Distributed inference in Bayesian networks[*]

F. J. Díez J. Mira

Dpto. Informática y Automática. UNED

Senda del Rey. 28040 Madrid. Spain

<FJavier.Diez@uned.es>

Abstract

Bayesian networks originated as a framework for distributed reasoning. In singly-connected networks, there exists an elegant inference algorithm that can be implemented in parallel having a processor for every node. It can be extended to take profit of the OR–gate, a model of interaction among causes which simplifies knowledge acquisition and evidence propagation. We also discuss two exact and one approximate methods for dealing with general networks. All these algorithms admit distributed implementations.

1 Introduction

In the 1970s, expert systems were created as an intent to separate represented knowledge from reasoning strategies. However, in the next decade it was shown that rule-based systems were not as modular as initially claimed [7]. It was also shown that there were inconsistencies in the use of MYCIN's certainty factors, the most widely applied model for reasoning in expert systems [10]. The fundamental problem is not specific of MYCIN's model, but common to all "modular" systems, in which there are strong implicit assumptions about conditional independence that are not valid in general [17, pp. 4–12], [14, pp. 125ff].

Bayesian networks (BNs), which were initially conceived as a framework for distributed reasoning [18], offer a solution by resting on a graphical representation in which all dependencies (and, by default, all independencies) are clearly indicated by the arcs connecting the nodes. This property is so important that the name of *independence networks* has been suggested as the most adequate denomination for this scheme [14]. There is a solid graph theory [19, 20] that constitutes the axiomatic framework of BNs.

The purpose of this paper is to present BNs from the viewpoint of distributed computation. Our objective is not to make an exhaustive review of the field, but rather to show how knowledge representation and reasoning (evidence propagation) can be performed in a distributed fashion having a processor for every node.

In order to structure the presentation, we will performe our study at the three levels defined by David Marr [13, section 2.1]: **theory**, **algorithm** and **implementation**. In the case of BNs, the theory of probabilistic inference is given by the definitions and the axioms of independence introduced in section 2. Some algorithms derived from the theory

[*] *Cybernetics and Systems*, **25** (1994) 36-61.

are presented in sections 3 to 5. All those algorithms admit parallel implementations, which are the object of this paper. Here, we restrict ourselves to evidence propagation. In [5], we follow the same tree-level approach to study distributed learning and explanation in Bayesian expert systems.

2 Theory of Bayesian networks

This section introduces quite informally the fundamentals of BNs. For a proper presentation, see [14, 17].

In a first approximation, a *Bayes network* can be defined as a directed acyclic graph where nodes represent variables and links represent causal relations. "Acyclic" means that, if there is a loop, its arrows do not form a closed path. The meaning of "causal relations" will be given by the independence axioms presented below. Except in clique trees, the terms "node" and "variable" can be used as synonyms. We represent a node or variable by a capital letter, X, and its values in lower case, x. Singly-connected networks are called *polytrees*.

A node X is a *parent* of Y is there is a link from X to Y; conversely, Y is a *child* of X. The set of parents of X is represented as $pa(X)$. The *family* of X is formed by X and its parents. Node U is an *ancestor* of Y if U is a parent of Y or there is a node X such that U is an ancestor of X and X is a parent of Y; conversely, Y is a *descendant* of U. In the context of causal networks, parent is synonym of cause, and child of effect.

A subset of nodes, $W = \{W_1, \ldots, W_N\}$, *separates* variables X and Y if the instantiation of the variables in the subset makes X and Y conditionally independent:

$$\forall x, \forall y, \forall w,\ P(y|w) \neq 0 \Longrightarrow P(x|y, w) = P(x|w).$$

Observe that this definition is symmetric for X and Y. It can be immediately extended from single variables to subsets of them.

Definition 1 (Bayesian network) *Given a directed acyclic graph and a joint probability distribution for the variables it represents, they are said to constitute a* Bayes network *if the set of parents of X, $pa(X)$, separates it from every node that is not one of its descendants.*

This defining property is called *d-separation* (directional separation) because of its asymmetry, which becomes apparent in the following properties derived from it and illustrated in fig. 1:

1. If A is a top node, $pa(A) = \emptyset$. Since trivially $P(x|\emptyset) \equiv P(x)$, the separation property turns into $P(e|a) = P(e)$ for every node E which is not one of the descendants of A; in other words, E is *a priori* independent of A. As a consequence, any two nodes D and E having no common ancestor are *a priori* independent.

2. If A is an ancestor of D, H is a descendant of D and there is no other path from A to H, both of them are separated by D:

$$P(h|d, a) = P(h|d). \tag{1}$$

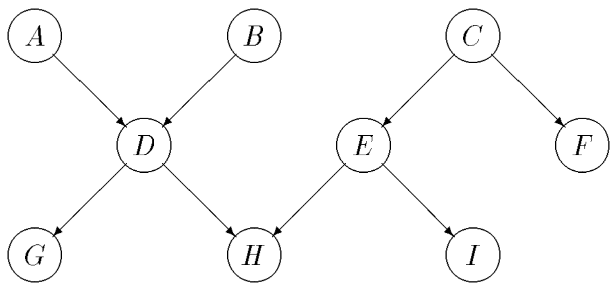

Figure 1: A small polytree.

3. If G and H are both children of D and they have no other common ancestor, the latter separates G and H, so that they are conditionally independent:

$$P(g|d,h) = P(g|d). \tag{2}$$

4. Two nodes which are independent (*a priori* or conditionally) remain to be so when no *common* descendant is instantiated. For example, A is independent of F even after the instantiation of D:

$$P(a|f,d) = P(a|d)\,, \tag{3}$$

and in the same way

$$P(e|f,c,h) = P(e|c,h)\,. \tag{4}$$

In general, the (*a priori* or conditional) independence of two nodes, A and E, is broken when some of its common descendants, H, is instantiated.

