Local Conditioning in Bayesian Networks*

F. J. Díez†

Dpto. Inteligencia Artificial. UNED
Senda del Rey. 28040 Madrid. Spain

<fjdiez@dia.uned.es>

Abstract

Local conditioning (LC) is an exact algorithm for computing probability in Bayesian networks, developed as an extension of Kim and Pearl's algorithm for singly-connected networks. A list of variables associated to each node guarantees that only the nodes inside a loop are conditioned on the variable which breaks it. The main advantage of this algorithm is that it computes the probability directly on the original network instead of building a cluster tree, and this can save time when debugging a model and when the sparsity of evidence allows a pruning of the network. The algorithm is also advantageous when some families in the network interact through AND/OR gates. A parallel implementation of the algorithm with a processor for each node is possible even in the case of multiply-connected networks.

1 Introduction

A Bayesian network is an acyclic directed graph in which every node represents a random variable, together with a probability distribution such that

$$P(x_1, \ldots, x_n) = \prod_i P(x_i|pa(x_i)) \tag{1}$$

where x_i represents a possible value of variable X_i and $pa(x_i)$ is an instantiation of the parents of X_i in the graph. For a node/variable with no parents, the conditional probability is just its a priori probability. The essential independence property of Bayesian networks, called d-separation [22, 20], can be deduced from eq. (1). The basic inference problem consists of computing from the conditional probabilities the a posteriori probability $P(x|\mathbf{e})$ of a variable X given a certain evidence $\mathbf{e} \equiv \{``X_i = x_i", ``X_j = x_j", \ldots\}$. For singly-connected networks, there exists an elegant and efficient exact algorithm [15, 21], but the general case is much harder: the time complexity of exact algorithms heavily depends on the structure of the network — there is an example in the appendix— while the complexity of approximate methods depends mainly on the numerical values of conditional probabilities; for both exact and approximate methods, the problem is NP-hard [4, 5].

The best-known exact methods are clustering and conditioning. A version of the former, clique-tree propagation [19, 14], has become the standard algorithm for

Artificial Intelligence*, **87 (1996) 1–20.

†WWW: `http://www.dia.uned.es/~fjdiez`

inference in Bayesian networks. Conditioning, on the other hand, up until now had never been used in real expert systems. The purpose of this paper is to offer an efficient version of conditioning suitable for practical applications. The key idea in our approach consists of conditioning exclusively inside each loop. It is called *local* because every node has a specific conditioning, indicated by a list of variables.

The paper is organized as follows: the remainder of this introduction summarizes Kim and Pearl's [15, 21] algorithm for singly-connected networks and reviews three conditioning methods. Section 2 explains how to build an associated tree by removing some links and assigning a list of conditioning variables to each node, and section 3 derives the formulas for evidence propagation; we will try to clearly explain all the technical details for a straightforward implementation. The two possible versions of the algorithm are discussed in section 4. Section 5 compares local conditioning with other known methods and, finally, the conclusion offers suggestions for future research.

1.1 Algorithm for the polytree

The goal of the algorithm is to find the a posteriori probability $P(x|\mathbf{e})$, i.e., the probability of proposition "The value of variable X is x" given the observed evidence $\mathbf{e}$.

In a polytree, i.e. in a singly-connected network, an arbitrary node X divides the evidence into that connected to its causes, $\mathbf{e}_X^+$, and that connected to its effects, $\mathbf{e}_X^-$. Similarly, a link XY divides $\mathbf{e}$ into the evidence above the link, $\mathbf{e}_{XY}^+$, and that below it, $\mathbf{e}_{XY}^-$. This partition of evidence justifies the following definition of the messages propagated in the network (see fig. 1):

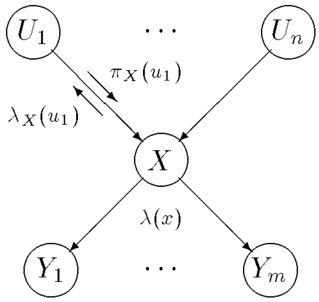

Figure 1: Evidence propagation by message-passing.

$$\pi(x) \equiv P(x, \mathbf{e}_X^+) \tag{2}$$

$$\lambda(x) \equiv P(\mathbf{e}_X^- \,|\, x) \tag{3}$$

$$\pi_X(u_i) \equiv P(u_i, \mathbf{e}_{U_i X}^+) \tag{4}$$

$$\lambda_{Y_j}(x) \equiv P(\mathbf{e}_{XY_j}^- \,|\, x) \tag{5}$$

These definitions are basically taken from [22], although equations (2) and (4) follow the modification introduced by Peot and Shachter [23] to simplify the computation in conditioning algorithms.

For a singly-connected network, d-separation results in two subsidiary properties [22]):

- Two children Y_i and Y_j of a node X are independent given the value of their parent:

$$P(y_i|\, x) = P(y_i|\, x, y_j)$$

- A parent U_i and a child Y_j of a node X are independent given the value of X:

$$P(u_i|\, x) = P(u_i|\, x, y_j)$$

Nevertheless, two parents of a node X, which in a polytree are always a priori independent, in general become correlated by the instantiation of X:

$$P(u_i|\, x) \neq P(u_i|\, x, u_j)$$

These independence properties lead to recursive expressions for computing the messages:

$$P(x|\mathbf{e}) = \alpha\, \pi(x)\, \lambda(x) \tag{6}$$

$$\pi(x) = \sum_{u_1,\ldots,u_n} P(x|\, u_1, \ldots, u_n) \prod_{i=1}^{n} \pi_X(u_i) \tag{7}$$

$$\lambda(x) = \prod_{j=1}^{m} \lambda_{Y_j}(x) \tag{8}$$

$$\pi_{Y_j}(x) = \pi(x) \prod_{k\neq j} \lambda_{Y_k}(x) \tag{9}$$

$$\lambda_{Y_j}(x) = \sum_{y_j} \left[\lambda(y_j) \sum_{v_1,\ldots,v_p} P(y_j|\, x, v_1, \ldots, v_p) \prod_{k=1}^{p} \pi_{Y_j}(v_k) \right] \tag{10}$$

where $V_1, \ldots, V_p$ are the causes of Y_j other than X, and $\alpha = [P(\mathbf{e})]^{-1}$ is a normalization constant to be computed after finding $\pi(x)$ and $\lambda(x)$.

