $v(i)$ is a collider, set $L(v(i)) = max\{L(u) : u \in Pa(v(i))\} + 1$.

Then the elements of $I(1)$ are those nodes v with $L(v) = 1$. The elements of $H(i-1)$ are those colliders v with $L(v) = i$ and the elements of $I(i)$ are those non-colliders v with $L(v) = i$. The ancestrality partition of the BN given in Figure 1 is,

$I(1) = \{a, b, c, d, e, f, g, h, j, l\}$, $H(1) = \{i\}$, $I(2) = \{k\}$, $H(2) = \{m\}$, $I(3) = \emptyset$.

3 Junction Trees

In this section we review the concept of junction trees for the HUGIN architecture [1]. A junction tree (JT) for a vector X_V or with *universe* V, is a pair $\mathcal{T} = (\mathsf{T}, \mathsf{P})$. T is a tree of subsets of V such that the so-called

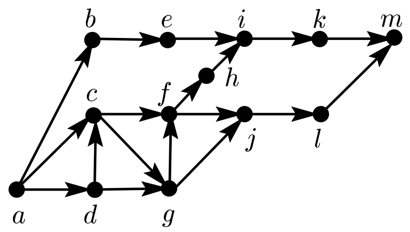

Figure 1: *The BN B for the example of the ancestrality partition*

junction tree property holds, that is, the intersection of two sets C_1 and C_2 is contained in all the sets in the unique path between C_1 and C_2. The edges of the junction tree are called separators. P is a potential on X_V, where a potential ϕ_U is a function whose domain is the state space of the variable X_U and whose counter domain is the real numbers. The potential P factorises according to T such that,

$$\mathsf{P} = \frac{\prod_{C \in \mathcal{C}} \phi_C}{\prod_{S \in \mathcal{S}} \phi_S},$$

where $\mathcal{C}$ is the set of nodes of T and $\mathcal{S}$ the set of separators.

We can construct a JT $\mathcal{T} = (\mathsf{T}, \mathsf{P})$ from a given decomposable BN $B = (G, P)$. The nodes of the tree T are cliques of the undirected version of G. The potentials ϕ_C are formed from the product of conditional probabilities $P(X_v|X_{Pa(v)})$ where $\{v\} \cup Pa(v) \subseteq C$. The potentials ϕ_S for the separators are initialised to 1. If the BN is not decomposable but its moral graph is chordal we can still construct a JT from the moral graph, but some conditional independence is lost in the moralisation step. If the BN is not decomposable and its moral graph is not chordal, then its moral graph is made chordal through a triangulation step before we construct the junction tree and this step can be computationally expensive. For a detail study on triangulation see [7, 8]. Triangulating the moral graph requires us to ignore some of the conditional independence encoded in the BN because we need to add extra edges, the so-called fill-in's. In a dynamic setting, the sequential addition of fill-in's as new variables are added to the system can often dramatically reduce efficiency.

Propagation in junction trees within the Hugin architecture is based on the operation of sum-flows from node C_i to node C_j, performed as follows,
1) compute $\phi'_S = \sum_{C_i \setminus S} \phi_{C_i}$ and assign it to S;
2) compute $\phi'_{C_j} = \phi_{C_j} * \frac{\phi'_S}{\phi_S}$ and assign it to C_j.

Propagation in the Hugin architecture is done in two steps. Let us suppose that we learn $X_a = x'_a$, or for short, that we learned X_a. First, we choose a clique that contains a, then we multiply the potential of this clique by the finding f_{a,x'_a} for x'_a. Second, we schedule sum-flows by choosing a clique as a root and then scheduling sum-flows first from the leaves of the tree to the root, the so-called Collect Evidence schedule, and then scheduling sum-flows from the root to the leaves, the so-called Distribute Evidence schedule. After these two schedules have been performed the JT is sum-consistent, that is, the potentials on the nodes and the separators hold the marginal potentials of the potential P of the JT. For a description of the Hugin architecture see [4, 2], and for a more technical description see [3]. Note that probabilistic propagation can be computed faster in Binary Join Trees [13] than in junction trees. In future research we will study the possibility of constructing hierarchical binary join trees and assess their properties.

4 Forecasting in the dynamic setting using junction trees

In this section we describe how JTs can be used for forecasting in a dynamic setting. We first address how a JT can be constructed sequentially and then demonstrate how it can be used for forecasting. The sequential construction is demonstrated for $t = 1$ and the transition from $t = 1$ to $t = 2$. However, the second transition is trivially identical to constructing a JT at time $t+1$ from a JT valid at time t. Figure 2 displays the BNs B_1 for time $t = 1$ and B_2 for time 2. BN B_1 is surrounded by the inner dashed lines and B_2 is surrounded by the outer dashed lines. Descendant nodes k, l and m were sequentially added into B_1 to obtain B_2.