It is possible to show that the distribution mentioned in the definition of Bayesian networks can be given by assigning a conditional probability $P(x|pa(x))$ for every node with parents, and an *a priori* probability for every node without parents [14]. The above independence properties could have been defined alternatively in terms of blocked and activated paths [15].

What is the semantics of Bayesian networks? The answer lies in the correspondence between these statistical independence properties and our common-sense knowledge about the interaction of causes and effects. For instance, eqs. (1) and (2) can be interpreted as saying that the probability that a cause D produces an effect G depends only on the value taken by the cause (for instance, absent, mild, moderate or severe), not on how it was produced nor on whether it has produced other effects [6]. This is the justification of why Bayesian networks, mathematically defined, are used to represent causal models. For this reason, parent is a synonym of immediate cause and children is a synonym of immediate effect.

3 Evidence propagation in polytrees

3.1 Algorithm

Our objective is to find the *evidential probability* $P^*(x)$, defined as the probability of proposition "The value of variable X is x" given the observed evidence $\mathbf{e}$:

$$P^*(x) \equiv P(X = x|\mathbf{e}) \tag{5}$$

In the case of a singly-connected network, an arbitrary node X divides this evidence into that coming from its causes, $\mathbf{e}^+_X$, and that coming from its effects, $\mathbf{e}^-_X$. Accordingly, we can define

$$\pi(x) \equiv P(x, \mathbf{e}^+_X) \tag{6}$$

and

$$\lambda(x) \equiv P(\mathbf{e}^-_X \,|\, x) \tag{7}$$

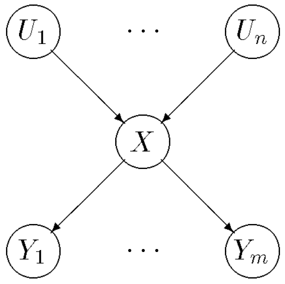

Figure 2: Parents and children of X.

If the causes of X are $U_1, \ldots, U_n$, and its effects $Y_1, \ldots, Y_m$ (see fig. 2), we introduce

$$\pi_X(u_i) \equiv P(u_i, \mathbf{e}^+_{U_i X}) \tag{8}$$

and

$$\lambda_{Y_j}(x) \equiv P(\mathbf{e}^-_{XY_j} \,|\, x) \tag{9}$$

where $\mathbf{e}^+_{XY}/\mathbf{e}^-_{XY}$ represents the evidence above/below link $X \rightarrow Y$. The axioms of independence for BNs allow us to easily compute these expressions. First, we have

$$\begin{aligned} P^*(x) &= \alpha\, P(x, \mathbf{e}^+_X, \mathbf{e}^-_X) \\ &= \alpha\, P(x, \mathbf{e}^+_X)\, P(\mathbf{e}^-_X \,|\, x) \\ &= \alpha\, \pi(x)\, \lambda(x) \end{aligned} \tag{10}$$

where $\alpha \equiv [P(\mathbf{e})]^{-1}$ is a normalization constant.

The causes of X have no common ancestor (we are in a singly-connected network), so they are independent when there is no evidence below X:

$$P(u_1, \ldots, u_n, \mathbf{e}^+_X) = P(u_1, \ldots, u_n \,|\, \mathbf{e}^+_{U_1X}, \ldots, \mathbf{e}^+_{U_nX}) = \prod_{i=1}^{n} \pi_X(u_i)\,.$$

Therefore,

$$\pi(x) = \sum_{u_1, \ldots, u_n} P(x \,|\, u_1, \ldots, u_n) \prod_{i=1}^{n} \pi_X(u_i)\,. \tag{11}$$

Conditional independence on X leads to

$$\begin{aligned} \lambda(x) &= P(\mathbf{e}^-_{XY_1}, \ldots, \mathbf{e}^-_{XY_m} \,|\, x) \\ &= \prod_{j=1}^{m} P(\mathbf{e}^-_{XY_j} \,|\, x) = \prod_{j=1}^{m} \lambda_{Y_j}(x) \end{aligned} \tag{12}$$

Similarly, node X separates Y_j from other effects and from causes of X. Therefore, after definition (8),

$$\begin{aligned}\pi_{Y_j}(x) &= P(x, \mathbf{e}^+_{XY_j}) = P(x, \mathbf{e}^+_X, \mathbf{e}^-_{XY_k}\ k \neq j) \\ &= \pi(x) \prod_{k \neq j} \lambda_{Y_k}(x) \qquad (13)\end{aligned}$$

In order to calculate $\lambda_{Y_j}(x)$, defined in eq. (9), let us denote by V the set of causes of Y_j different from X, as shown in fig. 3. We then have

$$\begin{aligned}\lambda_{Y_j}(x) &= \sum_{y_j, v} P(\mathbf{e}^-_{Y_j}, \mathbf{e}^+_{VY_j}, y_j, v \,|\, x) \\ &= \sum_{y_j, v} P(\mathbf{e}^-_{Y_j} \,|\, y_j)\, P(y_j \,|\, v, x)\, P(\mathbf{e}^+_{VY_j} \,|\, v)\, P(v) \\ &= \sum_{y_j} \left[P(\mathbf{e}^-_{Y_j} \,|\, y_j) \sum_{v} P(y_j \,|\, v, x)\, P(v, \mathbf{e}^+_{VY_j}) \right]\end{aligned}$$

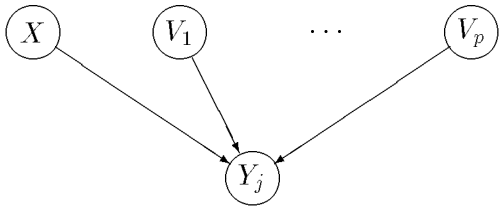

Figure 3: Parents of Y_j.