1.2 An overview of three conditioning methods

Because of *d*-separation, the instantiation of a node which is not at the bottom of a loop —more precisely, a node whose two neighbors in the loop are not both its parents— breaks the loop in the sense that some correlations disappear and evidence can be propagated in the loop as if it were part of a polytree. Therefore, any conditioning method requires a structural process for finding a *cutset* (a set of nodes that break the loops in the network) and a numerical process which propagates evidence in the resulting tree. A cutset for the network in fig. 2*a* is $\{A, E\}$; other possible cutsets are $\{C\}$, $\{A, D\}$ and $\{B, E\}$. $\{F\}$ is not valid because it does not break loop C–D–F–E–C.

Pearl's algorithm [22, pp. 204–210] can be called *global conditioning* (GC) because it conditions every node in the network on every variable in the cutset. After initializing the network [28], it recursively computes the weights transmitted from every evidence node to all other nodes in the network. Consequently, the complexity of GC is not only exponential in the cutset size, but also proportional to the number of findings.

Peot and Shachter [23] introduced two important improvements in the original algorithm. They define a knot as a portion of the network that cannot be unconnected by removing one edge. (Unfortunately, they do not offer an algorithm for finding the knots of a graph.) Since every knot has its own cutset, the method can be called *knot conditioning* (KC). For example, fig. 2*a* consists of two knots: $\{A, B, C, D, E, F, G\}$, with its corresponding edges, and $\{H\}$; the cutset for the latter is the empty set. For

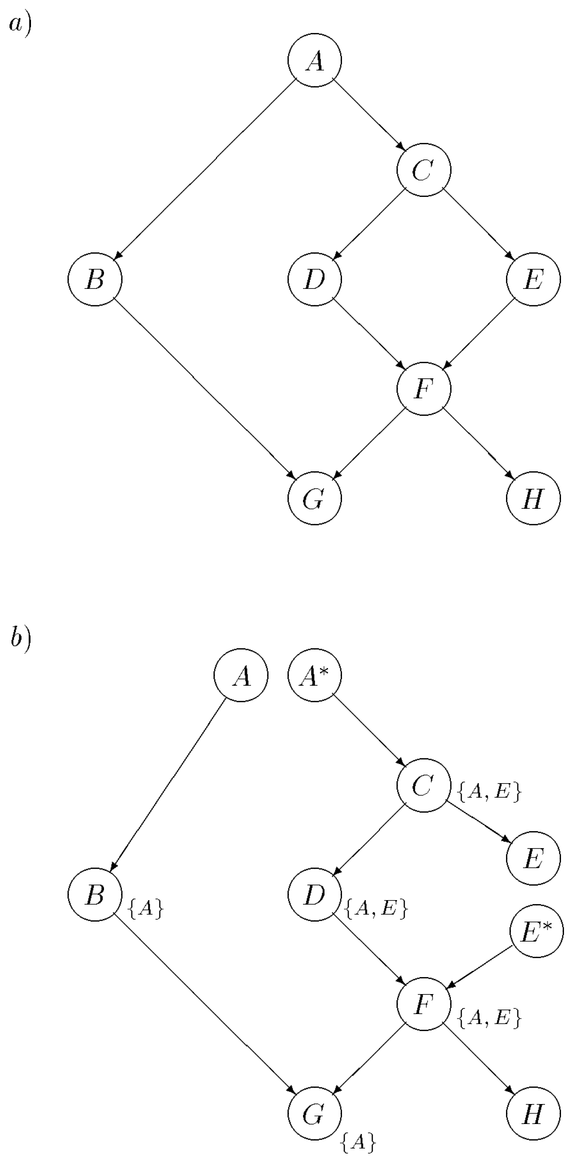

Figure 2: A network and its associated tree.

networks consisting of more than one knot, this is a first improvement in efficiency with respect to GC. The second one is a small change in the definition of $\pi(x)$ and $\pi_X(u_i)$ (see sec. 1.1) which allows the algorithm to fusion the influence of different findings without having to process the whole network for every evidence node. Therefore, the worst-case complexity of their algorithm is bounded by the maximum cutset size multiplied by the number of knots.

Local conditioning (LC), the algorithm introduced in this paper, goes a step further and, instead of considering the knots, applies conditioning exclusively within each loop; the term *local* means that there is a specific conditioning for each node. Fig. 2*b* displays an *associated tree* with a list of conditioning variables for every node. Observe that all the nodes in the path between a cutset node X and a phantom node X^* are conditioned on that variable and so every π- or λ-message flowing along this path will depend on x. Local conditioning includes the improvements introduced by Peot and Shachter and is much more efficient: there exist some structures for which KC has exponential complexity while LC only requires linear time (see sec. 5.1).

2 Associated tree

The process of building an associated tree encompasses two tasks: finding a cutset and assigning a list of conditioning variables to every node. Before reviewing previous algorithms for finding cutsets, this section introduces a new algorithm which performs both tasks at the same time and can even be integrated with evidence propagation (sec. 4). It consists of a depth-first search in the undirected graph as a means to detect the loops in the network.[1]

2.1 DFS, a depth-first search algorithm

The following example shows how to transform a Bayesian network (fig. 2*a*) into an associated tree (fig. 2*b*). The search begins at an arbitrary *pivot node* —A in our example— and travels through the network ignoring the direction of the edges, marking the nodes and including them in a list called PATH. A possible route goes through A, B, G, F, H, D, and C. Backtracking from a node H with no untraversed edges removes this node from PATH, such that this list always contains a path from the pivot node to the node currently investigated. For instance, at the moment of examining node C its value is (A, B, G, F, D, C).

When going forward from C to A, the fact that the latter is already marked denotes the presence of a loop. As a result, a phantom node A^* arises (fig. 2*b*) and a new edge A^*C replaces the original link AC, thus breaking the loop. Every node between A and A^*, i.e., every node in PATH from A to the end of the list, adds A to its own list of conditioning variables (see fig. 2*b*).

The search continues through E to F, which is marked, too. A new loop has appeared, but now the node F that has been reached is at the bottom of the loop. A way to make sure that the chosen node can break the loop is to condition on the variable at the top of the removed edge, E in this case. When detecting the second loop, the value of PATH is (A, B, G, F, D, C, E). Again, every node in PATH from F to the end of the list must add E to its list of conditioning variables. Then the algorithm backtracks to the pivot node and the search is over.

[1] We assume that the network is connected, as is the case for real-world models. Otherwise, the procedure should examine every connected part separately.

