Contents

1 Introduction **3**
1.1 Purpose and motivation . . . 3
1.2 Thesis contribution and organization . . . 5

2 Uncertainty representation **9**
2.1 Some uncertainty models . . . 10
2.1.1 Certainty factors . . . 10
2.1.2 Fuzzy logic . . . 10
2.1.3 Belief function . . . 10
2.2 Probability theory . . . 11
2.2.1 Marginal independence . . . 13
2.2.2 Conditional independence . . . 13
2.3 Bayesian networks . . . 15

3 Computations in Bayesian network **21**
3.1 Prevision computations . . . 22
3.2 Prevision using the graphical part of the BN . . . 24
3.3 Diagnostic computations . . . 25
3.4 Diagnostic using the graphical part of the BN . . . 30
3.5 Pearl's architecture: the polytree algorithm . . . 31
3.6 Propagation in multiply connected networks . . . 36
3.6.1 Global conditioning . . . 38
3.6.2 Local conditioning . . . 40

4 Undirected methods **43**
4.1 Markov Networks . . . 44
4.2 From Markov networks to join trees and junction trees . . . 48
4.3 Jensen's inference method . . . 53
4.4 Shafer-Shenoy's inference method . . . 57
4.5 Binary join trees . . . 65

5 A comparison between Pearl's and Jensen's algorithms **69**
5.1 Coping with loops in the Bayesian network . . . 69

5.2 *d*-separation analysis 75
5.3 Conclusion 79

6 Triangulated Bayesian Network 81
6.1 Constructing a new DAG: the triangulated BN 81
6.2 Arc reversal and the generation of new probabilities 84
6.3 An efficient variation of the polytree algorithm 86
6.4 Proofs 90

7 Binary join trees within Pearl's algorithm 101
7.1 Example of a propagation in a BJT 101
7.1.1 Computations in a nonbinary join tree 101
7.1.2 Computations in a binary join tree 103
7.2 Pearl's algorithm in a triangulated Bayes net 106
7.3 Simulating BJT's with Pearl's algorithm 109
7.3.1 Avoiding redundancies in Pearl's algorithm 109
7.3.2 Example of Pearl's propagation in a BJT 113

8 Updating undirected methods with local triangulation 119
8.1 Triangulation avoiding unnecessary fill-ins 121
8.2 A new algorithm for finding junction trees 123
8.3 Empirical results 128
8.4 Proofs 130

9 Conclusion 137

Chapter 1

Introduction

1.1 Purpose and motivation

Artificial Intelligence pays more and more attention to human reasoning under uncertainty. This concern is all the more important that there is a real need for softwares capable of performing automatically tedious tasks while respecting users preferences. Let us cite for instance spam filters which must be able to determine that a given e-mail is of no interest for the user, or intrusion detection systems which must classify internet packets according to what a system administrator might judge as a normal traffic or as an attempt to break into or to corrupt the system. We may also cite internet search engines that need isolate the data that are of interest to users. One of the common features of the above applications is that they must reason under uncertainty. For instance, an internet packet sent to a web server is, in itself, innocuous. However, if one million such packets are sent at the same time, this shall be judged as an attempt to perform a Denial of Service and thus the packets should be filtered before they reach the web server. Of course, in practice, determining the right action to take is less obvious as the data available to the software are more ambiguous. For instance, there exist some applications helping students learning various concepts like how to use properly the Unix system [Jam96], how to repair the hydraulic system of the F15 plane [MG96]. In such applications, when the student does not solve properly the problem he/she is asked to solve, the computer must try to understand what erroneous reasoning the user is having. Naturally, there may be several different erroneous reasonings and so the application must determine which one caused the failure and decide what is the best strategy to help the user even though it does not know with certainty what concept the user failed to understand. [PW95, FHKR95] describe another application where decisions must also be taken although the data available are imperfect: the plan of a driver must be detected simply by observing the movements of the car. All such problems require having a good model of the uncertainties they have to handle.

Different practical situations usually require different models of uncertainty as the amount and the quality of the data available vary from one application to the other. Hence, depending on the data, uncertainty may be dealt with using belief functions

[Sha76], fuzzy set theory, possibility theory, probability theory, etc. For instance, in medical decision making, large databases highlighting the connections between diseases and clinical reports are not uncommon, and probability theory is certainly a good uncertainty model for this case [FSMB91]. Other practical situations may suffer from a lack of data and, then, other alternative uncertainty models should be preferred. Yet, when there is sufficiently enough data and these are of good quality, probability theory should be used as: i) it has strong mathematical foundations; ii) it is easy to use; and iii) the conclusions that can be reached using this model are sound. Hence, in this thesis, for the management of uncertainties, we will abide to the laws of probability.

However, until a recent past, the computations of probabilities in complex situations proved to be impractical for two reasons: i) they were stored in extension, that is, the whole joint probability distributions had to be stored in computers, which was impossible for highly combinatorial spaces; ii) computing even a marginal probability over one random variable could take an unreasonable amount of time as too many computations were involved. Pearl [Pea88] realized that both problems could be fixed simply by decomposing the joint probability distributions, storing compactly the pieces involved in this decomposition and providing a clever way to combine the pieces when needed for computations. This led to *Bayesian networks*. These are graphs —actually directed acyclic graphs (DAG)— whose vertices represent random variables and whose arcs represent probabilistic dependencies between vertices/random variables. In itself the graphical structure of the Bayesian network represents the decomposition of the joint probability distribution of its vertices/random variables as the product of the conditional probabilities of each node conditionally to its parents in the graph. Using this decomposition and taking advantage of the graphical structure, efficient algorithms have been designed that can compute any joint, marginal or conditional probability of the random variables. These algorithms can even compute *a posteriori* probabilities, that is they can infer the impact of some evidence on the random variables of the network.

Although these algorithms differ widely from one another, they all share one common feature: their computations are achieved via a message passing scheme along the edges of a graph. In this respect, they can be called propagation algorithms as they propagate informations or the impact of evidence along the graph. In the 90's two broad classes of propagation methods emerged: i) those propagating messages along the arcs of oriented networks (the so-called Pearl's propagation scheme, see [KP83, Pea88, PS91, SC90]; and ii) those that first transform the Bayesian network into a non-oriented clique tree and, then, propagate evidence in this tree [LS88, Jen96, JLO90, MJ99, SS90b, She97]. Among the latter, the most popular are certainly Jensen's and Shafer-Shenoy's methods.

The undirected methods have superseded directed ones, especially after Shachter, Andersen and Szolovits showed in 1994 [SAS94] that, when Bayesian networks contain loops, all the known directed propagation methods were bound to be outperformed by undirected ones. This property resulted from the observation that the way Pearl's-like methods coped with cycles was a particular and inefficient subcase of Jensen's and Shafer-Shenoy's cycle management methods. Unfortunately, in most practical situa-

tions, Bayesian networks contain cycles. However, directed methods have advantages over undirected ones: the directed graphical structure contains more informations about independencies between random variables and these can be used to significantly reduce the computational burden of propagation. Hence, in this thesis, we are interested in improving Pearl's-like methods so that their cycle management is competitive with Jensen's while preserving the directed nature of the graph. Upon success, such an algorithm would not only be competitive with the state of the art propagation methods, it would even outperform them.