The causes of Y_j are *a priori* independent. Hence,

$$P(v, \mathbf{e}^+_{VY_j}) = P(v_1, \ldots, v_p \,|\, \mathbf{e}^+_{V_1Y_j}, \ldots, \mathbf{e}^+_{V_pY_j}) = \prod_{k=1}^{p} \pi_{Y_j}(v_k)$$

and, in consequence,

$$\lambda_{Y_j}(x) = \sum_{y_j} \left[\lambda(y_j) \sum_{v_1, \ldots, v_p} P(y_j \,|\, x, v_1, \ldots, v_p) \prod_{k=1}^{p} \pi_{Y_j}(v_k) \right] \qquad (14)$$

The ten numbered equations in this section display five definitions and the corresponding formulas to compute them. According to these formulas, the only data needed by a general node X are the probability of every value X for every instantiation of its causes, $P(x \,|\, u_1, \ldots, u_n)$. Instead, for a top node having no causes explicit in the model, the *a priori* probability $P(x)$ must be provided. A leaf node Z must be assigned a vector $\lambda(z)$: if there is no observed evidence for this node, its initial value is a constant vector; otherwise, the observed value is assigned a 1 and the other values of the variable are assigned 0's. This assignment of π's and λ's for top and leaf nodes respectively guarantees a termination condition for the recursive formulas given above.

The most important property derived from the axioms of independence is that, in polytrees, every link decomposes the network into two parts, whose only interaction comes through that link, and the messages between them are orthogonal, i.e. $\pi_Y(x)$ can be computed independently of $\lambda_Y(x)$ and vice versa. In fig. 4, which shows the computations performed at node X, this property becomes apparent as the absence of loops in the information flow.

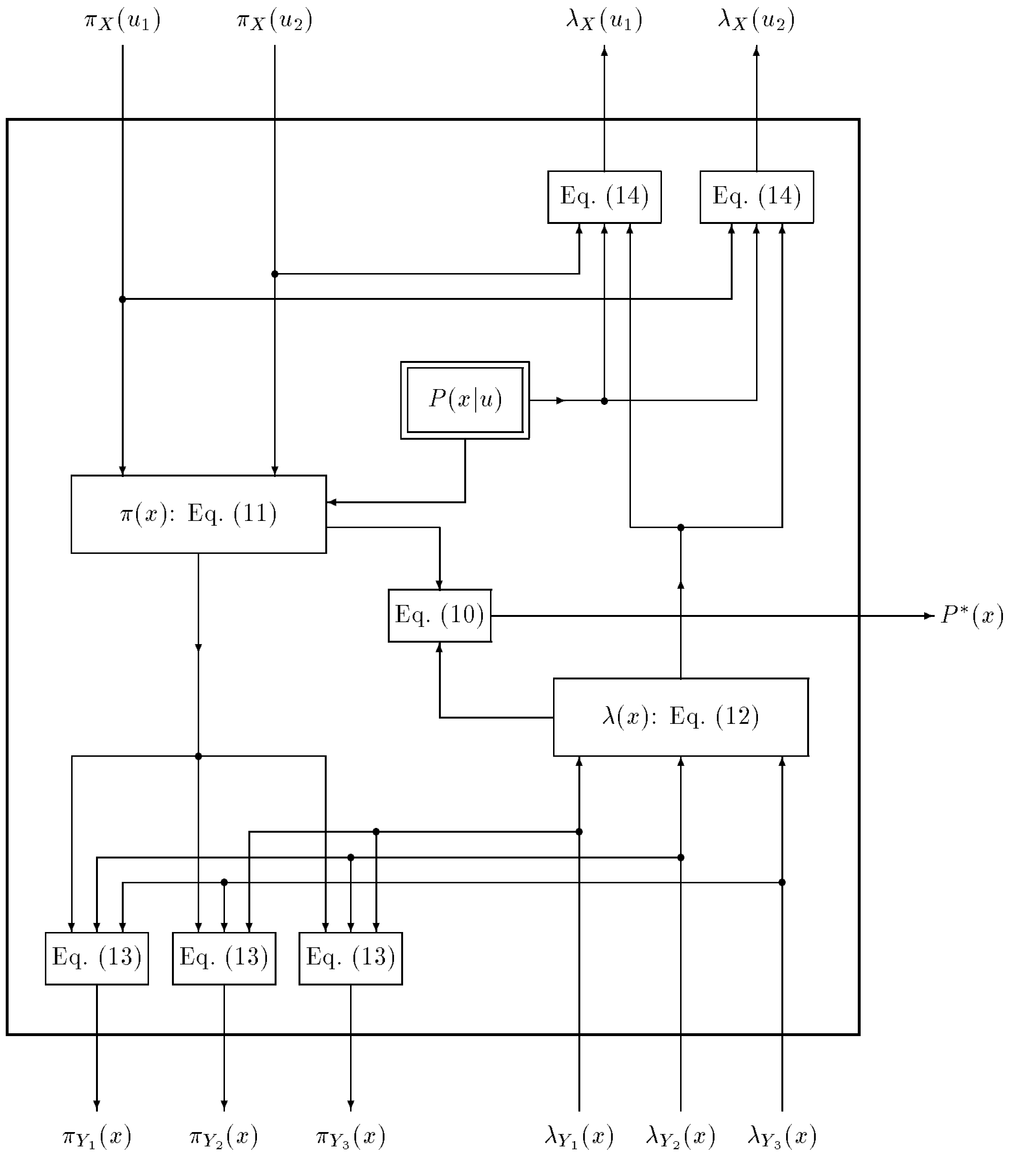

Figure 4: Computations performed at node X.