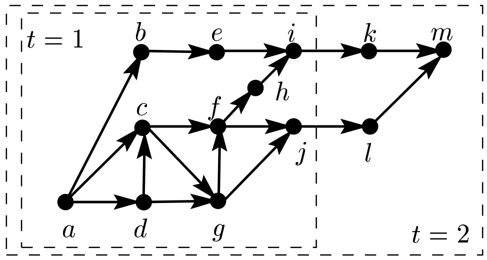

Figure 2: *The BN B_1 at time $t = 1$ is surrounded by the inner dashed box; the BN B_2 at time $t = 2$ is surrounded by the outer dashed box.*

The construction of the JT $\mathcal{T}_1$ from BN B_1 is done through a standard procedure: the BN is moralised, the moral graph is then triangulated and finally a tree of cliques is constructed from the triangulated graph. In the construction of JT $\mathcal{T}_1$, displayed in Figure 3, the fill-in's are: $\{a, f\}$, $\{a, g\}$, $\{b, f\}$ and $\{e, f\}$.

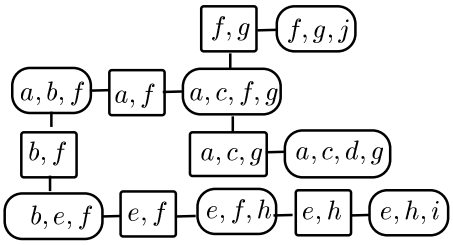

Figure 3: *The JT $\mathcal{T}_1$ constructed from BN B_1 in Figure 2.*

At time $t = 2$, the nodes k, l and m are added to B_2. If we wish to compute the marginals for X_m on B_2, we need to construct a JT for B_2. One choice would be to start over again and construct a JT for B_2. Another choice would be to use the fill-in's in the triangulation of the moral graph of B_1 as a starting point for the triangulation of the moral graph of B_2. Notice that in both choices on-line triangulation is required. The second choice is computationally more efficient because part of the moral graph of B_2 has been already triangulated with the triangulation of the moral graph of B_1.

The JT $\mathcal{T}_2$, constructed in this fashion, is displayed in Figure 4. The extra fill-in's are $\{b, g\}$, $\{e, g\}$, $\{f, i\}$, $\{g, i\}$, $\{i, j\}$ and $\{i, l\}$.

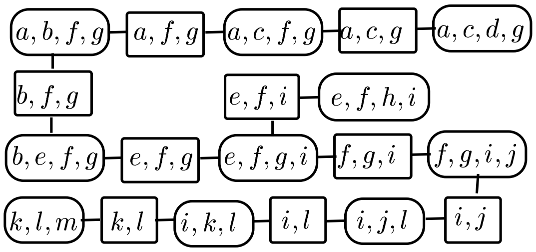

Figure 4: *The JT $\mathcal{T}_2$ constructed from BN B_2 in Figure 2*

In addition to the triangulation process being computationally expensive, the sequential on-line triangulation results in a sequential increment of the node/clique sizes. In $\mathcal{T}_2$, for example, node $\{a, b, f\}$ of $\mathcal{T}_1$ is of size 3 and it is included in node $\{a, b, f, g\}$ of $\mathcal{T}_2$ which is of size 4. Note as well that nodes $\{b, e, f\}$ and $\{e, f, g\}$ in $\mathcal{T}_1$ are joined together into node $\{a, e, f, g\}$ in $\mathcal{T}_2$.

Once we have constructed a JT, we can use it for forecasting. The JT presents an inefficiency in that the fill-in's added in the triangulation process may be superfluous once we learn a variable. In the JT $\mathcal{T}_2$, for example, if we learn that $X_a = x'_a$ and we remove a from B_2, then the fill-in's added because of the cycle (a, c, f, h, e, b) in the moral graph of B_2 are no longer needed. A solution would be to construct a JT for the reduced BN, but this process is computationally expensive. We will see in Section 6, that the HJT does not have this problem because no fill-in's are added at all.

5 Hierarchical Junction Tree

In this section we introduce the hierarchical junction tree and describe how it is constructed from a BN.

5.1 Definition

Definition 5.1 (Hierarchical junction tree) A *hierarchical junction tree $\mathcal{H}$ over ϕ_V* is a directed rooted tree $\mathcal{H} = (\mathcal{T}, \mathcal{F})$ where the edge direction is towards the root. The root is a directed junction tree over ϕ_V. The node set is $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p\}$ where each node $\mathcal{T}_i$, $1 \leq i \leq p$, is a directed junction tree. The ordered set $(\mathcal{T}_i, \mathcal{T}_j)$ is in the edge set $\mathcal{F}$ if $\mathcal{T}_i$ is a directed junction tree over ϕ_C where C is a node in the directed junction tree $\mathcal{T}_j$. The node C is then called a covering node in $\mathcal{T}_j$ and a cover of $\mathcal{T}_i$. Hereafter the term "hierarchical junction tree" is abbreviated to "HJT".

