On Directed and Undirected Propagation Algorithms for Bayesian Networks

Christophe Gonzales[1], Khaled Mellouli[2], and Olfa Mourali[3]

[1] LIP6 – université Paris 6 – France
[2] IHEC – Carthage – Tunisie
[3] ISIG – Université de Kairouan

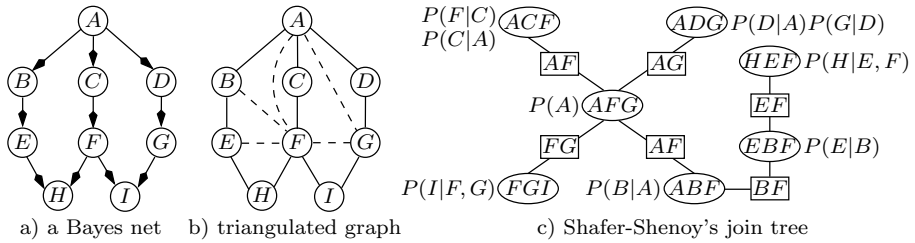
Abstract. Message-passing inference algorithms for Bayes nets can be broadly divided into two classes: i) *clustering* algorithms, like Lazy Propagation, Jensen's or Shafer-Shenoy's schemes, that work on secondary undirected trees; and ii) *conditioning* methods, like Pearl's, that use directly Bayes nets. It is commonly thought that algorithms of the former class always outperform those of the latter because Pearl's-like methods act as particular cases of clustering algorithms. In this paper, a new variant of Pearl's method based on a secondary directed graph is introduced, and it is shown that the computations performed by Shafer-Shenoy or Lazy propagation can be precisely reproduced by this new variant, thus proving that directed algorithms can be as efficient as undirected ones.

1 Introduction

In the last years, Bayesian nets (BN) [1–3] have become an increasingly popular knowledge representation framework for reasoning under uncertainty. They combine a directed acyclic graph (DAG) encoding a decomposition of a joint probability distribution over a set of random variables with powerful exact inference techniques [3–14]. These can answer various queries including *belief updating*, i.e., computing the posterior probability of every variable given a set of observations, *finding the most probable explanation*, i.e., finding a maximum probability assignment of the unobserved variables, *finding the maximum a posteriori hypothesis*, i.e., finding an assignment to a subset of unobserved variables maximizing their probability. This paper will be restricted to belief updating.

Although the BN's graphical structure is very efficient in its ability to provide a compact storage of the joint probability, it is not well suited for probabilistic computations when the DAG is multiply-connected [15]. A much more efficient structure called a *join* or *junction tree* and representing an alternative decomposition of the joint probability has been introduced in the 90's [16, 9] to serve as a support for inference algorithms [9, 10, 12, 17] (Fig. 1.c). Unlike the original BN, this new structure is undirected and, since [15]'s paper, the idea that propagation based on undirected graphs always outperform the variants of Pearl's algorithm (based on directed graphs) [18, 19, 3, 20] has often been conveyed in the literature. However, in a BN, the arc orientations provide some independence information called *d-separation* that can be effectively exploited to reduce

the inference computational burden [21] but that is lost by junction trees. For instance, in Fig. 1.a, if the value of A is known, then d-separation asserts that B is independent of C, D, F, G and I. Hence, upon receiving a new evidence on B, only the *a posteriori* probabilities of E and H need be updated.



a) a Bayes net b) triangulated graph c) Shafer-Shenoy's join tree

Fig. 1. A Bayesian network and one of its join trees.

The aim of this paper is to show how undirected inference techniques such as Shafer-Shenoy's method [12, 17] or Lazy Propagation [10] can be viewed as a variant of Pearl's algorithm. More precisely, it is shown that the computations performed by both algorithms in a join tree derived from a variable elimination technique similar to [7] can be precisely reproduced by Pearl with local conditioning performed on a particular DAG. The advantage of translating undirected inference techniques into directed ones is that d-separation can easily be applied to speed-up computations (see above). As for Lazy Propagation, which already uses d-separation, the advantage lies in the possibility of improving online triangulations or even avoiding them. Moreover, keeping the secondary structure used for computations as close as possible to the original one is attractive as it minimizes the quantity of information lost passing from one structure to the other (triangulation actually loses d-separation informations). This can prove useful for instance in hybrid propagation methods [22, 23] where approximate algorithms are used on some subgraphs of the BN, to select the most appropriate stochastic algorithm for each approximated subgraph, e.g., in some particular cases, it can be proved that logic sampling converges faster than Gibbs sampling.

The paper is organized as follows: Section 2 presents BN and describes Shafer-Shenoy's method. Section 3 illustrates on an example how the same computations can be conducted using a particular DAG and a general scheme for constructing this DAG is derived. Section 4 presents Pearl's method with local conditioning and shows why a new variant, when applied on such DAG, corresponds to Shafer-Shenoy. Section 5 extends these results to binary join trees and to Lazy Propagation. Finally Section 6 concludes the paper.