1.2 Thesis contribution and organization

As mentioned above, the aim of this thesis is to improve Pearl's algorithm to make it competitive with or even faster than undirected methods. The latter do not perform directly computations on the Bayesian network but rather they construct a more efficient undirected structure and, then, the computations are conducted using this structure. The latter is called a *junction tree* in Jensen's scheme and a *join tree* in Shafer-Shenoy's method. The basic idea underlying our improvement of Pearl's algorithm is to adapt the construction of this secondary structure to directed graphs. Thus our secondary directed structure should be as efficient as junction/join trees and, as it is directed, the independence informations contained in the graphical structure can be best exploited to reduce computations by avoiding those that are unnecessary. As our modification of Pearl's algorithm sort of unifies it with undirected methods, any improvement in undirected methods can be adapted to our method and, conversely, improvements stemming from Pearl can benefit to undirected methods. In this thesis, we provide two such improvements: we show how binary join trees can be adapted in our algorithm and, studying the relationships between conditioning and triangulations —the methods initially used for coping with loops by Pearl and Jensen respectively— we show how triangulations can be improved by breaking the Bayesian network into small parts, constructing junction trees for each small part and aggregating these separate junction trees to form a global junction tree. All these results are presented as follows:

Chapter 2 provides an overview of the theories dealing with uncertainty, and it justifies the use of probability theory as a preferred one. After a brief survey of the basic properties of probabilities, the chapter ends with an introduction to Bayesian networks.

In the third chapter, we detail how computations are done by Pearl in the Bayesian network. We first show how *prevision* and *Diagnostic* can be performed and, unifying them, we obtain Pearl's *polytree algorithm* [KP83, Pea88, PS91]. Unfortunately, this one can only be applied on singly-connected networks and, as mentioned above, most Bayesian networks are multiply-connected. Hence techniques for coping with loops have been designed. At the end of chapter 3 we study two such methods, namely *Global Conditioning* [Pea88] and *Local Conditioning* [Die96, FJ00].

Chapter 4 presents an overview of undirected propagation methods. These heav-

ily rely on Markov properties, so the beginning of the chapter is devoted to Markov properties and Markov networks. Then it is shown how the latter can be mapped into junction trees or join trees. Finally, propagation methods taking advantage of these tree structures are described, including Jensen's algorithm [JLO90, Jen96], Shafer-Shenoy's algorithm [SS90b] and even Shafer-Shenoy's algorithm for binary join trees [She97].

The fifth chapter can be thought of as a transition between the state of the art review and our own results. It discusses the advantages and drawbacks of Pearl's algorithm as compared to that of Jensen. Mainly, it explains why junction trees deal more efficiently with cycles than conditioning, but also why directed graphs are advantageous compared to undirected ones for determining whether evidence affect some random variables, thus making it possible to avoid unnecessary computations.

In the sixth chapter, we present our first improvement of Pearl's algorithm: that which makes the latter competitive with Jensen's algorithm. The chapter is organized as follows: first we show how a secondary structure similar to the triangulated Markov tree underlying Jensen's method can be obtained. We call this secondary directed graph a *triangulated Bayesian network* or TBN. This graph induces a new decomposition of the joint probability distribution of the random variables, so we describe how the conditional probabilities involved in this new decomposition can be retrieved. Finally, we show that the application of Pearl's algorithm with local conditioning in this structure has the same computational complexity as Jensen's algorithm.

Actually, the inference method presented in Chapter 6 is closer to Shafer-Shenoy than to Jensen. Indeed, no division is ever performed by our algorithm —or by Shafer-Shenoy— whereas there are some divisions in Jensen's method. Moreover, like Shafer-Shenoy, when our algorithm passes a message from a node X to a node Y, the latter is computed as some product involving messages sent to X by all of its neighbors except Y. It was shown that, although Shafer-Shenoy is as efficient as Jensen in complexity on every join tree, the latter usually performs fewer arithmetic operations, unless the join tree is binary [She97], in which case both algorithms perform the same amount of operations. Our algorithm suffers from the same problem: it is equivalent to Jensen in complexity but not in the number of arithmetic operations it performs. Hence we show in the seventh chapter how binary join trees can be adapted to our algorithm and thus the latter becomes as efficient as both Shafer-Shenoy and Jensen (both in the number of operations and in complexity).

In the eight chapter we study how triangulations can be improved. A toy example shows us that even state of the art triangulation methods used in the Bayes net community fail to detect that some of the edges they add to Markov trees are unnecessary. These edges unfortunately complexify the junction trees obtained from the Markov trees and decrease the efficiency of junction tree-based propagation algorithms. However, looking at the original Bayesian network, it can be easily seen that edges whose extremities do not belong to the same cycles can be dispensed with. Thus Chapter 8 presents a triangulation method avoiding such unnecessary edges. The underlying idea of the algorithm consists in breaking the Bayesian network into small parts, triangulating separately each part and computing their corresponding junction tree. Then it is

shown how these small junction trees can be combined to form an overall junction tree of the original Bayesian network. Empirical results show that this technique is very effective.

Finally, the last chapter concludes this thesis and suggests some perspectives for future research.

Chapter 2

Uncertainty representation

For about two thousand years, uncertainty has raised the interest of many philosophers — including Aristotle — and of mathematicians — like Cardano (1501-1576) and De Moivre (1667-1754) — who studied uncertainty through hazard games. Their studies led in the seventeenth and eighteenth centuries to the development of the probability theory (see Pascal (1623-1662), Bernoulli (1700-1782) or Laplace (1749-1827). And, of course, their concerns led them to base Probability Theory on infinitely reproducible experiments. It was not before the 20th century and Kolmogorov's axiomatization that Probability Theory became established in a more general framework where the states of nature could be, for instance, infinite.

Although Kolmogorov's theory was very attractive as it was not based on infinitely reproducible experiments, in practice, up to a recent past, estimating the probability of any event still required the observation of an infinite number — or at least a large number — of identical experiments. But in many practical situations, such a requirement could simply not be fulfilled. For instance, how can the probability of an earthquake in San Francisco be determined? In 1974, De Finetti [dF74] proposed an attractive solution: instead of observing the frequency of occurrence of events, probabilities can be estimated by experts. This led to subjected probability theory. Using some games involving some money aspect, the experts need assert probabilities correctly to expect maximizing the amount of money they get at the end of the game. Thus, in this theory, even probabilities of rare event can be estimated.

However, representing uncertainty by probabilities — either objective or subjective — still requires many informations and, in practice these may lack. For instance, the famous Ellsberg's urn problem cannot be represented by probabilities: in this problem, we know that an urn contains black, white and red balls and that one third are red and two third are white or black. As the proportion of white against black balls is unknown, no probability can be inferred. For such problems, alternative uncertainty representations exist that represent different knowledge. Let us cite for instance fuzzy logic, belied functions, etc.

2.1 Some uncertainty models

In this section, we recall briefly some classical models for representing uncertainty.

2.1.1 Certainty factors

The basic idea behind certainty factors is to allow the use of values between true and false in the interval [0,1]. Inference is applied with operators called *T-norms* defined as the product or the minimum of the certainty degrees.

Example 2.1 If proposition $P1$ is true with degree x and proposition $P2$ is true with degree y then proposition ($P1$ and $P2$) is true with degree $min(x, y)$ or with degree $x \times y$. ♦

Certainty factors served in the 80's as the foundation for well known Expert systems or the rule-based systems such as MYCIN [Sho76]. However, they were were criticized by many researchers —including [Pea88]— as their conclusions could be inconsistent.