The covering nodes of a JT $\mathcal{T}'$ in the HJT act as dummy nodes. In fact, the potential associated with a covering node has a JT representation given by the parent $\mathcal{T}$ of $\mathcal{T}'$ in the HJT. In the JTs in a HJT, the children play an important role as they actually connect the parent and child JT. The children of covering nodes are called *upper doors* and the separator of a covering node with an upper door is called an *elevator*. The analogy we use here is that of traveling in time up a multistorey building learning about some aspects of lower levels as we proceed.

5.2 The Construction of an HJT from a BN

In this section we demonstrate how to sequentially construct a HJT. We illustrate the construction of the HJT at time $t = 1$ and then the transition from time $t = 1$ to time $t = 2$. However this transition is identical to the transition from t to $t+1$. We construct the HJTs $\mathcal{H}_1$ and $\mathcal{H}_2$ from the BNs B_1 and B_2 displayed in Figure 2.

5.2.1 The construction of $\mathcal{H}_1$

The sequence in which the nodes arrived to B_1 is $\{a, b, d, c, e, g, f, h, i, j\}$, which is a total order of the nodes that is consistent with the partial order induced by the BN.

Step 1: The identification of the ancestrality partition of the BN. This task was described in Section 2. The ancestrality partition for B_1 is: $I_1(1) = \{a, b, d, c, e, g, f, h, j\}$, $H_1(1) = \{i\}$, $I_1(2) = \emptyset$.

Step 2: The construction of the junction forest for Layer 1. The junction forest for Layer 1 is constructed from the BN $B_{I_1(1)}$, displayed in Figure 5 whose node set is $I_1(1)$ and whose edges are inherited from B_1. The junction forest for this example consists of only one JT. Let us denote this JT by $\mathcal{T}_1^1$. The nodes of $\mathcal{T}_1^1$ are the maximal families of the nodes in $B_{I_1(1)}$, i.e., except for those families that are subsets of another family, the sets of the form $Fa(v) = \{v\} \cup Pa(v)$. The set $\{a, c, d\}$, for example, is the family $Fa(c) = \{c\} \cup Pa(c)$ and thus it will be a node of $\mathcal{T}_1^1$. Similarly $Fa(g) = \{d, c, g\}$ will be a node in $\mathcal{T}_1^1$. In contrast the set $Fa(d) = \{a, d\}$ will not be a node of $\mathcal{T}_1^1$ because it is a subset of $Fa(c)$.

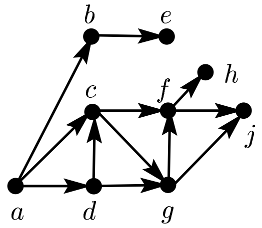

Figure 5: *The BN $B_{I_1(1)}$ induced by the set $I_1(1)$ on the BN B_1 in Figure 1*

The edge set of $\mathcal{T}_1^1$ is constructed from the edge set of $B_{I_1(1)}$. The maximal family $Fa(v_1)$ is a parent of maximal family $Fa(v_2)$ in $\mathcal{T}_1^1$ if w_1 is a parent of w_2 in $B_{I_1(1)}$ and $Fa(w_1) \subseteq Fa(v_1)$ and $Fa(w_2) \subseteq Fa(v_2)$. However, in most cases we have that v_1 is a parent of v_2. So, for example, $Fa(c)$ is a parent of $Fa(g)$. Each conditional probability $P(X_v|X_{Pa(v)})$ is assigned to the node that contains the family $Fa(v)$, and all separators are initialised to 1. The directed JT $\mathcal{T}_1^1$ constructed from $B_{I_1(1)}$ as described above is displayed at the bottom of Figure 6. The node $\{a, b\}$ for example, is $Fa(b) = \{b\} \cup Pa(b)$ and contains $Fa(a) = \{a\}$. Similarly node $\{a, d, c\}$ is $Fa(c)$. Note that there is an edge from $Fa(b)$ to $Fa(c)$ because a is a parent of c in $B_{I_1(1)}$ and $Fa(a) \subset Fa(b)$.

Step 3: The construction of the JTs for the elements of $H_1(1)$. In this example, $H_1(1)$ has one element: $\{i\}$. A collider adds a new JT which contains covering nodes. Let us denote this JT by $\mathcal{T}_1^2$. The potential covering nodes for $\mathcal{T}_1^2$ are the connected components of the BN induced in

B_1 by the previous layers. The covering nodes for $\mathcal{T}_1^2$ are those connected components that intersect $Pa(i)$. In our case the only component is $I_1(1)$ and it intersects $Pa(i)$. We add node $Fa(i) = \{e, h, i\}$ to this JT joining it with a directed edge from the covering node. Node $Fa(i)$ has $P(X_i|X_e, X_h)$ as potential and the potential of its separator with the covering node is initialised to 1. The covering node is not given a potential because it is a dummy node whose potential is given by JT $\mathcal{T}_1^1$.