2 Bayesian Networks and Shafer-Shenoy's Algorithm

Definition 1. *A Bayesian network is a triple* $(\mathcal{V}, \mathcal{A}, \mathcal{P})$*, where* $\mathcal{V} = \{X_1, \ldots, X_n\}$ *is a set of random variables;* $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ *is a set of arcs which, together with*

$\mathcal{V}$, *constitutes a directed acyclic graph* $\mathcal{G} = (\mathcal{V}, \mathcal{A})$; $\mathcal{P} = \{P(X_i|\mathrm{Pa}_i) : X_i \in \mathcal{V}\}$ *is the set of conditional probability matrices of each random variable* X_i *given the values of its parents* Pa_i *in graph* $\mathcal{G}$. *The BN represents a joint probability distribution over* $\mathcal{V}$ *having the product form:* $P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\mathrm{Pa}_i)$.

Thus, in the BN of Fig. 1.a, $P(\mathcal{V})$ can be decomposed as $P(A)P(B|A)P(C|A)$ $P(D|A)P(E|B)P(F|C)P(G|D)P(H|E,F)P(I|F,G)$. Shafer-Shenoy's algorithm uses a secondary undirected structure called a join (or junction) tree to perform probabilistic inference, see Fig. 1.c. As shown by [24], this structure can always be constructed from a triangulated graph (Fig. 1.b) resulting from an elimination sequence of the random variables. Here, we used H,I,E,D,C,B,A,F,G. The cliques (resp. separators) of the join tree, i.e., the ellipses (resp. rectangles), are initially filled with functions (called *potentials*) of the variables they contain. Usually, cliques are filled with the conditional probabilities of the BN and separators with unity tables, i.e., tables filled with 1's. Shafer-Shenoy then performs inference by sending messages in both directions along the edges of the junction tree. A message sent from a clique C_i to an adjacent clique C_j through separator $S_{ij} = C_i \cap C_j$ is computed by multiplying the potentials stored in C_i by the messages received from all the adjacent cliques of C_i except C_j and then summing out the result over the variables not in S_{ij}. For instance, on Fig 2, message ❶ corresponds to $\sum_G P(A) \times ② \times ④ \times ⑤$. Semantically, these operations correspond to marginalizing out from the joint probability the variables in $C_i \backslash S_{ij}$.

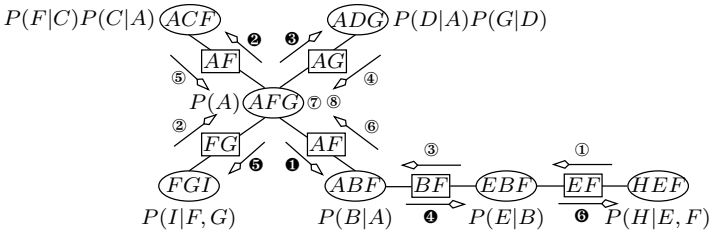


Fig. 2. The join tree and Shafer-Shenoy's inward and outward pass.

As for the order in which messages are generated, Shafer-Shenoy advocates to use an asynchronous algorithm, but we prefer presenting here a collect/diffusion (or inward/outward) method as it is more convenient for the next Sections and it is well known that both schemes produce the same set of messages:

Algorithm 1 (generation of messages).

1. *choose an arbitrary clique, e.g.,* AFG, *as the current clique;*
2. *inward pass: the current clique asks its adjacent cliques for their messages. In turn, they recursively ask their other adjacent cliques for messages. When a clique has received all the messages it waited for, it sends its own message.*
3. *outward pass: after the inward pass, clique* AFG *sends messages to its neighbors; they recursively send messages to their other adjacent nodes, and so on.*

Following the elimination sequence mentioned above, the messages sent during the inward and outward pass are computed as shown in the Table below. In this table, messages like $\mathbb{1}_{\mathcal{K}}$ represents $|\mathcal{K}|$-matrices filled with 1's and those like $P(\mathcal{T})_{\mathcal{K}}$ represent $|\mathcal{T}| \times |\mathcal{K}|$-matrices filled with $P(\mathcal{T})$ for every value of $\mathcal{K}$:

Table 1. Shafer-Shenoy's inward and outward pass computations.