2.1.2 Fuzzy logic

Sometimes we use qualitative terms like *almost, rather, enough, etc* that reflect *imprecision* rather than *uncertainty*, and we need a model to represent this kind of information: *it is almost empty, it is great enough*.

Fuzzy Logic tries to cope with imprecision and proposes new operators occurring on fuzzy sets and extends the interval [0,1] of classical set theory.

2.1.3 Belief function

In 1976, Shafer [Sha76] generalized probability theory, which resulted in *belief function theory* a.k.a. *theory of evidence* or *Dempster-Shafer theory*. The approach has many interpretations: *Lower probability model* [Jaffray, 1989; Walley, 1991]: where beliefs are represented by families of probability functions, *Dempster's model* [Kohlas and Monney, 1995]: derived from probability theory too, and *Transferable belief model* [Smets and Lennes, 1994; Smets 1998]: beliefs are quantified by belief function (BEL), not by probability.

According to the nature of the knowledge we have about the states of nature (or the events), uncertainty can be represented more faithfully by a model or another. However, Probability has very attractive mathematical foundations and, when enough information is available, it can be modeled and manipulated efficiently.

In the next section, we present and define probabilities, assuming that distribution of a random variable is known by repetitive observations or given by an expert.

2.2 Probability theory

In Probability Theory — at least the subpart we use in this thesis —, a random variable representing an event has a finite and discrete set of elementary values, e.g., the random variable representing the possible values obtained throwing a die may have the following support: $\{1, 2, 3, 4, 5, 6\}$.

Definition 2.1 (Probability) *Let Ω be a finite nonempty set, $\mathcal{A}$ a σ-algebra of Ω, and P a numerical function defined on $\mathcal{A}$.*
P is a probability on $(\Omega, \mathcal{A})$ if it satisfies the following three axioms:

$$\forall X \in \mathcal{A}, P(X) \geq 0; \tag{2.1}$$

$$P(\Omega) = 1; P(\emptyset) = 0; \tag{2.2}$$

$$\forall X, Y, \in \mathcal{A},\ \textit{if}\ X \cap Y = \emptyset,\ \textit{then}\ P(X \cup Y) = P(X) + P(Y). \tag{2.3}$$

In the following, we call *event* any element of $\mathcal{A}$, the intersection between X and Y is denoted indistinctly by XY, X, Y or $X \cap Y$, and the union is denoted by $X \cup Y$. Ω is the *certain event* and $\emptyset$ is the *impossible event*. Events X are called *elementary* when there do not exist “smaller” events in $\mathcal{A}$ the union of which is X.

According to Definition 2.1, it is possible to compute the probability of any event from the probabilities of elementary events. However, in practical situations, the events themselves are not of interest to the user but it is rather some function of the events — for instance their consequences — that are of interest.

Example 2.2 Assume you own \$1000 and let us play to the following game: I select arbitrarily 3 cards from a 32 cards deck and, depending on the cards that obtain, I give you or take from you the following amounts of money:

- if 3 kings obtained, you win \$2000;
- if 2 kings obtained, you win \$500;
- if 1 king obtained, you neither win nor lose any money;
- if no king obtained, you lose \$1000.

In this game, the events correspond obviously to the set of triples of cards than can obtain. Now, think about what is of concern to the player of this game. Is this the very triple that obtain or rather the amount of money he/she will get eventually? Of course, the second option seems more probable. In this case, the events of the game are not of interest but rather their (economic) consequences. ♦

It could thus be interesting to work directly with the gains rather than with the triples that can obtain. The tool for linking different related probabilistic spaces is called a *random variable*:

Definition 2.2 (Random variable) *Let Ω be a universe endowed with a probability $P(\cdot)$ and let Ω' another set. Let $\mathcal{A}$ and $\mathcal{A}'$ denote the σ-algebras of Ω and Ω' respectively. A random variable is a function $\Gamma : \mathcal{A} \mapsto \mathcal{A}'$ such that:*

$$\Gamma^{-1}(A') \in \mathcal{A} \quad \forall A' \in \mathcal{A}'.$$

A probability $P'(\cdot)$ over $\mathcal{A}'$ can then be defined as:

$$P'(A') = P\left(\Gamma^{-1}(A')\right) \quad \forall A' \in \mathcal{A}'.$$

$P'(\cdot)$ is called the probability distribution of random variable Γ. $\mathcal{A}'$ is called the support of Γ and for any $\gamma \in \mathcal{A}'$, $P'(\Gamma = \gamma)$ denotes the quantity $P\left(\Gamma^{-1}(\gamma)\right)$.

Definition 2.3 (probability distributions) *Let $(\Omega, \mathcal{A}, \mathcal{Q})$ be a probabilistic space.*

- *If X and Y are two random variables defined on $\mathcal{X}$ and $\mathcal{Y}$ respectively, then* **Joint probability distribution** *of X and Y is the function defined on $\mathcal{X} \times \mathcal{Y}$ by:*
$$P(\{X = x\} \cap \{Y = y\}) = \mathcal{Q}(X^{-1}(x) \cap Y^{-1}(y)).$$
  For simplicity of notation, it is usually denoted by $P(x, y)$. Since probabilities are additive, we can deduce from joint probabilities, the marginal probabilities.

- *If X and Y are two random variables defined on $\mathcal{X}$ and $\mathcal{Y}$ respectively, then the* **marginal probability distribution** *of X is the function defined by:*
$$P(X = x) = \sum_{y \in \mathcal{Y}} P(x, y).$$

- *If X and Y are two random variables defined $\mathcal{X}$ and $\mathcal{Y}$ respectively, and if $P(Y) > 0$ then the* **conditional probability distribution** *of X given a value of Y is the function defined by:*
$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)}.$$

The last definition, when used recursively on a set of random variables $\{X_1, \ldots, X_n\}$, yields the well known chain formula:

Definition 2.4 (chain formula)

$$P(X_1, \ldots, X_n) = P(X_1) \times \prod_{i=2}^{n} P(X_i | X_1, \ldots, X_{i-1}).$$

2.2.1 Marginal independence

Up to a recent past, the computer representation of probabilities was largely inspired from Kolmogorov's probability definition (Definition 2.1), that is, only the probabilities of elementary events were actually stored and they were used to compute any (marginal, conditional, joint) probability of interest. Unfortunately, the size required for such a representation grows exponentially with the number of random variables. For instance, the probability distribution over a set of n variables, each having a m-size support, requires the storage of m^n different values. As practical situations often involve numerous random variables, such a representation is often inadequate.

Reducing drastically the storage space is fortunately possible using independence relationships between random variables. Assume for instance that X and Y are *independent*, then it is well known that the joint probability $P(X, Y)$ is equal to the product of marginal probabilities of X and Y, that is $P(X, Y) = P(X)P(Y)$. Thus, applying this property recursively and assuming all the random variables are independent, it is no longer necessary to store m^n different values but only the $m \times n$ values of the marginal probabilities of the random variables. Then the probability of each elementary event can be computed as the product of the marginal probabilities of the random variables.

Example 2.3 Consider 100 dice with 10 faces each. The size of the space of all the sequences of the dice results is equal to 10^{100}. However, knowing that dice are independent, only the marginal probability of each die need be stored — which amounts to storing $100 \times 10 = 1000$ values — and the probability of any elementary event can be computed as the product of these marginal probabilities. For instance, the probability of having a sequence constituted only by 1's is equal to the product of each die being equal to 1. ♦

However, in practice, the random variables of interest are seldom independent and thus marginal independence is too strong a condition to be applied. Fortunately, another form of independence called *conditional independence* often holds and, although being weaker than marginal independence, it will enable us to store probabilities even in very complex practical situations.