3.2 Distributed implementation

The algorithm presented above requires that every node stores some information, both static and dynamic. The static information, such as the structure of the network and the conditional probabilities, is independent of the observed evidence. The dynamic information is obtained in the process of evidence propagation.

Firstly, a node must know its causes and its effects, in order to implement the network structure. This could be provided by two lists containing pointers to the corresponding nodes in the network. Nevertheless, there is some information associated to links rather than to nodes (for example, the $\pi_Y(x)$ and $\lambda_Y(x)$ messages), so it is more suitable to implement every link as an object capable of storing information, in order to avoid unnecessary recomputation. Nodes should contain pointers to links and vice versa. These pointers implement the topology of the network. It is then necessary to include the static numerical information, the conditional and *a priori* probabilities, as explained in the preceding section.

We will study now how dynamic information should be computed. From the formulas of evidence propagation, we can see that node X can send a message to its neighbor W only when X has received the messages from all of its other neighbors. A node X with n causes and m effects that has received q messages must be in one of three possible states:

1. $q \leq n + m - 2$. It means that X is still waiting for at least two messages. So it cannot yet send any message.

2. $q = n + m - 1$. Then, X has received a message from everyone of its neighbors, except from one—say W. In this case, X can already compute and send the message to W, but no other message yet.

3. $q = n + m$. In this case, X is able to compute and send all the messages it hasn't sent before.

At the beginning, $q = 0$ for every node, since no message has been transmitted yet; therefore, all *end* nodes are in state 2 because they have just one neighbor. The rest are in state 1. It is possible to show that there is always some node able to send a message, until the process is complete. The proof is easy but a little awkward, so we'd better consider as an example the polytree in fig. 1. Before evidence propagation starts, all end nodes (A, B, F, G and I) are in state 2. The rest are in state 1. When those end nodes send their respective messages, C and D change into state 2, and the same happens to E and H in the next step. After messages $\pi_H(e)$ and $\lambda_H(e)$ are transmitted, these latter nodes are in state 3. Then they send messages to their neighbors, and in two more steps the process is concluded.

The discussion above is relevant to show that it is not necessary to have a mechanism for global control. The model can be implemented as an *asynchronous network* in which the number of messages received by a node determines which messages it can compute and send.

If the computation at every node is bounded (by limiting the number of parents and values) and the algorithm is implemented sequentially, the time complexity is proportional to the number of nodes. In this case, it is more efficient to perform evidence propagation in two phases: evidence collection toward a pivot node and evidence distribution from this node, as was suggested in [12] for trees of cliques.

On the other hand, if there is a processor for every node, the time complexity of the algorithm is proportional to the maximum length in the network. The version presented here, based on three different states for a node, is slightly different from Pearl's [17] in that it avoids computing and sending premature meaningless messages. This distinction is not important if we can afford a hardware processor for every node. But if the conceptual processors (the nodes) are implemented by a smaller set of hardware processors, the computational squandering of computing useless messages may turn out to be very expensive. In this latter case, when nodes are queuing for access to a limited number of hardware processors, we encounter the typical problem of parallel programming, namely, what message has to be computed first in order to improve performance.

4 The OR–gate

The general approach in BNs involves having a table of conditional probabilities $P(x|\, u_1, \ldots, u_n)$ for each node X, where U_i are the parents (the causes) of X. Unfortunately, there are several shortcomings. First, the number of parameters required for a node grows exponentially with the number of its causes. Even if there is a database available, a lot of cases are required in order to obtain accurate parameters. If the lack of a database forces us to resort to human experts, it will be difficult for them to estimate so many conditional probabilities, depending on complex combinations of the different values for the direct causes of a node. And even when these parameters are obtained, there is still the burden of storing such big tables. A second drawback is related to computational complexity. The number of mathematical operations required by each node also grows exponentially with the number of its parents.

Theory. In the OR–gate [17, pp. 184–188], a parent node of X is not conceived as a mere factor (age of the patient, for instance) modulating the probability of X given a certain configuration of the other parents (sex, weight, smoking, etc.). Instead, node X represents a physical-world entity (for example, a disease) that may be present or absent, and its parents represent phenomena —in general anomalies— whose presence can produce X. In other words, a link in the OR–gate represents *the intuitive notion of causation*, not only the statistical definition given in [21].

In the generalized noisy OR–gate [9, 4], a variable X is assumed to have g_X degrees of presence. For example, if $X = \{\mathsf{absent}, \mathsf{mild}, \mathsf{moderate}, \mathsf{severe}\}$, then $g_X = 3$. "$X = 0$" means "X is absent" and successive integers indicate higher degrees of presence.

The basic parameter is the *effectiveness* $c_x^{u_i}$ defined as the probability that a cause U_i raises an effect X to a degree x when all of its other causes are absent:

$$c_x^{u_i} \equiv P(X = x|\, U_i = u_i, U_j = 0 \text{ for } j \neq i). \tag{15}$$

Obviously,

$$\sum_x c_x^{u_i} = 1,\ \forall u_i. \tag{16}$$

The first assumption of the OR–gate is that X is absent when all of its causes are absent:[1]

$$P(X = 0|\, U_i = 0, \forall i) = 1. \tag{17}$$

[1]In the leaky OR–gate [9], this assumption is relaxed (see below).

The second assumption is as follows:

$$P(X \le x|u_1, \ldots, u_n) = \prod_i P(X \le x|U_i = u_i, U_j = 0 \text{ for } j \ne i). \tag{18}$$

The intuitive interpretation is that $X \le x$ when every cause has raised X to a degree not higher than x. In other words, the value taken on by X is the maximum of the values produced by its causes acting independently.