Clearly, the associated tree obtained by this method depends on the choice of pivot node and on the order in which the neighbors of every node are visited. Selecting as the pivot the node which breaks the most loops is a possible heuristic rule, but it is difficult to determine which neighbor must be visited first in order to achieve an efficient associated tree.

2.2 Other algorithms for finding cutsets

The problem of finding a minimal cutset is NP-hard [26]. However, the complexity of global conditioning does not depend directly on the size of the cutset, but is proportional to the product of the number of values of the variables in the cutset. Hence an algorithm attempting to minimize the product state space is in principle more accurate than an algorithm which simply seeks a cutset with few variables.

The heuristic algorithm A_1 proposed by Suermondt and Cooper [26] iteratively includes in the cutset the node that has the most neighbors, thus trying to obtain a small cutset; it only examines the number of values of the candidate nodes if there is a tie among them. For a network with n nodes, the algorithm visits every node up to $3n$ times, which results in a worst-case complexity of $O(n^2)$. Unfortunately, the cutset obtained may include unnecessary nodes not belonging to any loop.

For this reason, Stillman [25] designed A_2, a modified version of A_1, which usually finds a smaller cutset. He also showed that for a certain network with n nodes and an optimal cutset of only two nodes, both A_1 and A_2 yield cutsets of size $\Omega(\lfloor \frac{n}{4} \rfloor)$ —a disastrous result. Incidentally, DFS finds a cutset of only two or three nodes for that network, depending on the pivot node chosen. Furthermore, the complexity of DFS is $O(e)$ because it visits every edge only once, and this implies a saving of time with respect to A_1 and A_2 for sparse networks. Other heuristic algorithms can be found in [13].

Recently, Bar-Yehuda et al. [2] have offered a new approach to the cutset problem by reducing it to the vertex feedback set problem. During the reduction, a variable X_i in the Bayesian network taking on $|X_i|$ values gives way to a node in the undirected graph whose weight is $\log_2(|X_i|)$. The vertex feedback set of the undirected graph coincides with the cutset of the Bayesian network.

For a graph (for a Bayesian network), the *performance ratio* of a vertex feedback set (of a cutset) is the quotient between its total weight and the total weight of an optimal solution. Hence the product state size and, consequently, the complexity of global conditioning, grow exponentially with the cutset performance ratio. The performance ratio *of an algorithm* is the worst-case performance ratio among all networks with n nodes. From the example above, it is clear that the performance ratio of A_1 and A_2 cannot be better than $\Omega(\lfloor \frac{n}{4} \rfloor)$. The algorithms in [2] achieve performance ratios of $4 \log_2 n$ and $2d^2$ for general networks. Subsequently, Becker and Geiger [3] have designed an algorithm of time complexity $O(e + n \log n)$ and performance ratio equal to 2; there are theoretical reasons suggesting that it is virtually impossible to improve this result. Empirically, its average performance ratio is 1.22, an excellent result.

Although these algorithms limit themselves to finding cutsets, it is possible to extend them in order to obtain an associated tree suitable for local conditioning. First, every node X_0 breaking a loop X_0–X_1–…–X_n–X_0, must be split into a phantom node, connected only to either X_1 or X_n, and a "real" node maintaining the links with all its other neighbors, as shown in fig. 2; a node breaking l loops originates l phantom nodes. In the resulting tree, every node between X_0^* and X_0 will add this variable to its conditioning list. Alternatively, it would be possible to assign X_0 to every link X_iX_{i+1} (see sec. 5.1). The extended algorithms still have polynomial time complexity.

Nevertheless, neither the number of variables in the cutset nor their product state size are relevant for local conditioning. Even in the case of two cutsets such that $C_1 \subset C_2$, local conditioning might be more efficient for C_2 than for C_1. Therefore, it would be desirable to design algorithms leading to efficient computations *in local conditioning* instead of adapting the algorithms developed for global conditioning. From similarity with other problems arising in graph theory and Bayesian networks, we make the conjecture that it is NP-hard to find an optimal associated tree for local conditioning.

3 Propagation of evidence

Local conditioning propagates evidence on an associated tree generated by some of the above algorithms. As an example, let fig. 3 represent a portion of a tree for a network whose only loop was A–B–D–C–A. Since it is in fact a singly-connected network, the definitions of $\mathbf{e}_X^+$, $\mathbf{e}_X^-$, $\mathbf{e}_{XY}^+$ and $\mathbf{e}_{XY}^-$ introduced in section 1.1 are still valid. In our example, $\mathbf{e} = \mathbf{e}_A^+ \cup \mathbf{e}_A^-$, $\mathbf{e}_A^- = \mathbf{e}_1 \cup \mathbf{e}_{AB}^-$, $\mathbf{e}_{AB}^- = \mathbf{e}_{VB}^+ \cup \mathbf{e}_2 \cup \mathbf{e}_{BD}^-$, etc.

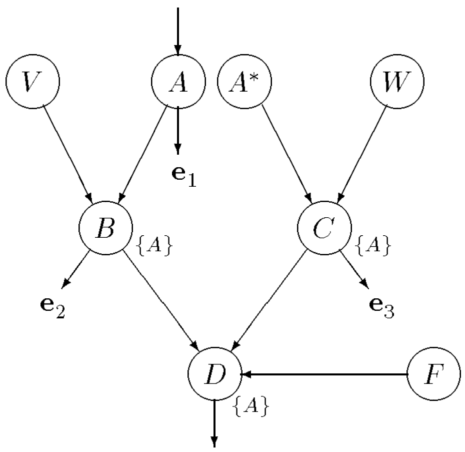

Figure 3: A portion of an associated tree.