Step 4: The construction of the edge set of the HJT. In Steps 2 and 3 we constructed the nodes of the HJT $\mathcal{H}$. In Step 4 we add the edges of $\mathcal{H}_1$. We add an edge from JT $\mathcal{T}_1$ to JT $\mathcal{T}_2$ if $\mathcal{T}_2$ has a covering node that is the universe of $\mathcal{T}_1$. In our example we have two nodes: $\mathcal{T}_1^1$ and $\mathcal{T}_1^2$. JT $\mathcal{T}_1^2$ has covering node $I(1)$ which is the universe of $\mathcal{T}_1^1$, therefore there is a directed edge from $\mathcal{T}_1^1$ to $\mathcal{T}_1^2$. The HJT $\mathcal{H}_1$ constructed from B_1 is displayed in Figure 6.

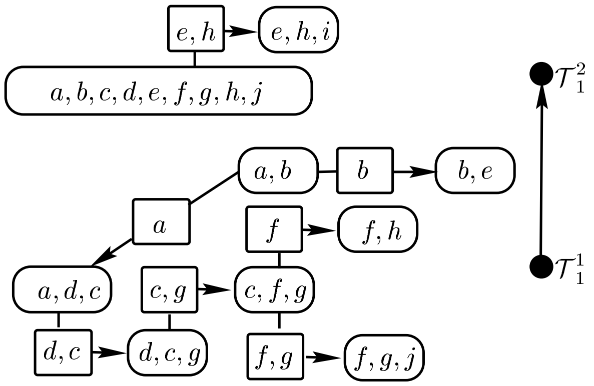

Figure 6: *The HJT $\mathcal{H}_1$ constructed from the BN B_1 displayed in Figure 1. The JT for Layer 1 is $\mathcal{T}_1^1$; the JT for Layer 2 is $\mathcal{T}_1^2$.*

5.2.2 The construction of $\mathcal{H}_2$

Starting with B_1, the new nodes k, l and m are sequentially added to obtain B_2, displayed in Figure 2.

Step 1: Updating the ancestrality partition. Node k has only one parent which is node i. So, because k is not a collider and its parents are in Layer 2, k must be in Layer 2: so $I_2(2) = \{k\}$. Node l has j as its only parent. Thus l is in Layer 1 because j is in Layer 1. Node m has parents k and l and so m, being a collider, is in Layer 3, because l is in Layer 1 and k is in Layer 2. The updated ancestrality partition for B_2 is thus: $I_2(1) = \{a, b, d, c, e, g, f, h, j, l\}$, $H_2(1) = \{i\}$, $I_2(2) = \{k\}$, $H_2(2) = \{m\}$.

Step 2: Updating the HJT. The addition of node k to B_1 adds the non-covering node $Fa(k) = \{i, k\}$ to $\mathcal{T}_1^2$ to obtain $\mathcal{T}_2^2$. Since k is in Layer 2 we add $Fa(k)$ to $\mathcal{T}_1^2$ with an edge from $Fa(i)$ to $Fa(k)$ because i is a parent of k. The addition of node l adds the non-covering node $Fa(l) = \{j, l\}$ to $\mathcal{T}_1^1$ to obtain $\mathcal{T}_2^1$, because l is in $I_2(1)$. The addition of node m adds a new JT $\mathcal{T}_2^3$ because m is a collider. In general the possible covering nodes are the connected components of the BN induced in B_2 by the previous layers. A connected component becomes a covering node if it intersects the parent set of the collider. In this example the covering node is $I_2(1) \cup H_2(1) \cup I_2(2)$. The node $Fa(m) = \{k, l, m\}$ is added to $\mathcal{T}_2^3$ and joined with a directed edge from the covering node. The node $Fa(m)$ is given the potential $P(X_m | X_k, X_l)$ and the separator potential is initialised to 1. The potential for the covering node is already given by JT $\mathcal{T}_2^2$. The HJT $\mathcal{H}_2$ is displayed in Figure 7.

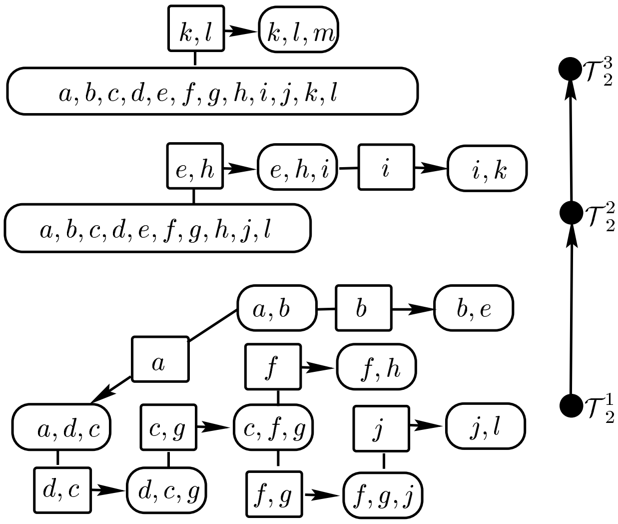

Figure 7: *A HJT $\mathcal{H}_2$ constructed from B_2 displayed in Figure 2.*

An immediate consequence of the construction of a HJT is expressed in the following proposition.