2.2.2 Conditional independence

Definition 2.5 (conditional independence) *Let X, Y, and Z be three random variables, and $P(Z) > 0$. X* **is independent from** *Y* **conditionally to (or given)** *Z, denoted by $(X \perp\!\!\!\perp Y \mid Z)$, if $P(X|Y, Z) = P(X|Z)$.*

Intuitively, this means that if the value of Z is known, X and Y become independent, or equivalently that our knowledge about X is not changed by adding some new information about Y to the information given by the value of Z.

The combination of conditional independence and of the chain formula (see Definition 2.4) can prove to be very powerful as the latter can be drastically simplified

in practical situations. Indeed, if, for a given $i \in \{2, \ldots, n\}$, $\{1, \ldots, i-1\}$ can be partitioned into two sets, say K_i and L_i such that $K_i \cap L_i = \emptyset$ and such that the random variable X_i is independent of the X_l's, $l \in L_i$, conditionally to the set of random variables $\{X_k : k \in K_i\}$, then

$$P(X_i|X_1, \ldots, X_{i-1}) = P(X_i|(X_k, k \in K_i), (X_l, l \in L_i)) = P(X_i|X_k, k \in K_i).$$

Thus the chain formula can be restated as:

$$P(X_1, \ldots, X_n) = P(X_1) \times \prod_{i=2}^{n} P(X_i|X_k, k \in K_i),$$

and, in practical situations, as the right hand side of the above equation usually involves only small sized K_i's, the probability of even very complex problems involving numerous random variables can be stored into nowadays personal computers. As an illustration, let us see a toy example due to Lauritzen and Spiegelhalter [LS88]:

Example 2.4 (dyspnoea) Shortness-of-breath (dyspnoea) (D) may be due to tuberculosis (T), lung cancer (L) or bronchitis (B), or none of them, or more than one of them. A recent visit to Asia (A) increases the chances of tuberculosis, while smoking (S) is known to be a risk factor for both lung cancer and bronchitis. The result of a single test chest X-ray (X) does not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea.
According to the chain formula alone:

$$\begin{aligned} P(A, S, T, L, B, D, X) = {} & P(A)P(S \mid A)P(T \mid S, A)P(L \mid A, S, T) \\ & P(B \mid A, S, T, L)P(D \mid A, S, T, L, B) \\ & P(X \mid A, S, T, L, B, D). \end{aligned}$$

Smoking does not seem to have any connection with visiting Asia, hence it is reasonable to assume that $P(S \mid A) = P(S)$. Similarly, tuberculosis may be due to visiting Asia but it has no correlation with smoking, so $P(T \mid S, A) = P(T|A)$. As lung cancer may be caused by smoking but is unrelated to visits in Asia and tuberculosis, $P(L \mid A, S, T) = P(L \mid S)$. Now, it can be proved statistically that bronchitis and lung cancer are probabilistically dependent. A thorough analysis shows that this is due to that fact that smoking both increases the risk of bronchitis and lung cancer. But conditionally to the fact that we know someone smokes or does not smoke, they become independent. Hence $P(B \mid A, S, T, L) = P(B|S)$. The same kind of reasoning can be applied to show that $P(D \mid A, S, T, L, B) = P(D \mid T, L, B)$ and that $P(X \mid A, S, T, L, B, D) = P(X \mid T, L)$. Consequently the joint probability over A, S, T, L, B, D, X can be expressed as:

$$\begin{aligned} P(A, S, T, L, B, D, X) = {} & P(A)P(S)P(T \mid A)P(L \mid T)P(B \mid S) \\ & P(D \mid T, L, B)P(X \mid T, L). \end{aligned}$$

Note that if each random variable can take 10 values, then storing P via its value for every elementary event requires storing 10^7 values whereas storing only the conditional probabilities in the right hand side of the above equation requires only 11320 values. ♦

In conclusion, the combination of the chain formula and conditional independence enables the storage of probabilities of even very large spaces. Of course, as the joint probability distribution is factorized, the computations of probabilities of interest (conditional, marginal, joint, etc) are more complicated to conduct and, at first sight, seem less obvious to automatize. Fortunately, representing the dependencies between random variables through graphs and applying techniques borrowed from graph theory, computations can be performed in a very systematic way, hence enabling computer programs to do them. The graphs we will study in this thesis are sometimes called Bayesian networks, or Bayesian belief networks, or even probabilistic networks.

2.3 Bayesian networks

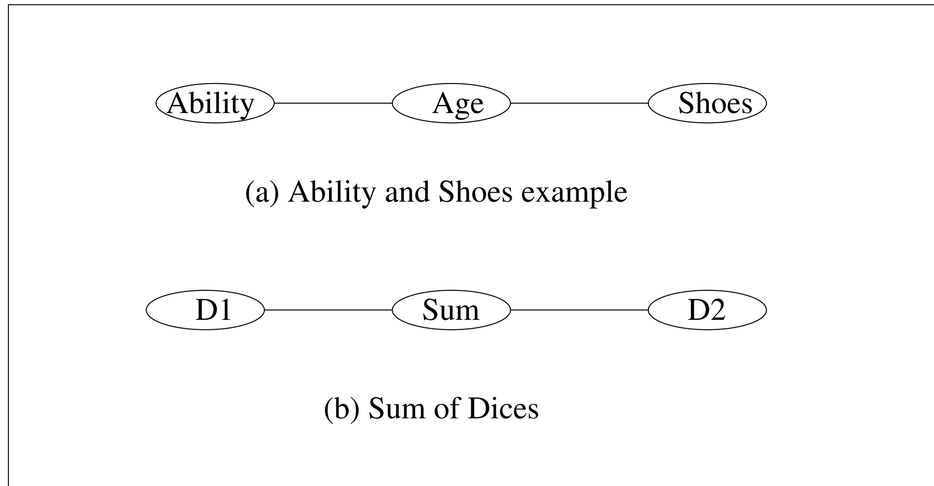
Of course, many different kinds of graphs may be used to represent the dependencies between random variables, but the most basic idea would be to create a graph the nodes of which would represent the random variables and, for each pair of dependent variables, there would exist an undirected edge (why should the edge be directed as probabilistic dependence is a symmetric concept?). However, the following examples show that such graphs might prove to be misleading:

Example 2.5 A very serious empirical study shows that there is a strong dependence between people's reading ability (variable $Ability$) and their shoe sizes (variable $Shoes$). Although surprising at first sight, the mystery disappears when another variable is taken into account, that is, the age of the person. Of course, young children have very small shoes and are not very good at reading whereas older people have larger shoes and may be expected to read better. Consequently, $Ability$ and $Shoes$ are dependent only through Age and the following relations should hold:

$$Ability \perp\!\!\!\perp shoes \mid Age \qquad \text{and} \qquad Ability \not\perp\!\!\!\perp Shoes. \tag{2.4}$$

As $Ability$ and $Shoes$ are dependent only through Age, an intuitive graphical representation would be that of Figure 2.1(a). ♦

Example 2.6 Consider now two dice $D1$ and $D2$. We throw them and denote by Sum the sum of their values. If the dice are not loaded, it is quite reasonable to assume that they are independent, that is knowing the value of one of them does not give any information about the value of the other one. However, knowing the value of Sum, they become dependent. Indeed, knowing that $sum = 4$, then if $D1 = 1$ we know for sure that $D2 = 3$. Representing graphically the dependencies between



(a) Ability and Shoes example

(b) Sum of Dices

Figure 2.1: Some graphs representing dependencies/independencies between random variables.