3.1 Propagation from a phantom node

The computation of $\pi(a)$ is straightforward since we assumed there was no loop above A. The value of $\lambda(a)$ is, according to its definition,

$$\lambda(a) = P(\mathbf{e}_A^-|a) = P(\mathbf{e}_1|a)\; P(\mathbf{e}_{AB}^-|a) = \lambda_{\mathbf{e}_1}(a)\; \lambda_B(a). \tag{11}$$

In the associated tree, the instantiation of A breaks the loop, such that

$$\begin{aligned}\lambda_B(a) &= P(\mathbf{e}_{AB}^-|a) = \sum_{b,v} P(\mathbf{e}_B^-, \mathbf{e}_{BV}^+, b, v|a) \\ &= \sum_{b,v} P(\mathbf{e}_B^-|\mathbf{e}_{BV}^+, b, v, a)\; P(b|\mathbf{e}_{BV}^+, v, a)\; P(\mathbf{e}_{VB}^+|v, a)\; P(v, a)\,.\end{aligned}$$

Because of d-separation, some factors inside the summatory are simplified:

$$\lambda_B(a) = \sum_{b,v} P(\mathbf{e}_B^-|b,a)\, P(b|v,a)\, P(\mathbf{e}_{VB}^+|v)\, P(v)$$

Let $\lambda(b|a)$ and $\pi_B(v)$ be

$$\lambda(b|a) \equiv P(\mathbf{e}_B^-|b,a) \tag{12}$$

$$\pi_B(v) \equiv P(v, \mathbf{e}_{VB}^+) = P(\mathbf{e}_{VB}^+|v)\, P(v) \tag{13}$$

Then the expression for $\lambda_B(a)$ is equal to:

$$\lambda_B(a) = \sum_b \left[\lambda(b|a) \sum_v P(b|v,a)\, \pi_B(v)\right] \tag{14}$$

Hence we need $\lambda(b|a)$. The definition of $\lambda_D(b|a)$

$$\lambda_D(b|a) \equiv P(\mathbf{e}_{BD}^-|b,a) \tag{15}$$

leads to

$$\lambda(b|a) = \lambda_{\mathbf{e}_2}(b)\, \lambda_D(b|a). \tag{16}$$

The computation of $\lambda_D(b|a)$ is similar to that of $\lambda_B(a)$:

$$\begin{aligned}
\lambda_D(b|a) &= \sum_{d,c,f} P(\mathbf{e}_D^-, \mathbf{e}_{CD}^+, \mathbf{e}_{FD}^+, d, c, f|b,a) \\
&= \sum_{d,c,f} P(\mathbf{e}_D^-|d)\, P(d|b,c,f)\, P(\mathbf{e}_{CD}^+|c,a)\, P(c|a)\, P(\mathbf{e}_{FD}^+|f)\, P(f) \\
&= \sum_d \left[\lambda(d) \sum_{c,f} P(d|b,c,f)\, \pi_D(c|a)\, \pi_D(f)\right],
\end{aligned} \tag{17}$$

where[2]

$$\pi_D(c|a) \equiv P(c, \mathbf{e}_{CD}^+|a). \tag{18}$$

The value of this message is

$$\begin{aligned}
\pi_D(c|a) &= P(\mathbf{e}_3|c)\, P(\mathbf{e}_C^+, c|a) \\
&= \lambda_{\mathbf{e}_3}(c)\, \pi(c|a)
\end{aligned} \tag{19}$$

where

$$\pi(c|a) \equiv P(c, \mathbf{e}_C^+|a) \tag{20}$$

$$\begin{aligned}
&= \sum_w P(\mathbf{e}_{WC}^+, c, w|a) \\
&= \sum_w P(c|w,a)\, \pi_C(w).
\end{aligned} \tag{21}$$

The removal of link AC from the original network means that it is not necessary to pass a message $\lambda_C(a)$, because evidence $\mathbf{e}_3$ has already been propagated up to A through C–D–B–A. Thus, messages do not flow on the whole network but on the associated tree, where there exists exactly one path from each piece of evidence to each node.

[2] We cannot define a $\pi_D(c|a)$ normalized for c because the normalization "constant" $\alpha = \left[\sum_c \pi_D(c|a)\right]^{-1}$ does depend on a and so it would disturb the computation of $P(a|\mathbf{e})$. This is the reason for using Peot and Shachter's definitions of $\pi(x)$ and $\pi_X(u_i)$ instead of Pearl's (see sec. 1.1).

3.2 Propagation towards a phantom node

Now messages must flow in the opposite direction, i.e. collecting all the evidence above A and transmitting its effect towards A^* so that every node can compute its probability. Note that messages in this section are *orthogonal* to those in the previous one, in the sense that they can be computed independently.

For node B, message $\lambda(b|a)$ was given by eq. (16). So we need

$$\pi(b,a) \equiv P(b, \mathbf{e}_B^+, a) \tag{22}$$

in order to compute

$$\begin{aligned} P(b|\mathbf{e}) &= \sum_a P(b,a|\mathbf{e}) = \alpha \sum_a P(b,a,\mathbf{e}) \\ &= \alpha \sum_a \lambda(b|a)\, \pi(b,a)\,. \end{aligned} \tag{23}$$

As we did before:

$$\pi(b,a) = \sum_v P(b|v,a)\, \pi_B(v)\, \pi_B(a) \tag{24}$$

$$\pi_B(a) \;=\; \lambda_{\mathbf{e}_1}(a)\, \pi(a). \tag{25}$$

In the same way

$$\pi(d,a) = \sum_{b,c,f} P(d|b,c,f)\, \pi_D(b,a)\, \pi_D(c|a)\, \pi_D(f)\,, \tag{26}$$

where

$$\pi_D(b,a) \equiv P(b,a,\mathbf{e}_{BD}^+) \tag{27}$$

$$= \lambda_{\mathbf{e}_2}(b)\, \pi(b,a). \tag{28}$$

Clearly,

$$P(d|\mathbf{e}) = \alpha \sum_a \lambda(d)\, \pi(d,a). \tag{29}$$

The computation of $P(c|\mathbf{e})$ requires a $\lambda(c;a)$ such that

$$P(c|\mathbf{e}) = \alpha \sum_a \lambda(c;a)\, \pi(c|a)\,. \tag{30}$$

Message $\lambda(c;a)$ happens to be harder to define than to compute. With a suitable partition of evidence,

$$P(c|\mathbf{e}) = [P(\mathbf{e})]^{-1} \sum_a P((\mathbf{e}_C^- \backslash \mathbf{e}_{AB}^+)|c,a)\, P(c,\mathbf{e}_C^+|a)\, P(a,\mathbf{e}_{AB}^+)$$

and comparing the last two expressions, the definition of $\lambda(c;a)$ turns out to be

$$\lambda(c;a) \equiv P((\mathbf{e}_C^- \backslash \mathbf{e}_{AB}^+)|c,a)\, P(a,\mathbf{e}_{AB}^+) \tag{31}$$

Note that the intuitive definition "$\lambda(c;a) \equiv P(\mathbf{e}_C^-, a|c)$" would not be correct because $P(\mathbf{e}_C^-, a|c) = P((\mathbf{e}_C^- \backslash \mathbf{e}_{AB}^+)|c,a)\, P(a,\mathbf{e}_{AB}^+|c)$. Message $\lambda_D(c;a)$ must propagate the influence of $\mathbf{e}_{CD}^-$ on C along the path A–B–D–C for every value a in a way such that the instantiation of C does not modify the probability of A. For this reason $\mathbf{e}_C^-$ was decomposed into two sets: the evidence between C and A, $\mathbf{e}_C^- \backslash \mathbf{e}_{AB}^+ = \mathbf{e}_C^- \cap \mathbf{e}_{AB}^-$, and the evidence above link AB.