Proposition 5.1 *The non-covering nodes of a HJT constructed from a BN are the maximal families $Fa(v)$ of the nodes of the BN. For a non-covering node $Fa(v)$ of a HJT constructed from the BN as described above, the parent set $Pa(v)$ of v is given by the set $\cup_r S_r$, where the sets S_r are the separators of $Fa(v)$ with its parents in the JT where $Fa(v)$ belongs.*

It is important to note that in general following the steps above do not always construct a JT as the resulting graph may not be a tree. Fortunately, this does not disrupt the forward propagation algorithm presented in Section 6. The discussion of this case is long winded and can be found in [12].

In the construction of the HJT, the edge direction is preserved through the edge direction of the JTs. This preservation provides a basis for translating Pearl's d-separation criterion to read conditional independence in a BN, to a criterion that we call "h-separation criterion" to read conditional independence from the HJT. The translation of the d-separation provides a proof that the HJT can be constructed without loss of conditional independence [12]. In particular note that no triangulation was required to construct the HJT. In the case when the BN is decomposable the HJT simplifies to a JT and the h-separation criterion to a criterion to read conditional independence on JTs. This criterion then says what we already know: separators separate!

Madsen & Jensen [11] wrote "It has for a long time been a puzzle why standard inference algorithms for BNs did not use the direction of the links in the network". Their Lazy propagation method provides a framework for propagation that utilises the direction of some of the edges. The HJT provides another framework for propagation that utilises the direction of all edges of the original BN: see Section 6.

6 Propagation in Hierarchical Junction Trees

Propagation in HJTs is inherited from propagation in JTs. We can see a HJT as a JT. In the HJT $\mathcal{H}_2$ displayed in Figure 7, for example, we may see $\mathcal{H}_2$ as the JT $\mathcal{T}_2^3$, where the potential of its covering node is given by $\mathcal{T}_2^2$. In addition, the potential of the covering node of $\mathcal{T}_2^2$ is given by $\mathcal{T}_2^1$.

In this article we are restricting our attention to forecasting: we want to compute marginals for descendant nodes given some of the ancestor nodes at a given time t. This implies that we require sum-flows from a covering node to an upper door. Let us denote this type of sum-flow by meta-sum-flow. A meta-sum-flow requires the computation of the joint distribution for variables that are not in the same node. This can be done using of the method of firing variables described in [4]. Note that Kjærulff [10] first introduced this idea in his Nested JTs propagation method.

In the following we describe how forecasting works in HJTs. In general we operate on a fixed HJT for time t. Here we demonstrate how forecasting works for the HJT for time $t = 2$, but the operations presented are also valid for a HJT for any given time t. The HJT for time $t = 2$ is shown in Figure 7. Let us suppose that the potential of the non-covering nodes $Fa(v)$ of this HJT has the conditional distribution $P(X_v|X_{Pa(v)})$ and that the separators have been initialised to 1. Let us address the forecasting task where the conditioning variable is X_a in $\mathcal{T}_2^1$ and our target variable is X_m which is in $\mathcal{T}_2^3$.

Step 1: Propagation within $\mathcal{T}_2^1$. Entering and propagating that $X_a = x'_a$ is done as in the standard JT. We first multiply a node that contains a by a finding f_{a,x'_a} for x'_a. We then propagate this information through $\mathcal{T}_2^1$. We choose node $\{a, b\}$ and multiply it by f_{a,x'_a}. In the Hugin architecture we choose a root, say C. We then need to perform a schedule of sum-flows given by a Collect Evidence operation to C, followed by a schedule of sum-flows given by a Distribute Evidence operation from C.

We choose the node $\{a, b\}$ as our root. All the sum-flows in the Collect Evidence to $\{a, b\}$ are vacuous because they do not affect any of the potentials. This feature is a consequence of the preservation of the edge direction in the HJT. In the sum-flow from $\{b, e\}$ to $\{a, b\}$, for example, node $\{b, e\}$ has the conditional distribution $P(X_b|X_e)$ and when we sum-marginalise it we obtain a potential identical to 1. The potential of the separator b is already identical to 1. Thus replacing it with the same potential is unnecessary. It is also unnecessary to multiply the potential of node $\{a, b\}$ by 1.

This feature is also part of Madsen & Jensen's [11] Lazy propagation method. Their method records the edge direction in the potentials which is then used to identify vacuous sum-flows. The Lazy propagation method is intended for general propagation tasks: that is, the computation of the distribution of any set of variables conditional on any other set of variables. This level of generality requires triangulation which does not allow the transfer of the edge direction of the original BN to the graph of the JT. In contrast, in the HJT we restrict our attention to forecasting which allows the preservation of the direction of all edges.

In conclusion, we only need to compute the sum-flows in the Distribute

Evidence from $\{a, b\}$. At this point the JT $\mathcal{T}_2^1$ is sum-consistent.