in:	elim node	sending clique	computation	message
	H	HEF	$\mathbb{1}_{EF} = \sum_H P(H\|E,F)$	①
	I	FGI	$\mathbb{1}_{FG} = \sum_I P(I\|F,G)$	②
	E	EBF	$\mathbb{1}_{BF} = \sum_E P(E\|B) \times$ ①	③
	D	ADG	$P(G\|A) = \sum_D P(D\|A)P(G\|D)$	④
	C	ACF	$P(F\|A) = \sum_C P(F\|C)P(C\|A)$	⑤
	B	ABF	$\mathbb{1}_{AF} = \sum_B P(B\|A) \times$ ③	⑥

out:	message	computation
	❶	$P(A,F) = \sum_G P(A)\mathbb{1}_{FG}P(G\|A)P(F\|A) = \sum_G P(A) \times$ ② $\times$ ④ $\times$ ⑤
	❷	$P(A)_F = \sum_G P(A)\mathbb{1}_{FG}P(G\|A)\mathbb{1}_{AF} = \sum_G P(A) \times$ ② $\times$ ④ $\times$ ⑥
	❸	$P(A)_G = \sum_F P(A)\mathbb{1}_{FG}P(F\|A)\mathbb{1}_{AF} = \sum_F P(A) \times$ ② $\times$ ⑤ $\times$ ⑥
	❹	$P(B,F) = \sum_A P(B\|A)P(A,F) = \sum_A P(B\|A) \times$ ❶
	❺	$P(F,G) = \sum_A P(A)P(G\|A)P(F\|A)\mathbb{1}_{AF} = \sum_A P(A) \times$ ④ $\times$ ⑤ $\times$ ⑥
	❻	$P(E,F) = \sum_B P(E\|B)P(B,F) = \sum_B P(E\|B) \times$ ❹

Computation of *a posteriori* marginal probabilities is performed in a similar way, except that new informations (evidence) about some random variables are entered into cliques as if they were part of the product decomposition of $P(\mathcal{V})$.

3 Constructing a New DAG for Shafer-Shenoy

The aim of this Section is to provide a generic algorithm based on a DAG that produces precisely the same computations as those of the preceding Section. Note that, for the moment, this algorithm is not related to Pearl's method. This will be the topic of the next Section. Here, our purpose is only the construction of a new graph. This one is usually different from the original BN and, to explain how it can be derived from the latter, it is best mimicking Shafer-Shenoy's algorithm using the same elimination ordering as before.

Before any elimination occurs, the conditional probabilities of the product decomposition of $P(\mathcal{V})$ are stored in the nodes of the BN as shown in Fig. 3.a. As mentioned in the preceding Section, eliminating variable H (resp. I) amounts to substitute $P(H|E,F)$ (resp. $P(I|F,G)$) by $\sum_H P(H|E,F) = \mathbb{1}_{EF}$ (resp. $\sum_I P(I|F,G) = \mathbb{1}_{FG}$). Such operations can be performed on the BN by replacing the probabilities stored in H and I by $\mathbb{1}_{EF}$ and $\mathbb{1}_{FG}$ (Fig. 3.b). As shown in Table 1, eliminating E is achieved by computing $\mathbb{1}_{BF} = \sum_E P(E|B)\mathbb{1}_{EF}$. If a node in the BN had the knowledge of both $P(E|B)$ and $\mathbb{1}_{EF}$, it would be able to perform this operation. Unfortunately, $\mathbb{1}_{EF}$ and $P(E|B)$ are stored in nodes H and E respectively. Hence either a message containing $P(E|B)$ should be sent

to H or a message containing $\mathbb{1}_{EF}$ should be sent to E. In this paper, to decide between these alternatives, the following rule will always be applied :

Rule 1. *Assume an algebraic operation F on some matrices stored into nodes $X_{i_1}, \ldots, X_{i_k}$ of $\mathcal{V}$ needs to be performed. Let X_{i_p} be any node such that no X_{i_q}, $q \neq p$, is a descendant of X_{i_p}, i.e., X_{i_q} cannot be reached from X_{i_p} following a sequence of arcs (along their directions). Then all the X_{i_q}'s, $q \neq p$, will send to X_{i_p} a message containing the matrix they store, and X_{i_p} will perform F.*

Using rule 1, to mimic the elimination of node H, node E must send to H message $P(E|B)$ and H computes $\mathbb{1}_{BF} = \sum_E P(E|B)\mathbb{1}_{EF}$. H then replaces its own probability $\mathbb{1}_{EF}$ by $\mathbb{1}_{BF}$. As E sent its conditional probability, it need not store anything anymore. Similarly, when eliminating D, node D should send message $P(D|A)$ and G should substitute $P(G|D)$ by $P(G|A) = \sum_D P(D|A)P(G|D)$. The elimination of C leads to C sending message $P(C|A)$ to F and F replacing $P(F|C)$ by $P(F|A) = \sum_C P(F|C)P(C|A)$. Of course, neither C nor D should store a conditional probability anymore since they already transmitted their own (Fig. 3.c). The elimination of B should be performed by computing $\mathbb{1}_{AF} = \sum_B P(B|A)\mathbb{1}_{BF}$. As $\mathbb{1}_{BF}$ and $P(B|A)$ are stored in H and B respectively, by rule 1, B should send a message to H. This implies adding a new arc (B, H) as illustrated on Fig. 3.d. Moreover, as after sending its message to H, B does not store a matrix anymore, it will never send any other message, hence arc (B, E) can be safely removed. This is illustrated by representing (B, E) by a dashed arc. Eliminating A requires several messages sent to either H or G (here rule 1 cannot settle), H was chosen arbitrarily on Fig. 3.e. Finally, eliminating F can be performed either by H transmitting a message to I or the converse. This example justifies the following scheme for constructing the new DAG:

Algorithm 2 (construction of a directed secondary structure).
1. *Assign to every node X_k of the original BN a label $L(X_k) = \{X_k\} \cup \mathrm{Pa}_k$;*
2. *For every node X_k, in their order of elimination, let $\mathcal{V}_{X_k} = \{X_{k_1}, \ldots, X_{k_p}\}$ be the set of nodes the labels of which contain X_k. Select among $\mathcal{V}_{X_k}$ a node X_i according to rule 1 and, for every node X_j in $\mathcal{V}_{X_k} \backslash \{X_i\}$, add an arc (X_j, X_i) if necessary, remove the other arcs outcoming from X_j. Let $L(X_i) = \cup_{X_{k_j} \in \mathcal{V}_{X_k}} L(X_{k_j}) \backslash \{X_k\}$ and let $L(X_{k_j}) = \emptyset$ for all $k_j \neq i$.*

The following lemma summarizes this Section:

Lemma 1. *Shafer-Shenoy's inward pass can be precisely reproduced by constructing the DAG resulting from Algorithm 2, sending messages downward along the solid arcs of this DAG and computing new messages as described above, the latter being obtained by multiplying the messages received by a node by the conditional probability stored into the node.*

4 A New Variant of Pearl's Algorithm

Pearl's-like methods are applied on DAG such as a BN but, as they need singly-connected networks, i.e., graphs without loops, to produce correct answers, they

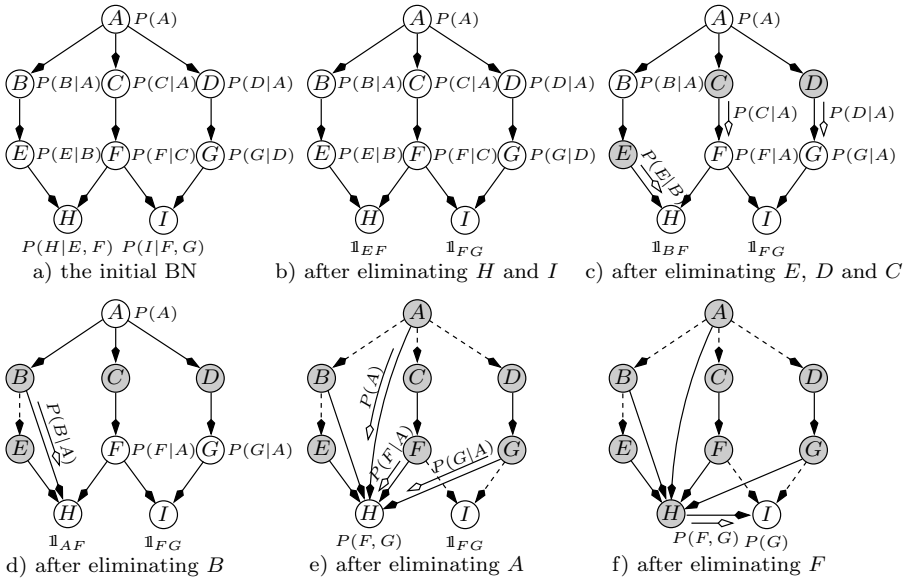


Fig. 3. The construction of a directed secondary structure.

often use a preprocess called *cutset conditioning* that transforms the BN into a singly-connected graph on which computations are then performed. The transformation advocated by [19] consists of applying the following algorithm:

Algorithm 3 (local cutset). *Let $(\mathcal{V}, \mathcal{A}, \mathcal{P})$ be a BN. Select some arcs in graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ the removal of which keeps the graph connected while removing all cycles. Assign to every remaining arc (X_i, X_j) initial label X_i. For each arc (X_k, X_j) removed, there still exists exactly one trail joining X_k to X_j, i.e., a sequence of arcs that, without taking into account their directions, can be followed to reach X_j from X_k. Add X_k to the label of every arc on this trail.*

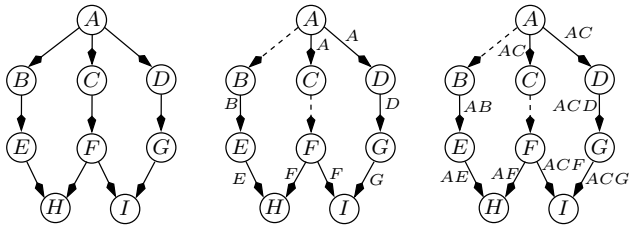
For instance, applying Algorithm 3 on the graph of Fig. 4.a results in the graph of Fig. 4.c: arcs (A, B) and (C, F) have been chosen arbitrarily to be removed. Fig. 4.b shows the initial labels of the remaining arcs. Trail B, E, H, F, I, G, D, A joins B and A, hence A is added to the label of every arc of this trail. Similarly, trail F, I, G, D, A, C joins F and C, thus C should be added to the label of every arc of this trail, hence resulting in Fig. 4.c. Once labels have been established, [19] advocates to use the following propagation algorithm:

Algorithm 4 (Pearl's-like method with local conditioning).