Then, the value of $\lambda(c;a)$ is

$$\lambda(c;a) = \lambda_{e_3}(c)\,\lambda_D(c;a) \tag{32}$$

where

$$\begin{aligned}
\lambda_D(c;a) &\equiv P((\mathbf{e}^-_{CD}\backslash\mathbf{e}^+_{AB})|c,a)\,P(a,\mathbf{e}^+_{AB}) &(33)\\
&= \sum_{d,f,b} P(\mathbf{e}^-_D,\mathbf{e}^+_{DF},(\mathbf{e}^+_{BD}\backslash\mathbf{e}^+_{AB}),d,b,f|c,a)\,P(a,\mathbf{e}^+_{AB})\\
&= \sum_{d,f,b} P(\mathbf{e}^-_D|d)\,P(d|b,c,f)\,P(\mathbf{e}^+_{DF},f)\,P((\mathbf{e}^+_{BD}\backslash\mathbf{e}^+_{AB})|a)\,P(a,\mathbf{e}^+_{AB})\\
&= \sum_d \lambda(d)\left[\sum_{b,f} P(d|b,c,f)\,\pi_D(b,a)\,\pi_D(f)\right]. &(34)
\end{aligned}$$

3.3 Discussion

The lists of conditioning variables (LCV) control the dimension of messages. The presence of A in $LCV(B)$, $LCV(C)$ and $LCV(D)$ prevents these nodes from summing on a the messages they send and receive. In contrast, message $\lambda_D(f)$ bears no extra conditioning because $LCV(F)=\emptyset$; the dependency on a is eliminated by summing on this variable:

$$\lambda_D(f) = \sum_{a,d}\left[\lambda(d)\sum_{b,c} P(d|b,c,f)\,\pi_D(b,a)\,\pi_D(c|a)\right]. \tag{35}$$

The above equations indicate that every node conditioned on A receives a message that originated at "real" node A and is, roughly speaking, proportional to $P(a)$, and a message from every phantom node A^*, conditioned on a. Eqs. (26) and (35), for instance, fusion a message $\pi_D(b,a)$, "proportional" to $P(a)$, and a message $\pi_D(c|a)$, conditioned on a.

In the implementation of the algorithm, a node need not care about which message is "proportional to a" and which one is "conditioned on a". It is thus not necessary to have different implementations for $\lambda_Y(x)$, $\lambda_Y(x;z)$ and $\lambda_Y(x|z)$. Eqs. (14), (17) and (34) are **formally equivalent** to eq. (10) for the polytree: the only difference is that messages corresponding to a loop are generally arrays instead of vectors. As a consequence, only four functions (or four message handlers, in Object-Oriented Programming) are required: $\pi(X)$, $\lambda(X)$, $\pi_Y(X)$ and $\lambda_Y(X)$.[3]

Nevertheless, the distinction between "proportional to $P(a)$" and "conditioned on a" is necessary from a mathematical point of view. Other presentations of conditioning methods, although describing correct algorithms, fail to define the π- and λ-messages properly. In the case of the example above, ref. [23] would give the correct formula for $P(b|\mathbf{e})$ but, instead of expressing eq. (30) accurately, it would read

$$P(c|\mathbf{e}) = \alpha\sum_a \pi(c,a)\,\lambda(c|a)\,.$$

In turn, ref. [6] contains incorrect expressions, such as

$$P(x,\mathbf{e}) = \sum_u \pi(x|u)\,\lambda(x|u)\,.$$

[3] *Notation:* $\pi(X)$ can represent a vector $\pi(x)$ as well as an array such as $\pi(x,a|b)$.

4 Two versions of local conditioning

Local conditioning, as any other exact algorithm, consists of two processes, numerical and structural, which have been described in sections 2 and 3 respectively. We return now to the example in fig. 2 to demonstrate that, instead of having two independent phases, it is possible to extend the DFS algorithm so that it propagates evidence at the same time as it finds the associated tree.

Starting at the same pivot node A as in section 2.1 and crossing the edges in the same order, the integrated algorithm successively requests $\lambda_B(A)$, $\lambda_G(B)$, $\pi_G(F)$, $\lambda_H(F)$ —which immediately returns vector $\lambda_H(f)$—, $\pi_F(D)$, $\pi_D(C)$ and $\pi_C(A)$. Node A is marked as "visited". Therefore edge AC is removed, with the consequent update of the conditioning lists of some nodes, and message $\pi(C)$ turns out to be $\pi(c|a)$. In the implementation, it is not necessary to create phantom nodes: they were just a crutch for deriving the formulas.

The following step is to compute $\lambda_E(C)$ and $\lambda_F(E)$. Since F is also a visited node, every node in the new loop adds E to its list of conditioning variables; message $\lambda_F(E)$ is empty because of the removal of edge EF. Subsequently, messages $\lambda_E(c;e)$, $\pi_D(c,e|a)$, $\pi_F(d,e|a)$ and $\pi(f,e|a)$ propagate backwards to the pivot node.[4] As G does not belong to the loop broken by E, G can sum over e and propagation goes on with $\pi_G(f|a)$, $\lambda_G(b|a)$ and $\lambda_B(a)$.

This first pass generates an associated tree. During the second pass, a message passes through every edge of the tree in the reverse direction; this way, every node receives all the information necessary to compute its probability. These two passes are respectively equivalent to *collect evidence* and *distribute evidence* in other Bayesian networks algorithms.

The main disadvantage of the integrated version of LC is that it finds the cutset by using depth-first search, in which there is hardly any room for applying heuristics, instead of using the much more efficient algorithms discussed in section 2.2. However, when only one evaluation of a Bayesian network is required —because the network will change or because it was generated for a specific query (sec. 5.2)— it may compensate to use the integrated version instead of spending time in finding an efficient tree.