$D1$, $D2$ and Sum seems quite trivial: $D1$ (resp. $D2$) is of course related to Sum, but $D1$ and $D2$ are not related, thus the graph of Figure 2.1(b) should represent perfectly the dependencies:

$$D1 \not\perp\!\!\!\perp D2 \mid Sum \qquad \text{and} \qquad D1 \perp\!\!\!\perp D2. \tag{2.5}$$

♦

Now the problem raised by the above two examples is that Equation (2.4) is the opposite of Equation (2.5) and yet both equations lead to the same graph. Consequently, undirected graphs — at least those of the above kind — are not sufficient to represent all the informations we have about probabilistic dependencies. To discriminate between Example 2.5 and Example 2.6, a basic idea — that will fortunately work in all practical situations — is to add directions to edges (arcs). For example, the graphs of Figure 2.1 could be advantageously replaced by those of Figure 2.2.

Note that, according to the chain formula,

$$P(Age, Ability, Shoes) = P(Age)P(Ability|Age)P(Shoes|Age, Ability)$$

and, by Equation 2.4, $P(Shoes|Age, Ability) = P(Shoes|Age)$. Thus

$$P(Age, Ability, Shoes) = P(Age)P(Ability|Age)P(Shoes|Age).$$

In the graph of Figure 2.2(a), the joint probability distribution can thus be expressed as the product of the probabilities of each node/random variable conditionally to its parents in the graph. Similarly, according to Equation 2.5,

$$P(D1, D2, Sum) = P(D1)P(D2)P(Sum|D1, D2),$$

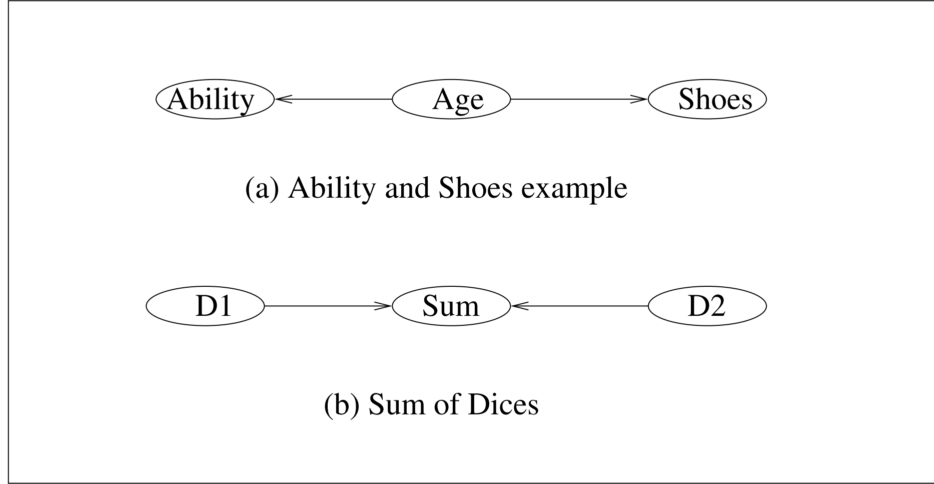


Figure 2.2: Examples of directed graph.

which corresponds again to the product of the probabilities of each node conditionally to its parents in the graph.

This suggests the concept of a Bayesian network: this is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ containing a finite set of nodes $\mathcal{V}$ representing random variables, and a finite set $\mathcal{A}$ of directed edges linking pairs of variables in $\mathcal{V}$. Once the BN is designed, each node of $\mathcal{V}$ stores its probability conditionally to its parents and the product of these conditional probabilities is equal to the joint distribution of the random variables in $\mathcal{V}$. Thus the graph contains a complete description of the joint probability distribution. It can then be used to compute any marginal, joint or conditional probability.

More formally, let $\mathcal{V} = \{X_1, \ldots, X_n\}$ be a set of random variables. As mentioned above, the chain formula states that:

$$P(X_1, \ldots, X_n) = P(X_1) \times \prod_{i=2}^{n} P(X_i | X_1, \ldots, X_{i-1})$$

and, if for every $i \in \{2, \ldots, n\}$, $\{1, \ldots, i-1\}$ can be partitioned into two sets, say K_i and L_i such that $K_i \cap L_i = \emptyset$ and such that the random variable X_i is independent of the X_l's, $l \in L_i$, conditionally to the set of random variables $\{X_k : k \in K_i\}$, then

$$P(X_1, \ldots, X_n) = P(X_1) \times \prod_{i=2}^{n} P(X_i | X_k, k \in K_i).$$

If graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ is designed so that $\mathcal{A} = \{(X_k, X_i) : i \in \{1, \ldots, n\}, k \in K_i\}$, then the joint probability distribution of $\mathcal{V}$ is equal to:

$$P(X_1, \ldots, X_n) = P(X_1) \times \prod_{i=2}^{n} P(X_i \mid (X_k, k \in \{k' : (X_{k'}, X_i) \in \mathcal{A}\})).$$

$(X_k, k \in \{k' : (X_{k'}, X_i) \in \mathcal{A}\})$ represents the set of variables having some outgoing arc toward X_i. They are also called parents of X_i and denoted by $Pa(X_i)$. Then, our joint probability over $\mathcal{V}$ can be formulated as:

$$P(X_1, \ldots, X_n) = P(X_1) \times \prod_{i=2}^{n} P(X_i | Pa(X_i)).$$

This leads to the definition of Bayesian networks:

Definition 2.6 (Bayesian network) *A Bayesian network (BN) is a triple* $(\mathcal{V}, \mathcal{A}, \mathcal{P})$ *such that:*

1. $\mathcal{V} = \{X_1, \ldots, X_n\}$ *is a set of random variables;*
2. $\mathcal{A} \subset \mathcal{V} \times \mathcal{V}$ *is a set of arcs which, together with* $\mathcal{V}$*, constitutes a directed acyclic graph (DAG),* $\mathcal{G} = (\mathcal{V}, \mathcal{A})$*;*
3. P *is the set of conditional probabilities of each node in* $\mathcal{V}$ *given its parents in the graph* $\mathcal{G}$*, i.e.,* $\mathcal{P} = \{P(X_i \mid Pa(X_i)) : X_i \in \mathcal{V}\}$.

Example 2.4 (continued) Remember that, in the dispnoea example, the joint probability distribution can be expressed as:

$$\begin{aligned} P(A, S, T, L, B, D, X) = {} & P(A)P(S)P(T \mid A)P(L \mid T)P(B \mid S) \\ & P(D \mid T, L, B)P(X \mid T, L). \end{aligned}$$

Usually, in this example, to avoid cycles, a new variable E is introduced, that represents an "or" operation between boolean variables T and L. Then the joint probability distribution can be factorized as:

$$\begin{aligned} P(A, S, T, L, E, B, D, X) = {} & P(A)P(S)P(T \mid A)P(L \mid T)P(B \mid S) \\ & P(E|T, L)P(D \mid E, B)P(X \mid E). \end{aligned} \tag{2.6}$$

The idea, here, is that the introduction of E still further reduces the space required to store the conditional probability tables (if every random variable can take 10 different possible values, we just need 2330 real numbers to store the conditional probability tables instead of 11320 numbers needed before the introduction of E). The Bayesian network corresponding to the factorization of Equation (2.6) is thus given by the graph of Figure 2.3. ♦

Bayesian networks are thus very compact representations of joint probability distributions. As we will see in the next chapter, they are also very powerful in that their graphical structure can be exploited to derive very efficient algorithms for computing any probability (conditional, marginal, joint).