1. *Select an arbitrary node, say X_i, in the graph resulting from Algorithm 3 (for instance node I in the graph of Fig. 4.c).*
2. *inward pass: the current node asks its adjacent nodes for their messages. In turn, they recursively ask their other adjacent nodes for messages. When a node has received all the messages it waited for, it sends its own message.*

3. outward pass: after the inward pass, node X_i sends to its neighbors messages; they recursively send messages to their other adjacent nodes, and so on.

A message sent by a node X_j to one of its children (resp. parents) X_k is the sum over the variables not belonging to the label of arc (X_j, X_k) (resp. (X_k, X_j)) of the product of $P(X_j|\mathrm{Pa}_j)$ by all the messages sent to X_j except that sent by X_k.



a) a Bayesian network b) initial labels c) final labels

Fig. 4. Local conditioning and arc labeling.

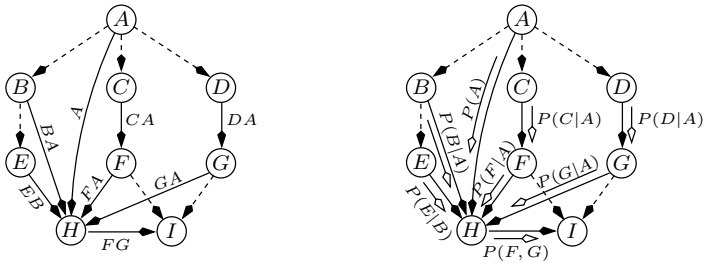
For instance, in Fig. 4.c, I would send a message to F equal to $\lambda(ACF) = \sum_{IG} P(I|F,G)\pi(ACG)$, where $\pi(ACG)$ is the message sent to I by G. Note that $\lambda(ACF)$ and $\pi(ACG)$ are messages of size $A \times C \times F$ and $A \times C \times G$ respectively. Thus arc labels indicate the size of messages sent throughout the network.

Now, let us come back to the unification of Pearl's and Shafer-Shenoy's algorithms. Applying the labeling algorithm below, which is a simple variant of Algorithm 3, to the secondary structure resulting from Algorithm 2, we obtain the graph of Fig. 5.a. It is striking that the labels correspond precisely to the size of the messages sent by Shafer-Shenoy, as described in the preceding Section.

Algorithm 5 (secondary structure labeling). *Let $\mathcal{G}$ be a BN and $\mathcal{G}'$ be the result of the application of Algorithm 2 to $\mathcal{G}$. For every arc (X_i, X_j) in $\mathcal{G}'$, assign label $\{X_i\}$ if it also belongs to $\mathcal{G}$, else $\emptyset$. For each arc (X_k, X_j) removed, add X_k to the label of every arc on the trail still joining X_j to X_k.*

The messages of the inward pass of Algorithm 4 performed on the graph of Fig. 5.a are precisely the same as those of Shafer-Shenoy. For instance, the message from H to I is equal to $\sum_{A,B,E,H} P(E|B) \times P(B|A) \times P(A) \times P(F|A) \times P(G|A) \times P(H|E,F) = \sum_A P(A)P(F|A)P(G|A) \times \sum_B P(B|A) \times \sum_E P(E|B) \times \sum_H P(H|E,F)$. The last sums are those of Shafer-Shenoy as they correspond to the messages and computations mentioned in Fig. 3. This suggests that Pearl can perform the same computations as Shafer-Shenoy when appropriately ordering its sequence of products and summations. This is, in essence, quite similar to the algorithm in [25], except that we use BN for inference rather than a secondary structure related only to computations and not to the original graph.

For the outward pass, Algorithm 4 will also produce messages of the same size as Shafer-Shenoy. However, if care is not taken, computations may be more time-consuming than in Shafer-Shenoy. For instance, assuming that I is selected



a) labels of the secondary structure e) Shafer-Shenoy's messages

Fig. 5. Arc labeling on the new Shafer-Shenoy's DAG.