Step 2: Graph simplification. After we have propagated the information through $\mathcal{T}_2^1$, we can then continue propagating this information to the JT in the next upper layer. However, before we proceed to the next JT we can remove the unnecessary zeros encoded in $\mathcal{T}_2^1$. This feature is analogous to the table compression in the Hugin architecture but the HJT also simplifies its graph: an important feature in the dynamic setting as new descendant variables are added with time. The potential $\phi_{\{a,d,c\}}$ for node $\{a, d, c\}$, for example, is the same as the potential $\phi_{\{d,c\}}$ for separator $\{d, c\}$ except for when we evaluate $\phi_{\{a,d,c\}}(x_a, x_d, x_c)$ for $x_a \neq x'_a$ where the potential is zero. Thus we can remove potentials $\phi_{\{a,d,c\}}$ and $\phi_{\{d,c\}}$ because their contribution to the potential of the JT cancels out. This removal is reflected in the graph of the JT by removing the node $\{a, d, c\}$ and the separator $\{d, c\}$. The same reasoning applies to node $\{a, b\}$ and separator $\{b\}$ and they are removed from $\mathcal{T}_2^1$. The separator $\{a\}$ is also removed because it only holds a constant. The removal of $\{a\}$ in $\mathcal{T}_2^1$ implies that we need to remove a from the covering nodes of $\mathcal{T}_2^2$ and $\mathcal{T}_2^3$. The resulting HJT $\mathcal{H}_{2+}$ is displayed in Figure 8.

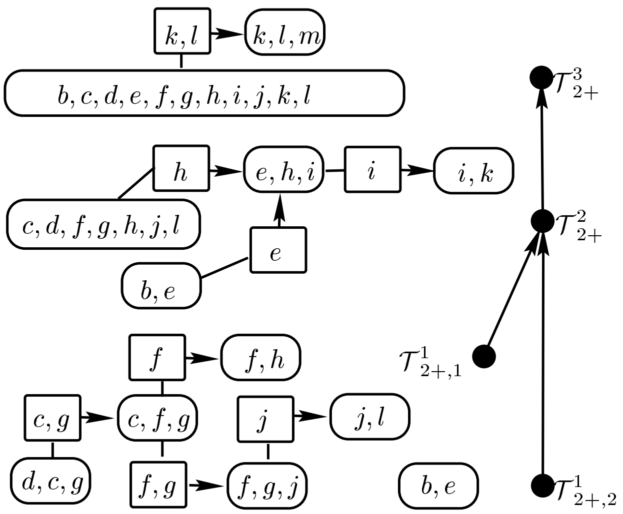

Figure 8: *The HJT $\mathcal{H}_{2+}$ obtained from $\mathcal{H}_2$ by entering the evidence that X_a has been learned.*

Step 3: Propagation within $\mathcal{T}_{2+}^2$. We need to propagate the update potentials of the covering nodes of $\mathcal{T}_{2+}^2$ to the rest of the JT. At this point the potential of the non-covering nodes have the conditional distribution inherited from the original BN, and the separator potentials are set to 1. The schedule that will be used is formed by a schedule from the leaves to the covering nodes followed by a schedule from the covering nodes to the leaves. This schedule is a generalisation of the Hugin architecture because it simplifies to a Collect and a Distribute Evidence schedule when there is only one covering node.

The sum-flows in the schedule from the leaves to the nodes are all vacuous for the same reasons as we discussed in Step 1. The sum-flows from the covering nodes to the leaves need to be computed. Note that we need to compute a sum-meta-flow from covering node $\{c, d, f, g, h, j, l\}$ to upper door $\{e, h, i\}$. In this case this sum-meta-flow consists of computing the marginal for X_h from the potential of node $\{f, h\}$ in the JT $\mathcal{T}^1_{2+,1}$.

Step 4: Graph Simplification. The next step is to propagate to $\mathcal{T}^3_{2+}$. However $\mathcal{H}_{2+}$ admits another simplification. There is no reason to keep JT $\mathcal{T}^1_{2+,1}$ in a different level than $\mathcal{T}^2_{2+}$ because there is no undirected cycle induced when we join them. The same reasoning applies to covering node $\{b, e\}$.

Let us first join JT $\mathcal{T}^2_{2+}$ with $\mathcal{T}^1_{2+,1}$. We join the upper door $\{e, h, i\}$ to a node in $\mathcal{T}^1_{2+,1}$, which will be termed *a lower door* of $\{e, h, i\}$. A lower door for $\{e, h, i\}$ must contain a family $Fa(v) = \{v\} \cup Pa(v)$ where v is in the elevator. The elevator for $\{e, h, i\}$ is $\{e, h\}$ and h is in $\mathcal{T}^1_{2+,1}$, and so we need to find $Fa(h)$. Using Proposition 5.1 in Section 5, we know that node $Fa(h) = \{f, h\}$ and thus a lower door for $\{e, h, i\}$ is $\{f, h\}$. We then join $\{f, h\}$ to $\{e, h, i\}$ with a directed edge, see Figure 9. Node $\{b, e\}$ is the other lower door of $\{e, h, i\}$. Node $\{b, e\}$ contains e and because $\{b, e\}$ has no parents in $\mathcal{T}^1_{2+,1}$, it must contain $Fa(e)$. The simplified HJT $\mathcal{H}_{2++}$ is displayed in Figure 9. Notice that $\mathcal{H}_{2++}$ has only two layers.