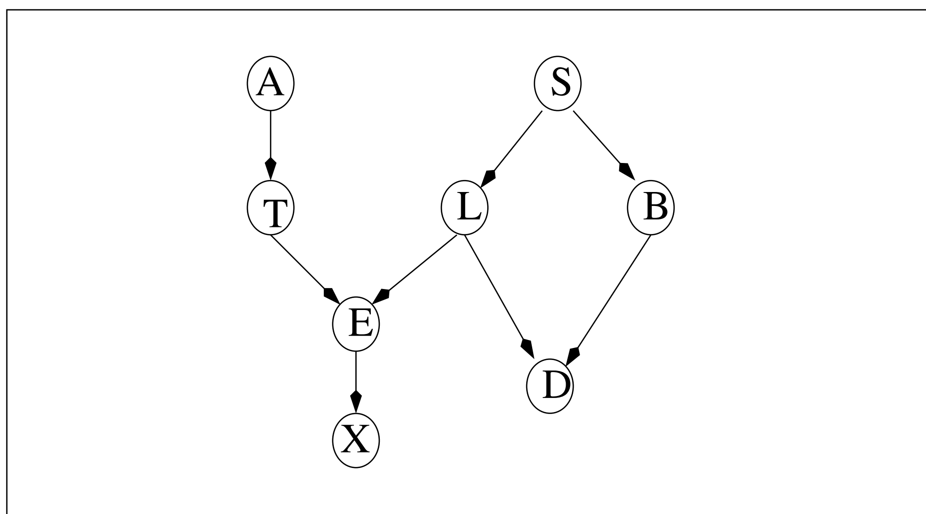


Figure 2.3: The dyspnoea example.

Chapter 3

Computations in Bayesian network

We showed in the preceding chapter that Bayesian networks hold an accurate decomposition of joint probabilities over sets of random variables. However, we did not show how these networks could be used in order to perform probabilistic computations. This is the object of the present chapter. In particular, we detail the different computations required to determine both *a priori* or *a posterior* probabilities. The latter corresponds to conditional probabilities given some new piece of information. For instance, in the dispnoea example (Example 2.4), $P(T)$ is an a priori probability, it reflects the probability of any person in the world to have a tuberculosis, whereas $P(T|A = true)$ is an a posteriori probability: it simply corresponds to the probability of a person having a tuberculosis when the person is known to have gone to Asia.

Roughly speaking, a Bayesian network can be used for two main purposes:

- **performing prevision:** if the arcs of the Bayesian network are oriented in the direction of causality, that is arc (X, Y) means that X is a cause of Y, then computing the probability of some node at the bottom of the network (for instance D in the graph of Figure 2.3) given some knowledge about a node at the top of the graph (for instance A) consists in inferring the impact of a cause to a consequence. It thus corresponds to a prevision that the cause will have such or such consequence.

- **performing diagnostic:** diagnostic is exactly the converse: if we observe some consequence, can we find its cause? In Bayesian networks terms, can we infer the impact of lower nodes in the network to the upper nodes?

Section 3.1 illustrates on an example the computations occurring for prevision purposes and Section 3.2 restates these computations in terms of graphical operations. Sections 3.3 and 3.4 are similar but illustrate diagnostic computations instead of previsions. All these computations will be generalized in a single algorithm introduced by Pearl [Pea88]: the polytree algorithm. Unfortunately this algorithm does perform consistent computations only in singly-connected networks, that is, networks without cycles. Hence Section 3.6 will conclude this chapter with extensions of the polytree algorithm capable of coping with multiply-connected networks.

3.1 Prevision computations

Consider the decomposition of the joint probability induced by the graph of Figure 3.1, that is:

$$
\begin{aligned}
P(A,B,C,D,E,F,G) = {} & P(A)P(B)P(C \mid A,B)P(D \mid C) \\
& P(E \mid C)P(F \mid C)P(G \mid E).
\end{aligned}
\tag{3.1}
$$

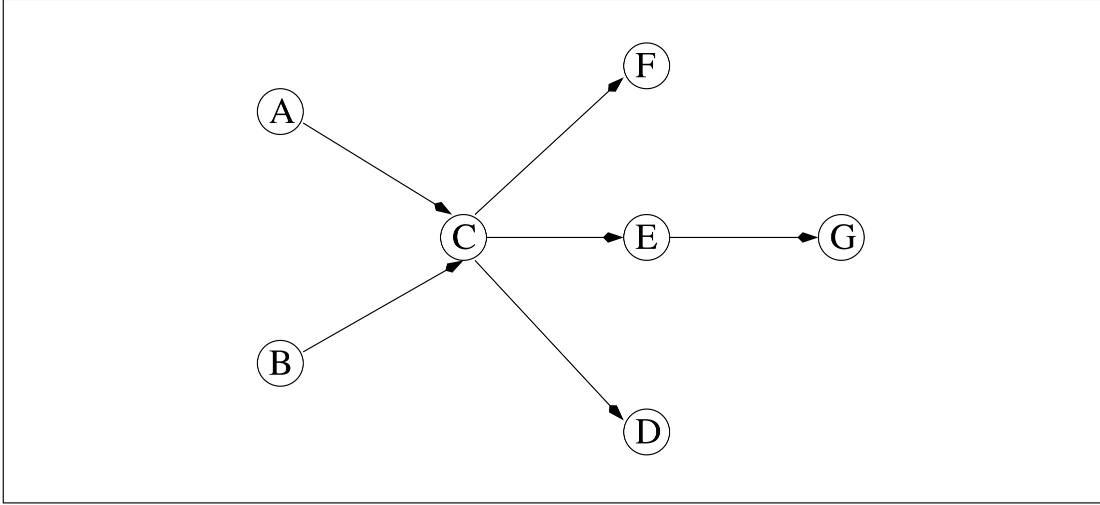


Figure 3.1: Decomposition of a joint probability.