at step 1, the outward pass will start by I sending to H message $\sum_I P(I|F,G) = \mathbb{1}_{FG}$. The message sent by H to G is:

$$\begin{aligned}\pi_{F,G} &= \textstyle\sum_{A,B,E,F,H} P(E|B)P(B|A)P(A)P(H|E,F)\mathbb{1}_{FG}P(F|A) &(1)\\ &= \textstyle\sum_{A,F} P(A)P(F|A)\mathbb{1}_{FG}\sum_B P(B|A)\sum_E P(E|B)\sum_H P(H|E,F) &(2)\\ &= \textstyle\sum_{A,F} P(A)P(F|A)\mathbb{1}_{FG}\mathbb{1}_{AF}, &(3)\end{aligned}$$

the latter being message ❶. Now, to actually perform the computations of (3) rather than those of (1), remind the sequence of messages sent to H, i.e., first $P(E|B)$ from E, then $P(B|A)$ from B, and finally $P(A)$, $P(F|A)$ and $P(G|A)$ from A, F and G. The corresponding products/sums performed when these messages were received are described in Table 2. Remark that (3) corresponds to multiplying $P(A)P(F|A)$ by the output of the penultimate computation (the third one) and by the message sent by I, and then to summing out unwanted variables. When H computes messages sent to F and A, the same element of the stack $\mathbb{1}_{AF}$ can be used to calculate $\sum_{A,G} P(A)P(G|A)\mathbb{1}_{FG}\mathbb{1}_{AF}$ and $\sum_{F,G} \mathbb{1}_{FG}\mathbb{1}_{AF}P(F|A)P(G|A)$. The former message corresponds to Shafer-Shenoy's ❷. The latter is never sent by Shafer-Shenoy, but it will eventually be computed to obtain the marginal probability of A, F or G as it corresponds to the product of the messages sent to clique AFG or, when multiplied by $P(A)$, to the product of the messages sent via separators AF, FG and AF. Similarly, when H computes the message sent to B, it should use the second element of the stack to avoid computing twice $\sum_H P(H|E,F)$. Thus, to behave as Shafer-Shenoy, Pearl should store in each node a stack of the temporary computations performed during the inward pass (see Table 2) and use this stack during the outward pass. Note that this does not actually require more space than in Shafer-Shenoy's algorithm since, in the latter, these temporary matrices are stored within separators.

Additional savings can be gained observing that once H has sent messages to A, F, G, the product of the messages it received from I and these nodes will be used for computing those sent to E and B. Thus, provided each time a node sends messages to some other nodes it keeps track of the product of the messages the latter sent it, it can be shown that Pearl will perform the same

Table 2. H's stack of temporary inward pass computations.

stack index	stack content	sender
1	$\mathbb{1}_{EF} = \sum_H P(H\|E,F)$	
2	$\mathbb{1}_{BF} = \sum_E P(E\|B)\mathbb{1}_{EF}$	E
3	$\mathbb{1}_{AF} = \sum_B P(B\|A)\mathbb{1}_{BF}$	B
4	$P(F,G) = \sum_A P(F\|A)P(G\|A)P(A)\mathbb{1}_{AF}$	A,F,G

computations as Shafer-Shenoy. For instance, once messages to A, F and G have been sent, node H can store $P(A,F) = \sum_G P(A)P(F|A)P(G|A)\mathbb{1}_{FG}$. Then it can compute $\sum_{A,F} P(A,F)\mathbb{1}_{BF}$, which is precisely the message it should send to B. Note that it also corresponds to the product of the messages sent to clique ABF by Shafer-Shenoy and that $\mathbb{1}_{BF}$ is the top of H's stack when matrix $\mathbb{1}_{AF}$ has been popped. Then H can store $P(B,F) = \sum_A P(B|A)P(A,F)$, and it can send to E message $\sum_F P(B,F)\mathbb{1}_{EF}$, which corresponds to the product of the messages sent to clique EBF by Shafer-Shenoy. Note again that $\mathbb{1}_{EF}$ is the top of H's stack once $\mathbb{1}_{BF}$ has been popped. Finally, messages sent to C and D correspond to marginalizations of the messages sent to cliques ADG and ACF. This justifies the following Proposition:

Proposition 1 (unification of Pearl and Shafer-Shenoy). *Let $\mathcal{G}$ be a BN. Let $\mathcal{G}'$ be the result of the application of Algorithms 2 and 5 on $\mathcal{G}$, according to an elimination ordering σ. Assign to each node X_i an empty stack $\mathcal{T}(X_i)$. Using the two passes below, Pearl performs the same computations as Shafer-Shenoy:*
Inward pass: *Messages are like in Algorithm 4. For each message sent by a node X_i, let $\mathcal{Q}$ be the union of $P(X_i|\mathrm{Pa}_i)$ and the set of messages received by X_i. Let S be the set of variables of these factors ordered according to σ. For every $X_k \in S$, select the factors in $\mathcal{Q}$ that contain X_k; remove them from $\mathcal{Q}$, multiply them and sum the result over X_k. Add the result to $\mathcal{Q}$ and to X_i's stack, and indicate which senders sent these factors.*
Outward pass: *Messages are like in Algorithm 4. For each node X_i in σ's reverse order, let M be the message received by X_i during the outward pass (if any). Compute messages to X_i's adjacent nodes as follows: Let S be the variables in the "sender" column at the top of X_i's stack $\mathcal{T}(X_i)$. For all nodes X_k in S, send a message to X_k equal to the sum over the variables not in the label of arc (X_k, X_i) of the product of M by the messages sent to X_i by nodes in $S\backslash\{X_k\}$ and the stack content of the element just under the top if it exists else $P(X_i|\mathrm{Pa}_i)$. After messages have been sent to all nodes in S, pop X_i's stack once. If the sender's column is empty, pop it again. Update M by multiplying it by the messages sent to X_i by all nodes in S and sum over the variables not belonging to any factor in $\mathcal{T}(X_i)$. Iterate the process until the stack is empty.*