On the other hand, a distributed implementation of local conditioning has to previously generate the associated tree, so that evidence propagation can begin at every end node simultaneously. This allows a parallel implementation of local conditioning with a processor for every node, as was possible for polytrees [21, 9]. The combination of clustering and conditioning for distributing evidence propagation is discussed in [13] and [24].

5 Comparison to other methods

5.1 Conditioning algorithms

Section 1.2 was an overview of three conditioning methods: Pearl's [22] global conditioning (GC), Peot and Shachter's [23] knot conditioning (KC) and local conditioning (LC). For a given network, KC is at least as efficient as GC; in general, the latter is much more efficient than the former. In the same way, LC is at least as efficient as KC since, given a knot and its corresponding cutset, LC never conditions on that

[4] Note that e and a play different roles in these equations because evidence is traveling from a "real" node E and from a phantom node A^*.

cutset the variables outside the knot. For a knot consisting of several adjacent loops, LC is much more efficient than KC because it does not accumulate the conditioning of the different loops that form the knot.

For instance, the square ladder (fig. 4) is a single knot containing N adjacent loops. In KC, the whole network is conditioned on the cutset, whose size is $O(n)$, where $n = 2N+2$ is the number of nodes; as a consequence, the complexity grows exponentially with n. In contrast, given that every node belongs to at most two loops, independently of the number of squares in the ladder, LC achieves linear complexity for this structure. The same difference holds for the diamond ladder (see fig. 5 in the appendix) and for many other networks formed by adjacent loops.

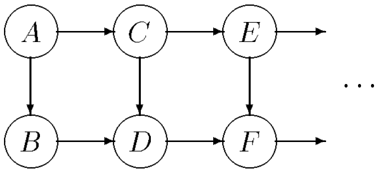

Figure 4: The square ladder contains N squares and $2N + 2$ nodes.

Recently, Darwiche [6] has proposed a new method based on relevant and local cutsets; $\mathbf{R}_X^+$, $\mathbf{R}_X^-$, $\mathbf{R}_{XY}^+$ and $\mathbf{R}_{XY}^-$ are the relevant cutsets for $\pi(x)$, $\lambda(x)$, $\pi_Y(X)$ and $\lambda_Y(X)$ respectively. Unfortunately, the algorithms he proposes for finding relevant cutsets produce very ineffective results that ruin the performance of his conditioning algorithm. An example in [6] applies a cutset $\mathbf{R}_X^+$ containing six variables where only one was actually necessary. For binary variables, it amounts to requesting and computing every message $\pi_X(u_3, u_1)$ 2^5 times. Local cutsets are a partial remedy to this problem: an optimal cutset indicates that it is possible to retrieve from a cache the message $\pi(u_3)$ used in every computation of $\pi_X(u_3, u_1)$; but again, a suboptimal local cutset results in computing $\pi(u_3)$ several times instead of just retrieving the value obtained for the first request.

The problem of finding the tightest cutsets (for a certain global cutset) can be solved by considering an associated tree with phantom nodes, instead of absorbing arcs: section 2.2 showed how to assign a list of conditioning variables $LCV(XY)$ to each link XY. But in local conditioning it suffices to have a list of conditioning variables $LCV(X)$ at each node, because any message originating at X is conditioned, at most, on $LCV(X) \cup \{X\}$, and any message arriving at node Y is summed over the variables not included in $LCV(Y) \cup \{Y\}$. Therefore LC is implicitly using the tightest relevant cutsets $\mathbf{R}_{XY}^+$ and $\mathbf{R}_{XY}^-$ searched by Darwiche:

$$\mathbf{R}_{XY}^+ = \mathbf{R}_{XY}^- = LCV(XY) = (LCV(X) \cup \{X\}) \cap (LCV(Y) \cup \{Y\}) \tag{36}$$

Clearly, the distinction between $\mathbf{R}_{XY}^+$ and $\mathbf{R}_{XY}^-$ was not necessary.

With regard to $\mathbf{R}_X^+$, eq. (7) indicates that messages involved in the computation of $\pi(x)$ are conditioned on $\bigcup_i LCV(U_i X)$; on the other hand, $\pi(x)$ only needs the conditioning affecting the portion of the network below X. A similar reasoning for $\lambda(x)$ leads to[5]

$$\mathbf{R}_X^+ = \mathbf{R}_Y^- = \left(\bigcup_i LCV(U_i X) \,\cap\, \bigcup_j LCV(XY_j) \right) \backslash \{X\} \tag{37}$$

[5] Variable X does not belong to $\mathbf{R}_X^+$ and $\mathbf{R}_X^-$ because relevant cutsets are meant to only keep track of the extra conditioning for $\pi(X)$ and $\lambda(X)$.

In contrast,

$$LCV(X) = \left(\bigcup_i LCV(U_iX) \cup \bigcup_j LCV(XY_j) \right) \setminus \{X\} \tag{38}$$

Since $\mathbf{R}_X \subseteq LCV(X)$ —it is not necessary to differentiate between $\mathbf{R}_X^+$ and $\mathbf{R}_X^-$— this cutset may reduce the number of conditioning variables in $\pi(X)$ and $\lambda(X)$. In fig. 3, for instance, $LCV(D) = \{A\}$, while $\mathbf{R}_D = \emptyset$; thus eqs. (26) and (29) should be replaced by

$$\pi(d) \;=\; \sum_{a,b,c,f} P(d|b,c,f)\, \pi_D(b,a)\, \pi_D(c|a)\, \pi_D(f) \tag{39}$$

$$P(d|\mathbf{e}) = \alpha\, \lambda(d)\, \pi(d) \tag{40}$$

which require fewer computations. Therefore, a worthy refinement of local conditioning based on Darwiche's work consists of using $\mathbf{R}_{XY} = LCV(XY)$ (assigned by the algorithms in sec. 2) and $\mathbf{R}_X$ (given by eq. (37)) instead of using a list of conditioning variables for each node.

In summary, the main problem in [6] was that suboptimal cutsets drastically degraded the performance of the algorithm. When using the tightest relevant cutsets (we have explained how to find them), it computes essentially the same information as local conditioning, with relevant cutsets playing the role of lists of conditioning variables. The only difference is that, instead of propagating messages conditioned on several variables, Darwiche's algorithm requests every message several times with different instantiations of the relevant cutset.

5.2 Clustering methods

The most widely used algorithm for multiply-connected networks is clique-tree propagation [19, 14]. The structural phase, sometimes called "compilation of the network", consists of triangulating the graph and forming a tree of cliques. The triangulation is the crucial step that determines the efficiency of the numerical phase. Unfortunately, it is NP-hard to find an optimal clique tree [29].