Computing the marginal probability of any random variable simply amounts to sum out over the other variables $P(A,B,C,D,E,F,G)$. Thus

$$
P(G) = \sum_{A,B,C,D,E,F} P(A,B,C,D,E,F,G).
$$

As mono-processor computers can only perform one operation at a time, this summation can be completed as:

$$
\begin{aligned}
P(G) &= \sum_{A,B,C,D,E,F} P(A,B,C,D,E,F,G) \\
&= \sum_E (\sum_C (\sum_D (\sum_F (\sum_B (\sum_A P(A,B,C,D,E,F,G)))))).
\end{aligned}
\tag{3.2}
$$

Computing $P(G)$ thus naturally leads to the following computations:

1. We begin by summing out A from $P(A,B,C,D,E,F,G)$:

$$
\begin{aligned}
\sum_A P(A,B,C,D,E,F,G) = {} & \left(\sum_A P(A)P(C \mid A,B) \right) P(B) \\
& P(D \mid C)P(E \mid C)P(F \mid C)P(G \mid E).
\end{aligned}
$$

As A and B are independent (see the graph), $P(A) = P(A \mid B)$ and, consequently,

$$\begin{aligned}\sum_A P(A)P(C \mid A, B) &= \sum_A P(A \mid B)P(C \mid A, B) \\ &= \sum_A P(C, A \mid B) = P(C \mid B).\end{aligned}$$

Thus:

$$\begin{aligned}\alpha = \sum_A P(A, B, C, D, E, F, G) = P(C \mid B)P(B)P(D \mid C) \\ P(E \mid C)P(F \mid C)P(G \mid E).\end{aligned}$$

2. We can now sum (aggregate) on B:

$$\begin{aligned}\beta = \sum_B \alpha &= \left(\sum_B P(C \mid B)P(B)\right) P(D \mid C)P(E \mid C)P(F \mid C)P(G \mid E) \\ &= \left(\sum_B P(C, B)\right) P(D \mid C)P(E \mid C)P(F \mid C)P(G \mid E) \\ &= P(C)P(D \mid C)P(E \mid C)P(F \mid C)P(G \mid E).\end{aligned}$$

3. Then, we marginalize on F:

$$\begin{aligned}\gamma = \sum_F \beta &= \left(\sum_F P(F \mid C)\right) P(C)P(D \mid C)P(E \mid C)P(G \mid E) \\ &= P(C)P(D \mid C)P(E \mid C)P(G \mid E).\end{aligned}$$

4. Similarly, summing over D results in:

$$\begin{aligned}\delta = \sum_D \gamma &= \left(\sum_D P(D \mid C)\right) P(C)P(E \mid C)P(G \mid E). \\ &= P(C)P(E \mid C)P(G \mid E).\end{aligned}$$

5. Summing over C amounts to compute:

$$\begin{aligned}\epsilon = \sum_C \delta &= \left(\sum_C P(C)P(E \mid C)\right) P(G \mid E) \\ &= \left(\sum_C P(E, C)\right) P(G \mid E) = P(E)P(G \mid E).\end{aligned}$$

6. Finally, summing over E results in

$$\sum_E \epsilon = \sum_E P(E)P(G \mid E) = \sum_E P(E, G) = P(G).$$

3.2 Prevision using the graphical part of the BN

In this section, we show the relationship between the computations performed in the preceding section and the graphical structure of the Bayesian network. We know that initially each node in a Bayesian network stores its conditional probability given the values of its parents (see Figure 3.2).

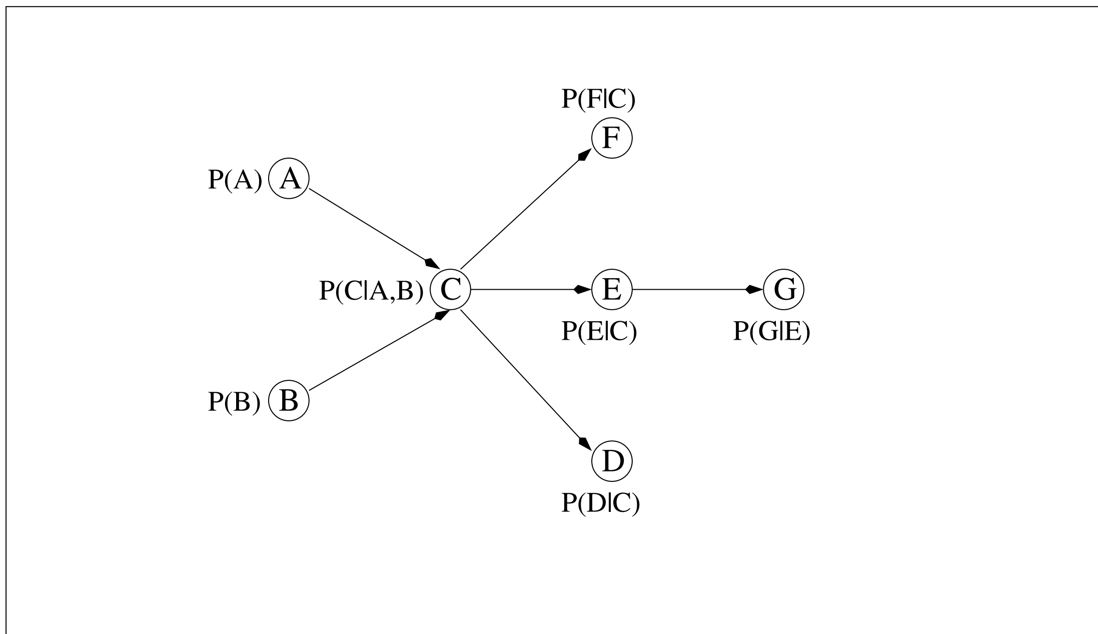


Figure 3.2: The conditional probabilities stored into the network.

1. The first computation performed in the preceding section was:

$$\sum_A P(A)P(C \mid A, B).$$

   In the graph of Figure 3.2, $P(A)$ is stored into node A and $P(C|A, B)$ is stored into C. Hence none of these node has sufficient knowledge to perform the above product and summation. To do so, either A should send information $P(A)$ to C or C should send $P(C|A, B)$ to A. Here the former will be applied. Hence A sends some message $P(A)$ to C which, in turn, can compute and store $P(C|B) = \sum_A P(A)P(C \mid A, B)$. This message is illustrated on Figure 3.3(a).

2. Summing over B involved computing:

$$\sum_B P(C \mid B)P(B).$$

Here again, probability $P(B)$ is stored into B and $P(C \mid B)$ is stored into C. Thus, B will send a message $P(B)$ and C will be able to compute the product and summation and will store the result, i.e., $P(C)$. This is illustrated on Figure 3.3(b).

3. Summing over F amounted to compute $\sum_F P(F \mid C)$, which can be performed simply by F since $P(F \mid C)$ is stored into F. In matrix terms, $\sum_F P(F \mid C)$ is equal to a vector of size $|C|$ constituted only by 1's.

4. Similarly, summing over D, which only amounts to compute $\sum_D P(D \mid C)$, which results in a vector of size $|C|$ constituted only by 1's.

5. The summation over C is performed by:

$$\sum_C P(C)P(E \mid C).$$

   As $P(C)$ is now stored into C and $P(E \mid C)$ is stored in E, either C must send a message to E or E must send a message to C. Here, C will send message $P(C)$ to E. To bring the message passing algorithm closer to the polytree algorithm, C will in fact receive the messages of size $|C|$ from F and D, it will multiply these messages by the value it already stored, i.e., $P(C)$, and then it will send the resulting message, i.e., $P(C)$ to E. Now E can perform the product $P(C)P(E \mid C)$ and sum it over C, hence resulting in $P(E)$. The process is illustrated on Figure 3.3(c).

6. summing over E yields computing $\sum_E P(E)P(G \mid E)$. Thus E needs sending message $P(E)$ and G can finally compute both the product and the summation. See Figure 3.3(d).

As can be seen, we can deduce that prevision is based on a **node elimination process**. Indeed, summing out a given variable can be thought of as eliminating the corresponding node from the Bayesian network (as shown in Figure 3.3).