5 Extension to Binary Join Trees and Lazy Propagation

It is well known that, in general, Shafer-Shenoy's algorithm is slower than Jensen's and that, to be competitive, it needs to be run in a binary join tree, that is, in

a tree where no node has more than 3 neighbors. Algorithms do exist to map a general join tree into a binary one [17]. The problem with general join trees comes from the outward phase and is illustrated on Fig. 6.a: assume that messages ①, ②, ③ and ④ were sent during the inward phase, then messages ❶, ❷, ❸ and ❹ of the outward phase are computed as:

$$❶ = \sum_F P(A) \times ① \times ② \times ③ \qquad ❷ = \sum_G P(A) \times ① \times ② \times ④,$$
$$❸ = \sum_A P(A) \times ① \times ③ \times ④ \qquad ❹ = \sum_G P(A) \times ② \times ③ \times ④.$$

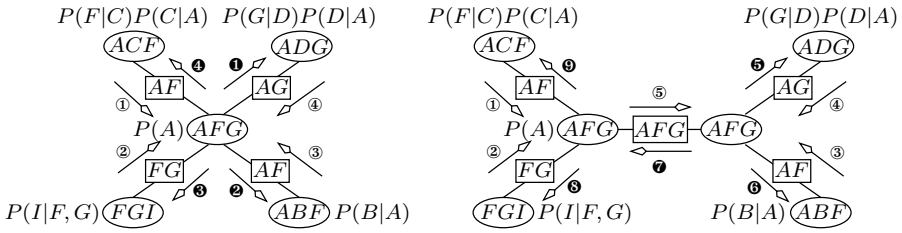


Fig. 6. Join trees vs. binary join trees.

Remark that some products are done several times. To avoid these redundancies, Shenoy proposes to modify the join tree so that no node has more than 3 neighbors (Fig. 6.b). Computations in this new structure are then:

$$⑤ = P(A) \times ① \times ② \qquad ❺ = \sum_F ③ \times ⑤ \qquad ❻ = \sum_G ④ \times ⑤,$$
$$❼ = ③ \times ④ \qquad ❽ = \sum_A P(A) \times ❼ \times ① \qquad ❾ = \sum_G P(A) \times ❼ \times ④.$$

Actually, message ❼ captures the idea that product ③ × ④ should not be performed several times. But avoiding these redundancies can also be obtained by observing that if we multiply during the inward phase the messages that arrive one by one and we store these results into a stack (see the left part of Table 3), then the outward phase messages can be computed by using this stack and multiplying by the messages on the right part of the Table. As is noticeable, from bottom up, these products can be computed incrementally requiring only one product for each line. This justifies the following proposition:

Proposition 2 (binary join tree unification). *Applying the algorithm of Proposition 1 with the rules below is equivalent to performing Shafer-Shenoy in a binary join tree:*
Inward pass: *When messages arrive to node X_i, perform the products one by one and store all of them into X_i's stack.*
Outward pass: *send outward messages to X_i's neighbor in the reverse order in which these neighbors sent their messages during the inward phase.*

The second extension we should mention is the unification with Lazy Propagation [10]. The latter is essentially similar to Jensen's or Shafer-Shenoy's algorithms in that it uses a join tree to perform propagation of potentials. However, it departs from these algorithms in the following manner:

Table 3. Binary join trees and the product of messages.

inward sending clique	inward phase products	outward phase products
initial potential	$P(A)$	$\times ❷ \times ❸ \times ❹ = ❾$
ACF	$P(A) \times ①$	$\times ❸ \times ❹ = ❽$
FGI	$P(A) \times ① \times ②$	$\times ❹ = ❻$
ABF	$P(A) \times ① \times ② \times ③$	$= ❺$

1. Instead of storing only one potential in each clique, it stores a list of potentials (or conditional probability tables). It performs products on some potentials only when necessary, i.e., when they contain a variable that is to be marginalized-out.
2. It recognizes summations like $\sum_H P(H|T)$ the result of which is known for sure to be 1.
3. It uses *d*-separation to avoid sending a message from one clique to another if the random variables of these cliques are independent due to some evidence.