Shachter, Andersen and Szolovits [24] demonstrated that global conditioning is equivalent to computing probability in a specific cluster tree, or put another way, equivalent to using a particular triangulation. In local conditioning, the equivalent cluster tree is formed by removing phantom nodes and replacing every node X_i with a cluster C_i containing variable X_i, its parent variables and its conditioning variables: $C_i = \{X_i\} \cup pa(X_i) \cup LCV(X_i)$. An empirical comparison should determine if the algorithms in section 2 can produce cluster trees as efficient as those obtained by heuristic search [16, 18] or simulated annealing [17, 29].

In contrast, for some triangulations there is no equivalent associated tree. Therefore, the main advantage of clustering is that it can use the most efficient triangulation, while local conditioning has a quite restricted choice. If the network is given once and for all, the time spent in building an efficient cluster tree usually pays off when propagating evidence.

Nevertheless, the integrated version of local conditioning computes the probability directly on the original network and hence can save time in the following situations:

- When a knowledge engineer is debugging a model and wants to measure the impact of some modifications.

- When there is sparce evidence. In medicine, for example, only a few tests are performed for each patient and doctors usually provide the expert system with only a small part of the potential findings of those tests. Thus the pruning of unobserved nodes reduces the size of the network and may eliminate many loops [1].

- When the network is generated dynamically. For instance, a Bayesian network can answer queries such as "What is the probability of $P((x_2 \vee x_3) \wedge x_6)$?" by introducing new nodes interacting through AND/OR gates [22, pp. 223ff]. Natural language processing [11], information retrieval [10], planning and temporal reasoning are some of the fields in which applications usually generate a specific network for each situation.

In addition, local conditioning is able to treat the AND/OR–gates and their generalization directly: the expressions for Q, π and λ in [7, 9] do not change when applying conditioning. Instead, clustering algorithms have to add dummy nodes [12] even in the case of a polytree, reducing the efficiency of evidence propagation and hindering the explanation of reasoning.

6 Implementation

DIAVAL [8] is a prototype expert system for echocardiography whose network contains over 200 nodes and several loops. The computation of evidence using the integrated version of local conditioning takes less than 5 seconds on a 486/66Mz computer. This is a satisfactory result, considering that the current version was implemented in Common Lisp without much concern for efficiency. The computational time is significantly reduced when the algorithm neglects leaf nodes for which there is no observed evidence, since it usually eliminates a considerable number of loops. A future implementation in C++ is expected to achieve a much higher performance.

7 Conclusions and future work

We have introduced local conditioning as a new exact algorithm for inference in Bayesian networks, discussing the advantages and disadvantages of its two versions. Although the algorithm is already in use in a prototype expert system, the assessment of its suitability for real-world applications requires an empirical comparison of at least three methods:

1. The integrated version of local conditioning, which builds the tree as it propagates evidence (sec. 4).

2. Local conditioning with different heuristic algorithms for building associated trees.

3. Clustering algorithms with different triangulation techniques [16, 17, 18, 29].

With regard to the second method, it would be of interest to prove or refute the conjecture that it is NP-hard to find an optimal associated tree. If it is confirmed, future research should either extend the algorithms developed for global conditioning (sec. 2.2), or design new algorithms suitable for local conditioning.

As mentioned above, every associated tree is equivalent to a specific cluster tree (i.e. to a certain triangulation), but the opposite is not always true. Therefore, if the optimal triangulation of a network is not equivalent to any associated tree, clustering algorithms using techniques for finding near-optimal triangulations (third method) will outperform local conditioning based on heuristic algorithms (second method).

On the other hand, local conditioning based on depth-first search (first method) has the virtue of propagating evidence in the original network, without any previous processing. In a singly-connected portion of a Bayesian network, it coincides with Kim and Pearl's polytree algorithm [15, 21], the most efficient solution. In the presence of loops, the question is whether the time spent in searching an efficient associated tree or an efficient cluster tree will compensate when performing the numerical computation of probability. Two determinant factors are the number and length of the loops and the amount of available evidence, since the pruning of the network may eliminate a great number of loops when there are few findings. The use of AND/OR-gates in the model is an additional reason to use local conditioning.

Acknowledgements

A first version of this paper was written at the Dept. of Computer Science, University of California Los Angeles, with research facilities provided by Prof. Pearl, and appeared as Technical Report R–181, Cognitive Systems Laboratory, UCLA, in July 1992. The work was directed by Prof. José Mira as thesis advisor and supported by a grant from the Plan Nacional de Formación de Personal Investigador of the Spanish Ministry of Education and Science. The author wishes to thank Moisés Goldzsmidt, David Heckerman, Judea Pearl, Mark Peot, Ross Shachter and an anonymous referee for helpful discussions.

Appendix: The complexity of exact algorithms

Since probabilistic inference is NP-hard for multiply-connected Bayesian networks [4, 5], no algorithm is expected to have polynomial time complexity for the general case. Nevertheless, for a particular network, the complexity of exact algorithms depends as much on its topology as on the number of nodes. The study of a particular structure, the diamond-ladder (fig. 5), will illustrate this point.

A possible way to form the associated tree is to remove every link $D_{i-1}C_i$, for $1 \leq i \leq N$; DFS can produce this tree by choosing D_0 as the pivot node and covering the network in the order D_0, B_1, D_1, C_1, B_2, etc. For the i-th diamond, $\pi_{D_i}(B_i)$ and $\pi_{D_i}(C_i)$ are conditioned on D_{i-1}, but $\lambda(D_i)$ is simply a vector, so conditioning does not accumulate. Table I displays the number of mathematical operations (namely additions, multiplications and a few divisions) required for every expression, in the case of binary variables. Therefore, the number of operations is 123 for every diamond. The computation of $P(D_0|\mathbf{e})$ takes five more operations. If the ladder has N diamonds, the number of nodes is $n = 3N + 1$ and the total number of operations is $123N + 5 = 41n - 36$.

For this example, clustering methods have the same order of complexity. Suermondt and Cooper [27] calculated that, besides the compilation of the network, clique-tree propagation needs $330N + 111 = 110n - 119$ operations for evidence propagation in this structure, although this is likely an overestimation. The clustering algorithm proposed in [24] needs only $84N + 5 = 28n - 33$ operations, as can be deduced by examining the M_{ij}-messages.