Algorithm. From these postulates, it is possible to deduce the general formula for $\pi(x)$ as a function of the $\pi_X(U_i)$'s and of these parameters (see [4] for a proof):

$$\pi(x) = \begin{cases} Q(x) - Q(x-1) & \text{for } x \ne 0 \\ Q(X = 0) & \text{for } x = 0 \end{cases} \tag{19}$$

where

$$Q(x) = \prod_i Q_{U_i}(x) \tag{20}$$

and

$$Q_{U_i}(x) = 1 - \sum_{u_i=1}^{g_{U_i}} \left[\pi_X(u_i) \sum_{x'=x+1}^{g_X} c_{x'}^{u_i} \right]. \tag{21}$$

We also have

$$\lambda_X(u_i) = \sum_x \lambda(x)\,[\pi(x)]_{U_i=u_i}. \tag{22}$$

Here, $[\pi(x)]_{U_i=u_i}$ must be computed after eqs. (19) to (21) but with variable U_i clamped to value u_i; thus,

$$[Q_{U_i}(x)]_{U_i=u_i} = 1 - \sum_{x'=x+1}^{g_X} c_{x'}^{u_i}. \tag{23}$$

In some domains, it is often impossible to detail all the causes of a certain anomaly. To deal with this difficulty, we can include a dummy parent node standing for all the causes of X not explicit in the model. In this case, the interaction is called *leaky* OR–gate [9]. In the implementation, instead of adding a dummy node, the leaky OR–gate can be implemented by assigning parameters c_x^L to node X, as shown in table 1. The computation of Q_L is as follows:

$$Q_L(x) = 1 - \sum_{x'=x+1}^{g_X} c_{x'}^L, \tag{24}$$

and this factor must be included in eq. (20) too.

As a consequence of the constraints given by eqs. (16) and (17), only $g_U \cdot g_X$ parameters are required for link UX. In the case of binary variables, only one parameter is necessary for each link. For example, in an expert system for cardiology, where we have 8 explicit causes of mytral stenosis, the general case would need a table containing $2^8 = 256$ parameters; the leaky OR–gate needs only 9 parameters, namely, one for every cause plus one more for representing the causes not explicit in the model. Thus, the number of parameters and the computational complexity are reduced. A few questions such as "What is the probability that amyloidosis produces mytral stenosis (when no other cause is present)?" are more easily answered by a physician or from a textbook than a great amount of questions involving a complex casuistry.

At the level of evidence propagation, the main interest of the OR–gate is that the computation of $\pi(x)$ takes a time proportional to the number of causes of X, instead of the exponential time required by the general case.

Another advantage is that in a noisy OR–gate it is easier to generate a linguistic explanation of "What is the most probable cause of X?" than in the case of probability tables.

In a similar way, it is possible to define the noisy AND–gate and its generalization to multivalued variables, which shares the same properties as the generalized noisy OR–gate. Obviously, the three types of interaction (probability tables, OR, AND) can be present in a certain BN for different families.

Distributed implementation. From the point of view of a parallel implementation (or even of object-oriented programming), the parameters of the AND/OR–gate —unlike the case of probability tables— are not a feature of node X but rather they are associated to each particular link UX. Observe in fig. 5 and in table 1 where the c_x^u's are stored. Node X must just know what type of gate to apply and the actual parameters can be stored in the corresponding links. Comparing fig. 5 to the corresponding drawing for the general case (fig. 4), we observe that now links are not passive channels of information, but active processors which convert message $\pi_X(u)$ into $Q_U(x)$ and generate $\lambda_X(u)$, thus releasing node X from some computations.

		General model	OR–gate
Node X	Stores	$P(x\|u)$	c_x^L
	Receives	$\pi_X(u_i)$, $\lambda_{Y_j}(x)$	$Q_{U_i}(X)$, $\lambda_{Y_j}(x)$
	Sends	$\lambda_X(u_i)$, $\pi_{Y_j}(x)$, $P^*(x)$	$\lambda(x)$, $\pi_{Y_j}(x)$, $P^*(x)$
Link U_iX	Stores		$c_x^{u_i}$
	Receives		$\pi_X(u_i)$, $\lambda(x)$, $Q_{U_{i'}}(x)$
	Sends		$Q_{U_i}(x)$, $\lambda_X(u_i)$

Table 1: General case and the OR–gate

5 Dealing with loops

The expressions derived in the last two sections provide an elegant and efficient method for computing probability in polytrees. However, it can be deduced from the **theory** of BNs that multiply connected nets give rise to an NP–hard problem, because of the presence of loops [1, 2]. Among the different **algorithms** proposed, we will present here two exact methods, clustering and conditioning, and an approximate method, simulation.