In our algorithm, using *d*-separation is quite obvious since we are working on a BN, and avoiding computing unity summations is easy: it only requires to know which variables of the conditional probability tables are on the left of conditioning bars. As for the first feature, our algorithm can be easily adapted: it is sufficient manipulate lists of potentials instead of performing directly all the products. Hence, Pearl's algorithm can be adapted to behave as Lazy Propagation.

6 Conclusion

This paper has shown that Pearl's-like methods could be adapted using a new secondary directed structure and an appropriate ordering to perform the same computations as Shafer-Shenoy or Lazy Propagation, thus proving that directed inference methods could be as efficient as undirected ones. The advantage of using directed secondary structures is twofold: first, it enables to perform quite simply *d*-separation analyses and, secondly, it enables to limit the amount of information lost during triangulation by keeping the secondary structure as close as possible of the original one. For instance, there are cases in which even if the BN contains cycles, Pearl's algorithm can compute correctly all marginal *a posteriori* probabilities without needing any conditioning, e.g., when the parent nodes of the cycle sinks are independent. In such cases, working on a directed structure is better than working on an undirected one since the latter requires an unnecessary triangulation.

References

1. Cowell, R., Dawid, A., Lauritzen, S., Spiegelhalter, D.: Probabilistic Networks and Expert Systems. Statistics for Engineering and Information Science. Springer-Verlag (1999)

2. Jensen, F.: An introduction to Bayesian Networks. Taylor and Francis (1996)
3. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman (1988)
4. Allen, D., Darwiche, A.: New advances in inference by recursive conditioning. In: Proceedings of UAI. (2003) 2–10
5. D'Ambrosio, B., Shachter, R., Del Favero, B.: Symbolic probabilistic inference in belief networks. In: Proceedings of AAAI. (1990) 126–131
6. Darwiche, A., Provan, G.: Query DAGs: A practical paradigm for implementing belief network inference. In: Proceedings of UAI. (1996) 203–210
7. Dechter, R.: Bucket elimination: A unifying framework for several probabilistic inference algorithms. In: Proceedings of UAI. (1996) 211–219
8. Huang, C., Darwiche, A.: Inference in belief networks: A procedural guide. International Journal of Approximate Reasoning **15**(3) (1996) 225–263
9. Jensen, F., Lauritzen, S., Olesen, K.: Bayesian updating in causal probabilistic networks by local computations. Computational Statistics Quarterly **4** (1990) 269–282
10. Madsen, A., Jensen, F.: LAZY propagation: A junction tree inference algorithm based on lazy inference. Artificial Intelligence **113**(1–2) (1999) 203–245
11. Poole, D., Zhang, N.: Exploiting contextual independence in probabilistic inference. Journal of Artificial Intelligence Research **18** (2003) 263–313
12. Shafer, G.: Probabilistic expert systems. Society for Industrial and Applied Mathematics (1996)
13. Sharma, R., Poole, D.: Efficient inference in large discrete domains. In: Proceedings of UAI. (2003) 535–542
14. Zhang, N., Poole, D.: Inter-causal independence and heterogeneous factorization. In: Proceedings of UAI. (1994) 606–614
15. Shachter, R., Andersen, S., Szolovits, P.: Global conditioning for probabilistic inference in belief networks. In: Proceedings of UAI. (1994)
16. Lauritzen, S., Spiegelhalter, D.: Local computations with probabilities on graphical structures and their application to expert systems. The Journal of The Royal Statistical Society – Series B (Methodological) **50**(2) (1988) 157–224
17. Shenoy, P.: Binary join trees for computing marginals in the Shenoy-Shafer architecture. International Journal of Approximate Reasoning **17**(1) (1997) 1–25
18. Diez, F.: Local conditioning in Bayesian networks. Artificial Intelligence **87** (1996) 1–20
19. Faÿ, A., Jaffray, J.Y.: A justification of local conditioning in Bayesian networks. International Journal of Approximate Reasoning **24**(1) (2000) 59–81
20. Peot, M., Shachter, R.: Fusion and propagation with multiple observations in belief networks. Artificial Intelligence **48** (1991) 299–318
21. Geiger, G., Verma, T., Pearl, J.: Identifying independence in Bayesian networks. Networks **20** (1990) 507–534
22. Dawid, A., Kjærulff, U., Lauritzen, S.: Hybrid propagation in junction trees. In: Proceedings of IPMU94. (1994) 87–97
23. Kjærulff, U.: HUGS: Combining exact inference and Gibbs sampling in junction trees. In: Proceedings of UAI. (1995)
24. Rose, D.: Triangulated graphs and the elimination process. Journal of Mathematical Analysis and Applications **32** (1970) 597–609
25. Bloemeke, M., Valtorta, M.: A hybrid algorithm to compute marginal and joint beliefs in Bayesian networks and its complexity. In: Proceedings of UAI. (1998) 16–23