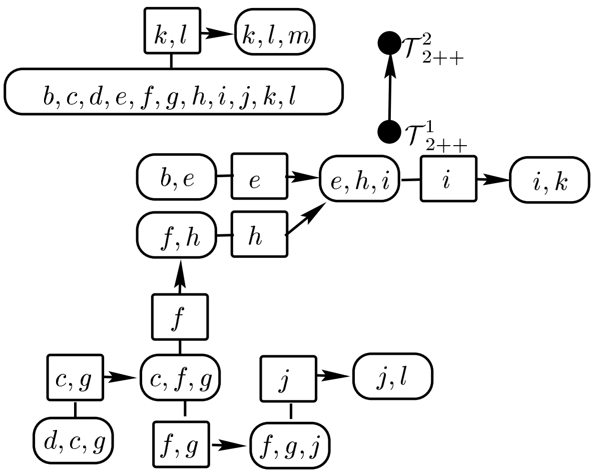

Figure 9: *The HJT $\mathcal{H}_{2++}$ obtained from $\mathcal{H}_{2+}$ by joining $\mathcal{T}^1_{2+}$ and $\mathcal{T}^2_{2+}$.*

Step 5: Propagation within $\mathcal{T}^2_{2++}$. In this task we choose the covering node as a root and perform Collect and Distribute Evidence. In this case, like in Steps 1 and 3, the sum-flows in the Collect Evidence schedule are vacuous. The sum-flows in the Distribute Evidence need to be computed. The schedule in this case consists of the sum-meta-flow from the covering node $\{b, c, d, e, f, g, h, i, j, k, l\}$ to the upper door $\{k, l, m\}$. This requires us to compute the joint marginal distribution for X_k and X_l. We can compute this marginal using the method of firing variables on the JT $\mathcal{T}^1_{2++}$, described in [4].

The order of the computational complexity of a sum-meta flow is then determined by the cardinality of the elevator and the cardinality of the largest maximal family in the JT whose universe is the covering node. Let us suppose that s is the size of the state space of all variables. Then the computational complexity in our example is of order $s^1 * s^3$, where s^1 is due to the cardinality of the elevator being 2 and s^3 due to the cardinality of the largest maximal family in $\mathcal{T}^1_{2+,1}$ being 3. If a further elevator would be involved then the order of the computational complexity of the sum-meta-flow would be multiplied by s^{r-1} where r is the cardinality of this extra elevator. Kjærulff [10] provides a detailed discussion of the computational complexity of this type of operation.

At this point the potential in node $\{k, l, m\}$ encodes the marginal joint distribution for the variables X_k, X_l and X_m. The marginal for X_m can be computed from this potential through sum-marginalisation.

The description above is for the case when we learn a variable in the first layer of the HJT. If we learn a variable in an upper layer we need to triangulate the ancestral graph of this variable. Let us suppose we have the BN B_2, displayed in Figure 2, and we learn that $X_k = x'_k$. In this case the conditional independence statements of the ancestral variables of X_k changes and we need to bring the ancestors of k to the first layer $I_2(1)$. We then need to triangulate the BN induced by the new layer. The HJT $\mathcal{H}_{2\triangle}$ constructed in this manner is displayed in Figure 10. Note that the JT $\mathcal{T}^1_{2\triangle}$ is undirected. We draw this JT with undirected edges because it is not possible to keep track of the edge direction of the original BN due to the fill-in's added in the triangulation step. In this case, if we wish to improve computational efficiency, we can use either Nested JTs [10] or Lazy propagation [11] algorithms.

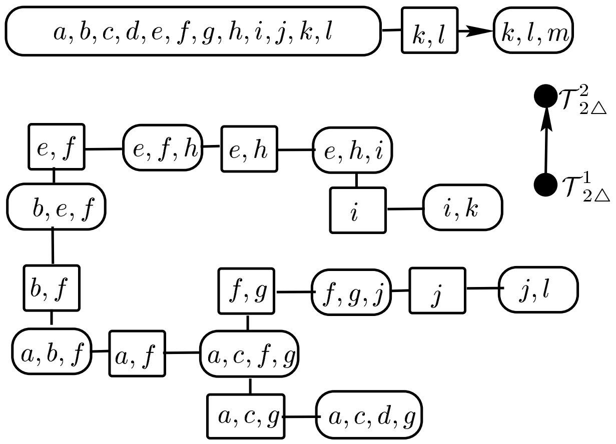

Figure 10: *The HJT $\mathcal{H}_{2\triangle}$ obtained from $\mathcal{H}_2$ by triangulating the ancestral graph of k.*