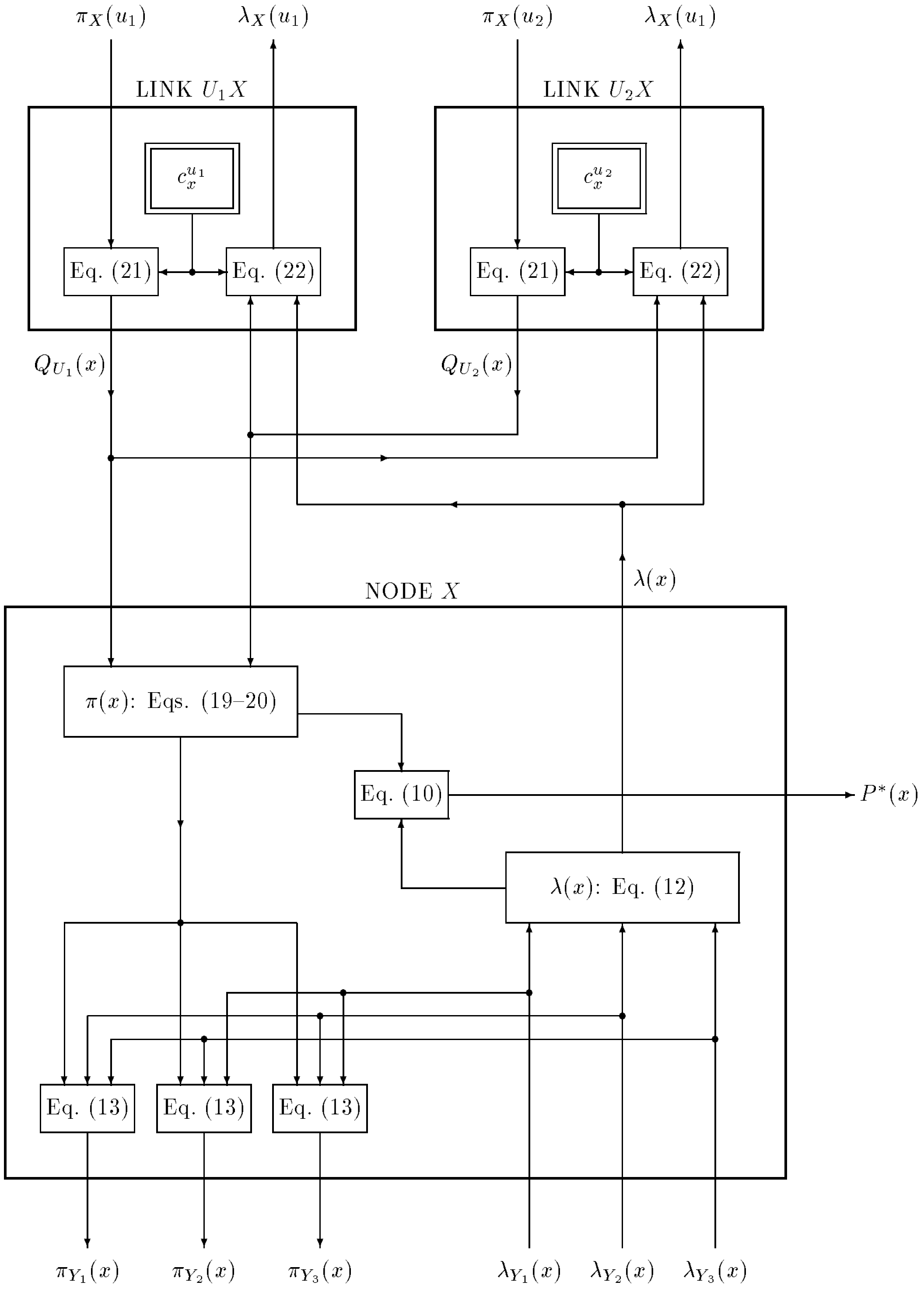

Figure 5: Computations performed at the OR–gate.

5.1 Clustering

Nowadays, the most widely used **algorithm** for computing probability in BNs is the clique-tree propagation method, developed by Lauritzen and Spiegelhalter [12] and improved by Jensen et al. [11]. A generalization of the method has recently been proposed by Shachter et al. [23].

The method is based on a compilation of the network in order to build a tree of cliques. The first step is to "moralize" the graph by marrying (adding links between) the parents of each node. If the network is still non-triangulated, some additional links are required; Lauritzen and Spiegelhalter suggest using maximum-cardinality search [25] for this process. Cliques are defined as "maximal fully connected sets of nodes" in the triangulated graph. In general, a variable is included in more than one clique. The tree of cliques, built in the static phase of the method, is independent of the observed evidence.

Then the tree is initialized by assigning to every clique C_i a function of its variables, called *potential function* Ψ_i, obtained from the conditional probability tables and the available evidence, which constitutes an initial distribution of marginal probabilities. These quantities are used to update the probabilities when new evidence arrives. The probability corresponding to a variable can be deduced by marginalizing and normalizing the probability table of one of the cliques which contain that variable. When findings are entered, the process of calibration is in fact the propagation of evidence. It is very similar to evidence propagation in polytrees.

The method proposed by Shachter et al. [23] defines cluster trees as a generalization of clique trees. In their algorithm, the message transmitted from cluster C_i to cluster C_j (fig. 6) is computed as

$$M_{ij} = \sum_{C_i \backslash C_j} \Psi_i \prod_{k \in K_{i-j}} M_{ki} \tag{25}$$

where $C_i \backslash C_j$ means the variables of C_i not included in C_j, and K_{i-j} indicates the neighbor clusters of C_i other than C_j. Indicating by K_i all the clusters adjacent to C_i, the probability of a configuration of the latter is given by

$$P_i = \alpha \, \Psi_i \prod_{k \in K_i} M_{ki}. \tag{26}$$

As usual, α is a normalization constant. We can see that Ψ_i and P_i depend on the variables contained in C_i, and message M_{ki} depends on the variables belonging to both C_i and C_j.

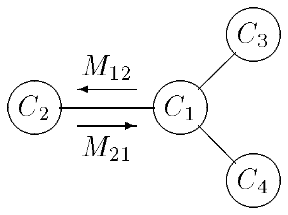

Figure 6: Cluster tree.

Again, a **distributed implementation** is possible. Fig. 7 shows the computations performed at a clique C_1 connected to C_2, C_3 and C_4. Every node must store the following information:

- a *list of the variables* clustered into the node;
- *conditional probability tables* (the table corresponding to each family is assigned to one of the cliques in which it is contained); some cliques may have a table equal to unity for all its values;
- a *potential function* Ψ_i, obtained from the conditional probability tables and the available evidence associated to this clique $\mathbf{e}_i$. This value is assigned when initializing the clique tree and does not change during evidence propagation.