In order to highlight how much the complexity of exact algorithms depends on

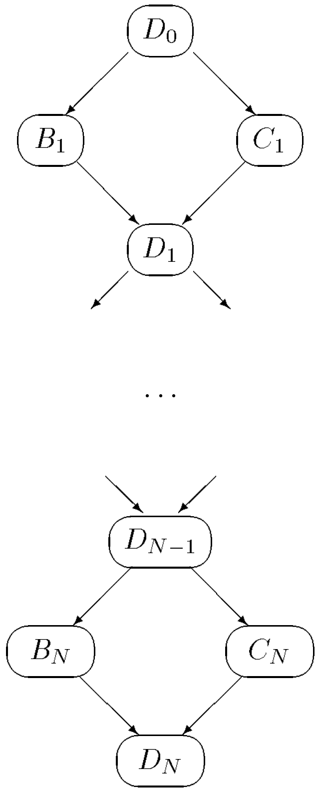

Figure 5: Diamond-ladder structure.

From D^*_{i-1} towards D_{i-1}	$\pi(c_i \mid d_{i-1})$	0
	$\pi_{D_i}(c_i \mid d_{i-1})$	6
	$\lambda_{D_i}(b_i \mid d_{i-1})$	10
	$\lambda_{B_i}(d_{i-1})$	6
From D_{i-1} towards D^*_{i-1}	$\pi_{B_i}(d_{i-1})$	4
	$\pi(d_i)$	46
	$\lambda_{D_i}(c_i; d_{i-1})$	28
Probability	$P(b_i \mid \mathbf{e})$	9
	$P(d_i \mid \mathbf{e})$	5
	$P(c_i \mid \mathbf{e})$	9
Total		123

Table 1: Number of mathematical operations required.

the topology of the network, consider the addition of a link from D_0 to D_N. The number of nodes remains the same and the number of edges varies only from $4N$ to $4N+1$, but the complexity of LC is almost double, because every node B_i and D_i will bear an extra conditioning on D_0. A triangulation of the augmented graph may lead to the presence of D_0 in every clique, which is equivalent to conditioning the entire network on D_0 [24]. It does not seem plausible that a different triangulation or a different exact algorithm might produce a significantly better result. Apparently, the addition of the new arc raises the intrinsic complexity of the network, independently of the algorithm employed.

In contrast, the addition of a new link barely increases the complexity of an approximate method and such an increase is the same for link D_0D_1 as for link D_0D_N.

References

[1] M. Baker and T. E. Boult. Pruning Bayesian networks for efficient computation. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 225–232. Elsevier Science Publishers, Amsterdam, 1991.

[2] R. Bar-Yehuda, D. Geiger, J. Naor, and R. Roth. Approximation algorithms for the vertex feedback set problem with applications to constraint satisfaction and Bayesian inference. In *Proceedings of the 5th Annual ACM-Siam Symposium On Discrete Algorithms*, pages 344–354, Airlington, VA, 1994.

[3] A. Becker and D. Geiger. Approximation algorithms for the loop cutset problem. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 60–68, Seattle, WA, 1994. Morgan Kaufmann, San Francisco, CA.

[4] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[5] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[6] A. Darwiche. Conditioning algorithms for exact and approximate inference in causal networks. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 99–107, Montreal, 1995. Morgan Kaufmann, San Francisco, CA.

[7] F. J. Díez. Parameter adjustement in Bayes networks. The generalized noisy OR–gate. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 99–105, Washington D.C., 1993. Morgan Kaufmann, San Mateo, CA.

[8] F. J. Díez. *Sistema Experto Bayesiano para Ecocardiografía*. PhD thesis, Dpto. Informática y Automática, UNED, Madrid, 1994. In Spanish.

[9] F. J. Díez and J. Mira. Distributed inference in Bayesian networks. *Cybernetics and Systems*, 25:39–61, 1994.

[10] R. Fung and B. del Favero. Applying Bayesian networks to information retrieval. *Communications of the ACM*, 38:42–48,57, 1995.

[11] R. P. Goldman and E. Charniak. Probabilistic text understanding. *Statistics and Computing*, 2:105–114, 1992.

[12] D. Heckerman. Causal independence for knowledge acquisition and inference. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 122–127, Washington D.C., 1993. Morgan Kaufmann, San Mateo, CA.

[13] M. Hede and R. Stigaard. Distributed Hugin — An approach to distributed inference in large Bayesian networks. Master's thesis, Dept. Mathematics and Computer Science, Aalborg University, 1994.

[14] F. V. Jensen, K. G. Olesen, and S. K. Andersen. An algebra of Bayesian belief universes for knowledge-based systems. *Networks*, 20:637–660, 1990.

[15] J. H. Kim and J. Pearl. A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 190–193, Karlsruhe, Germany, 1983.

[16] U. Kjærulff. Graph triangulation — Algorithms giving small total state space. Technical Report R–90–09, University of Aalborg, Denmark, 1990.

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

[18] A. Kong. Efficient methods for computing linkage likelihoods of recessive diseases in inbred pedigrees. *Genetic Epidemiology*, 8:81–103, 1991.

[19] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.

[20] R. E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. Wiley-Interscience, New York, 1990.

[21] J. Pearl. Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[22] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988. Revised second printing, 1991.

[23] M. A. Peot and R. D. Shachter. Fusion and propagation with multiple observations in belief networks. *Artificial Intelligence*, 48:299–318, 1991.

[24] R. D. Shachter, S. K. Andersen, and P. Szolovits. Global conditioning for probabilistic inference in belief networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 514–522, Seattle, WA, 1994. Morgan Kaufmann, San Francisco, CA.

[25] J. Stillman. On heuristics for finding loop cutsets in multiply connected belief networks. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 233–243. Elsevier Science Publishers, Amsterdam, 1991.

[26] H. J. Suermondt and G. F. Cooper. Probabilistic inference in multiply connected belief networks using loop cutsets. *International Journal of Approximate Reasoning*, 4:283–306, 1990.

[27] H. J. Suermondt and G. F. Cooper. A combination of exact algorithms for inference on Bayesian belief networks. *International Journal of Approximate Reasoning*, 5:521–542, 1991.

[28] H. J. Suermondt and G. F. Cooper. Initialization for the method of conditioning in Bayesian belief networks. *Artificial Intelligence*, 50:83–94, 1991.

[29] W. X. Wen. Optimal decomposition of belief networks. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 209–224. Elsevier Science Publishers, Amsterdam, 1991.