7 Conclusions

The HJT provides a framework for forecasting in dynamic systems with heterogeneous evolution, such as the system defined by Smith & Papamichail [15]. The use of the standard JT, that is the JT with the Hugin architecture in these type of settings leads to chronic inefficiencies. This is simply because the Hugin architecture is not specifically designed for this task, but rather for the computation of marginals of any set of variables conditional any other set of variables. This generality requires the re-computation of a JT whenever new variables enter the BN which translates to on-line triangulation. The necessity of on-line triangulation creates a sequence of JTs whose nodes increase in size with time, see Section 4. In contrast, if we restrict our attention to forecasting where all that is required is the computation of marginals of some descendant variables given some ancestor variables in the BN, we can make enormous efficiency gains. We have noted that the sequential construction of the HJT requires no loss of conditional independence [12], and so no triangulation is needed. Furthermore the HJT is sympathetic to the addition of descendant nodes to a BN, see Section 5. This simply corresponds to the addition of non-covering nodes to the HJT of the last BN.

If the conditioning variables are in the first layer, no triangulation is required for computing marginals of descendant variables. If the conditioning variables are not in the first layer, we need to bring them and their ancestors to the first layer. But now only the construction of the JTs for this layer requires triangulation, see Section 6.

The order of the computational complexity of the propagation in HJTs is determined by the number of layers, the cardinality of the elevators and the cardinality of the non-covering nodes. However we have demonstrated in Section 6 that the HJT simplifies in the light of sequentially learning the conditioning variables, which effectively sequentially decreases the number of layers as new layers of target variables are added. This means that in many forecasting tasks, at any one time, only a few layers of the HJT are active and so efficient propagation is possible.

In this paper we have focused our attention only on the forecasting problem. However the HJT is a useful frame for propagation in a much wider setting whenever there is interest only in certain target nodes and conditioning sets, where the conditioning sets do not contain all terminal nodes of the BN and the target nodes all roots. For a much more detailed discussion of this general scenario see [12]. Of course in standard medical applications where roots and target nodes are diseases and terminal nodes and conditioning variables are observed symptoms, the efficiency gains of the HJT are at least minimal. This is also true of hereditary models where we typically condition on the observed current population and our target is the structure of the ancestors. This type of applications dominated the development of systems like Hugin and accounts for the puzzling redundancy of directionality. However, for applications like forecast systems where the main focus of attention is on future effects rather than past causes, we have demonstrated that such directionality can definitely be used. A full characterisation of when the HJT is more efficient than the JT is harder to achieve and will be the subject of future research.

Acknowledgement

This research was performed while R. O. Puch was supported by the Mexican council for science and technology (CONACYT-88260).

References

[1] SK Andersen, KG Olesen, FV Jensen, and F Jensen. Hugin – a shell for building Bayesian belief universes for expert systems. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1080–5. Morgan Kaufmann, San Mateo, California, (Ed. NS Shridharan), 1989.

[2] RG Cowell, AP Dawid, SL Lauritzen, and DJ Spiegelhalter. *Probabilistic networks and expert systems.* Springer Verlag, 1999.

[3] AP Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.

[4] FV Jensen. *An introduction to Bayesian networks.* University College London Press, London, United Kingdom, 1996.

[5] FV Jensen, SL Lauritzen, and KG Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[6] K Kanazawa, D Koller, and S Rusell. Stochastic simulation algorithm for dynamic probabilistic networks. In *Proceedings of the 11th conference in uncertainty in artificial intelligence (Eds. P Besnard and S Hanks)*, pages 346–351. Morgan Kaufmann, San Francisco, California, 1995.

[7] U Kjaerulff. Triangulation of graphs – algorithms giving small total state spaces. *Research Report,* Institute of Electronic Systems, Department of Mathematics and Computer Science, Aalborg University, Aalborg, Denmark., 1990.

[8] U Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2:7–17, 1992.

[9] U Kjaerulff. dhugin: A computational system for dynamic time-sliced Bayesian networks. *International Journal of Forecasting,* Special Issue on Probability Forecasting, 11:89–111, 1995.

[10] U Kjærulff. *Inference in Bayesian networks using Nested Junction Trees*, pages 51–74. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[11] AL Madsen and FV Jensen. Lazy propagation: A junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1-2):203–245, 1999.

[12] RO Puch. *Hierarchical Junction Trees.* PhD thesis, Department of Statisitcs, University of Warwick, 2000.

[13] PP Shenoy. Binary join trees for computing marginals in the Shenoy-Shafer architecture. *International Journal of Approximate Reasoning*, 17:239–63, 1997.

[14] JQ Smith, AE Faria, S French, D Ranyard, D Vlesshhouwer, J Bohunova, T Duranova, M Stubna, L Dutton, C Rojas, and A Sohier. Probabilistic data assimilation within rodos. *Radiation Proteccion Dosimetry*, 73(1-4):57–9, 1997.

[15] JQ Smith and KN Papamichail. Fast Bayes and the dynamic junction forest. *Artificial Intelligence*, 107:99–124, 1999.