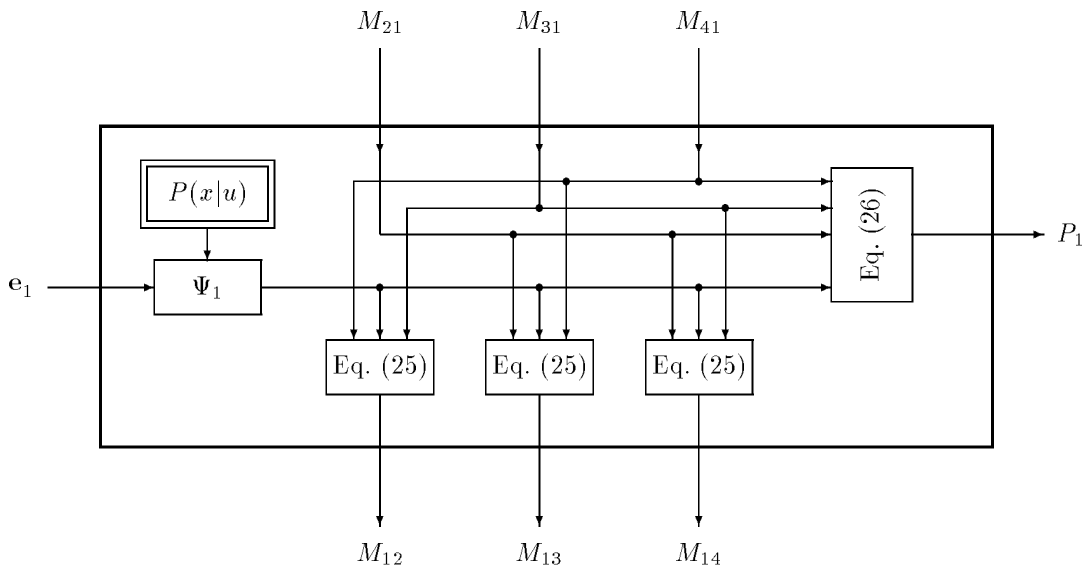

Figure 7: Computations performed at cluster C_1.

Besides having $\mathbf{e}_i$ as an input, node C_i receives messages M_{ji} from all its neighbors C_j and uses them to compute P_i and the M_{ij} messages. Observe again in fig. 7 that there is no loop in the information flow. In this figure, eq. (25) should have an input indicating $C_1 \backslash C_j$ but this is more a property of the link than of the node and is omitted for the sake of clarity.

Everything we said about propagation in polytrees can be also applied to cluster trees, with the following differences:

1. A polytree is a directed graph —a link goes from a cause to one of its effects— while cluster trees are undirected—there is no distinction between causes and effects. In fact, fig. 7 is essentially the same as fig. 4. Only the intermediate computation of $\pi(x)$ and $\lambda(x)$ makes them look different.
2. In a polytree every node represents a variable; in a cluster tree, every node lumps together several variables. Therefore, in the latter case, the probability of a variable must be obtained by marginalization on any of the cliques containing that variable.
3. Messages transmitted in a polytree depend on one variable; in a cluster tree, messages between two nodes depend on the variables they share. (The intersection of two adjacent clusters is called their separator.)

5.2 Conditioning

Among exact BNs **algorithms**, conditioning is based on the fact that variables in a loop have independent probabilities when a variable not at the bottom of the loop is instantiated (conditional independence). The original method proposed by Pearl [17, pp. 204–210] was quite inefficient. In addition to requiring initialization [24], the time complexity of evidence propagation was proportional to the number of instantiated leaf nodes and, what is much worst, exponential in the number of nodes required to break the loops.

Peot and Shachter [22] described a method whose complexity was independent of the number of instantiated leaf nodes. Furthermore, it allowed the decomposition of the network into blocks, thus reducing the complexity of the computation for many structures.

Recently, a more efficient algorithm to deal with loops, called "local conditioning" has been proposed in [3]. The main innovation of this technique consists of detecting the loops each node belongs to—the paper suggest using a depth-first search for this purpose. As a result, every node is assigned a list of variables on which it must be conditioned. At the same time, conditioning allows the removal of some edges from the original graph so that it turns into a singly-connected network (in which every node still represents a variable).

In general, messages propagated between nodes do not depend now on one variable (as in the case of the polytree), but on several of them. When entering a loop, the dimension of messages is increased by one, and when exiting, it is reduced again. The list of variables associated to each loop allows the nodes to make local decisions on when to apply conditioning, so that it is not propagated any longer than necessary. As a consequence, this algorithm has linear complexity for some structures for which previous methods were exponential.

According to [23], a conditioning solution for a BN is equivalent to a clustering solution with a certain triangulation. However, local conditioning is the only known exact algorithm for BNs that allows a **parallel implementation** with a processor for every node. In fact, local conditioning propagates evidence in an associated tree obtained by removing some edges in the original graph. Figs. 4 and 5 and table 1 are still valid. The only differences are that every node must also store a list of loops and that, in general, messages now depend on more than one variable.

In [5], we show how to integrate local conditioning with learning, in a distributed fashion. We expect that a distributed capability of explanation could also emerge from local conditioning.

5.3 Simulation

The fact that exact inference in general BNs is NP-hard [1], was often used as an argument in favor of approximate methods, implicitly assuming that the time complexity of the proposed stochastic algorithms was polynomial. However, it has been recently shown that "approximating probabilistic inference in Bayesian belief networks is NP-hard" too [2]. Anyway, stochastic simulation is still a useful method for practical applications.

In this case, probabilities are estimated by computing the frequency of a given scenario in a series of simulation runs. The Bayes network constitutes the probability model that generates the trials. The advantage of simulation over exact methods is that its complexity depends just on the number of nodes and edges, and is not affected by the topology of the network. On the other hand, simulation is very sensitive to numerical values, which could make the approximation converge too slowly.

The first simulation scheme was logic sampling, as presented in [8]. The network is used as a trial generator. Only those trials that agree with observed evidence are taken into account. The shortcoming of this simple method is that it generates many irrelevant trials, especially when several leaf nodes are instantiated, leading to a low efficiency of the algorithm.

In the simulation **algorithm** proposed by Pearl [16], evidence variables are clamped to their observed values, in order to generate only useful trials. Although it has been improved by other researchers (see references in [2]), we will here present this straight simulation method because of its simplicity and because it lends itself naturally to parallel implementation.

The basic idea of this algorithm is that every node is assigned a value by a random process according to the probability distribution induced by the values of its neighbors. In a sequential implementation, only one node is activated at a given moment. Nevertheless, in a **parallel implementation**, the simulation might be wrong if two adjacent nodes were activated simultaneously. In order to avoid this difficulty, arc reversal was introduced as a distributed control strategy which guarantees that every processor is activated sufficiently often without having two adjacent processors activated at the same time.

The first step is to initialize the network by assigning an orientation to every link, with the only condition that the resulting network is acyclic. The causal ordering of the Bayesian network fulfills this property, but any other acyclic ordering would also be valid. Then, simulation starts. Each processor remains idle until all its links are pointing inward. It is then activated and performs the following steps:

1. Compute the conditional distribution of the variable given the states of its neighbors (the values assumed by the variables).
2. Get a value for the variable by sampling the computed distribution.
3. Reverse the direction of all its arrows, so that they point outwards. Then, this node becomes idle again and other nodes are activated.

It can be shown that two neighbor nodes are never activated simultaneously, that every processor is fired sufficiently often and that the process converges (see references in [16]).

Observe that all computations as well as the control strategy are local and can be implemented as distributed asynchronous processes.

6 Conclusion

We have presented several inference methods for Bayesian networks, following the scheme "theory-algorithm-implementation". In the case of polytrees, the algorithm based on π/λ-messages lends itself naturally to parallel implementation, and the same is true for stochastic simulation. When the network contains loops, the local conditioning algorithm cuts them by applying conditioning, while clustering methods deal with them by grouping the variables; in both cases, the objective is to build a tree so that propagation is similar to that for singly-connected BNs. All these methods can be implemented in parallel having a processor for every node (for every cluster, in clustering algorithms), with local computations and local control.

Discussions in this paper are based on some of the algorithms currently available. Our purpose was not, however, to study particular methods but rather to show the possibilities that BNs offer for distributed computation. In [5], we study how to develop local learning and local explanation capabilities, in order to turn BNs into Bayesian expert systems.

References

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

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

[3] F. J. Díez. Local conditioning in Bayesian networks. Technical Report (R–181), Cognitive Systems Laboratory, University of California, Los Angeles, 1992. Submitted to *Artificial Intelligence.*

[4] 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.

[5] F. J. Díez and J. Mira. Distributed reasoning and learning in Bayesian expert systems. In C. A. Ntuen, editor, *Advances in Fault-Diagnosis Problem Solving.* CRC Press, Boca Raton, FL, 1993. To appear.

[6] F. J. Díez Vegas and J. Mira Mira. Causal Bayesian reasoning in medicine. *Cybernetics and Systems*, 23:417–429, 1992.

[7] D. E. Heckerman and E. J. Horvitz. The myth of modularity in rule-based systems for reasoning with uncertainty. In J. F. Lemmer and L. N. Kanal, editors, *Uncertainty in Artificial Intelligence 2*, pages 23–34. Elsevier Science Publishers B.V., Amsterdam, 1988.

[8] M. Henrion. Propagation of uncertainty by logic sampling in Bayes' networks. In J. F. Lemmer and L. N. Kanal, editors, *Uncertainty in Artificial Intelligence 2*, pages 149–164. Elsevier Science Publishers B.V., Amsterdam, 1988.

[9] M. Henrion. Some practical issues in constructing belief networks. In L. N. Kanal, T. S. Levitt, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 161–173. Elsevier Science Publishers B.V., Amsterdam, 1989.

[10] E. Horvitz and D. Heckerman. The inconsistent use of measures of certainty in Artificial Intelligence research. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 137–151. Elsevier Science Publishers B.V., Amsterdam, 1986.

[11] 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.

[12] 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.

[13] D. Marr. *Vision.* Freeman, New York, 1982.

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

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

[16] J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32:245–257, 1987.

[17] J. Pearl. *Probabilistic Reasoning in Expert Systems.* Morgan Kaufmann, San Mateo, CA, 1988. Revised second printing, 1991.

[18] J. Pearl. Belief networks revisited. *Artificial Intelligence*, 59:49–56, 1993.

[19] J. Pearl and A. Paz. GRAPHOIDS: A graph-based logic for reasoning about relevance relations. Technical Report (R–53–L), Cognitive Systems Laboratory, University of California, Los Angeles, 1985.

[20] J. Pearl and T. S. Verma. The logic of representing dependencies by directed graphs. In *Proceedings of the 6th National Conference on AI (AAAI-87)*, pages 374–379, Seattle, WA, 1987.

[21] J. Pearl and T. S. Verma. A statistical semantics for causation. *Statistics and Computing*, 2:91–95, 1992.

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

[23] R. D. Shachter, S. T. Andersen, and P. Szolovits. The equivalence of exact methods for probabilistic inference on belief networks. Submitted to *Artificial Intelligence*, October 1991.

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

[25] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13:566–579, 1984.