3.3 Diagnostic computations

In this section, we consider computations conducted in the other direction of the edges, i.e., messages are transmitted from the leaves (the nodes at the bottom of the graph) to the roots (the upper nodes). If the edges are not only interpreted in terms of probabilistic dependencies but also in terms of causal relationships, that is, an edge (X, Y) means that X is a cause of Y, then sending messages from the "lower" nodes to the upper ones can be interpreted as computing the probabilities of "causes" given some known consequences, or equivalently as performing a diagnostic task.

Thus, assume some piece of information on the lower nodes is inserted into the network. Say for instance that new informations (a.k.a. evidence) e_i's with respect to

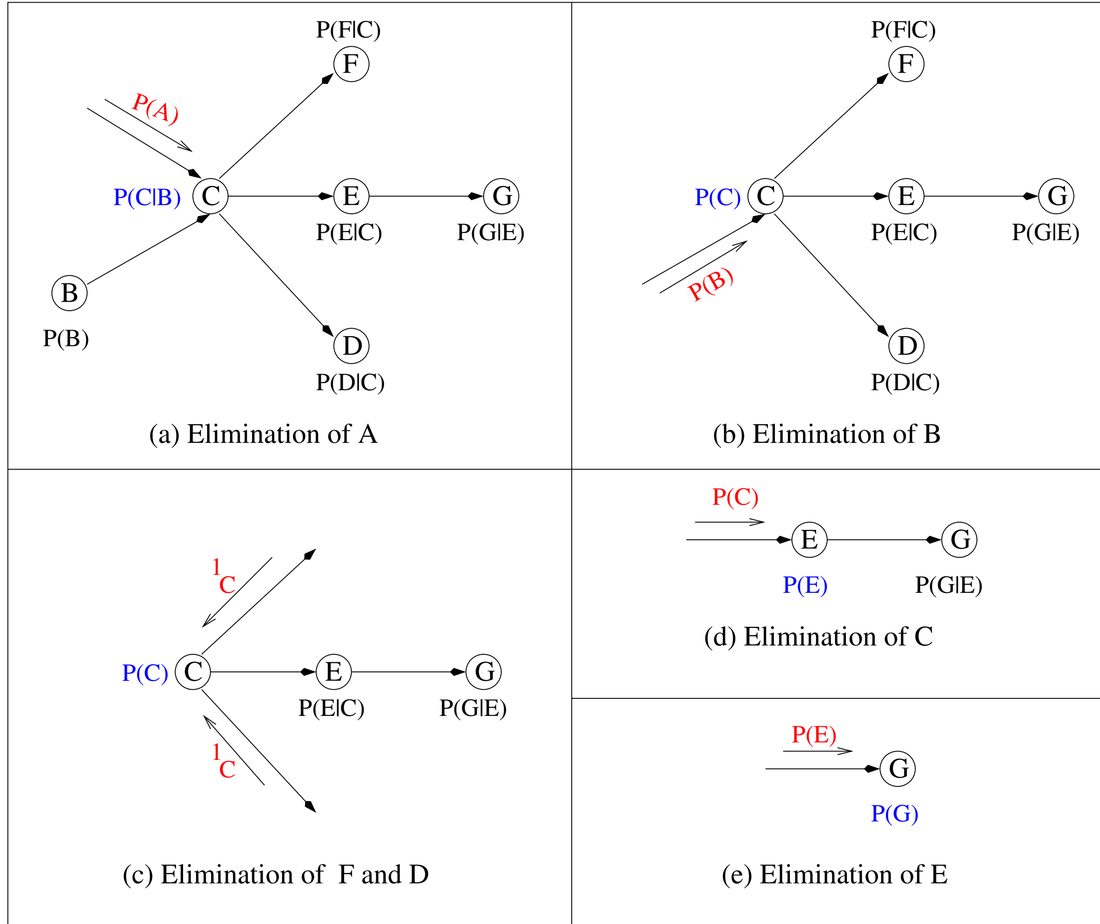


Figure 3.3: The prevision computations performed on the graphical structure.

nodes X_i's are known. Let $e = \{e_i\}$. The diagnostic task simply consists in computing for every node X (and especially the upper ones) probability $P(X|e)$. The computation we will perform will be based on the following quite "natural" assumption:

Hypothesis 3.1 *Let e_i be an evidence on node/variable X_i. Then e_i is independent from the rest of the graph and the other evidences conditionally to X_i.*

This hypothesis seems quite reasonable since it can be restated as: once the value of X_i is known, $\mathcal{V}\backslash\{X_i\}$ is independent from e_i. But if you already know the value of X_i, which evidence e_i can give you additional probabilistic information about X_i? None since the most informative piece of information about X_i, that is its value, is already known. Thus, once the X_i's value is known, e_i does not bring any further information to the rest of the graph, hence $e_i \perp\!\!\!\perp (\mathcal{V}\backslash\{X_i\})|X_i$.

As for the prevision process, we illustrate the probabilistic computations in Figure 3.4, and we suppose that G receives evidence $\mathbf{e_G}$, so that the *a posteriori* joint probability corresponds to the expression given in the following equation:

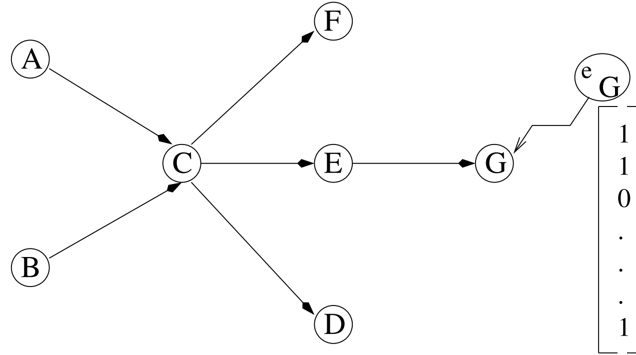


Figure 3.4: New evidence entries.

$$P(A,B,C,D,E,F,G,e_G) = P(e_G \mid A,B,C,D,E,F,G)P(A,B,C,D,E,F,G).$$

Since by Hypothesis 3.1 e_G is independent from $\{A,B,C,D,E,F\}$ conditionally to G, the above equation simplifies to:

$$P(A,B,C,D,E,F,G,e_G) = P(e_G \mid G)P(A,B,C,D,E,F,G).$$

Now, one may wonder what kind of information is contained into $P(e_G \mid G)$. Usually, e_G simply states that some values that could potentially be taken by random variable G cannot obtain anymore — sometimes, the evidence may even be so informative that there remains only one possible value for the random variable and the latter is thus known for sure. Assume for instance that e_G states that variable G that, previously, could take values $1,2,3,4,5$ cannot take values 1 and 2 anymore, i.e., e_G = "G can only take values 3,4,5". $P(e_G|G = 1)$ corresponds to the probability of observing evidence e_G when the value of G is 1. Thus $P(e_G|G = 1) = 0$ since it is not possible to observe that G cannot take value 1 when it actually has this value. Conversely, $P(e_G|G = 3) = 1$ since the information that G cannot take values 1 and 2 holds when it is known that the value of G is actually 3. Consequently $P(e_G|G)$ is a vector of size $|G|$ filled with values 0 and 1: $(0,0,1,1,1)$.

As for section 3.1, we shall see how computations of the a posteriori probability of a random variable can be performed using the decomposition of the joint probability distribution. So consider the distribution illustrated on Figure 3.4, that is:

$$P(A,B,C,D,E,F,G) = P(A)P(B)P(C|A,B)P(D|C)P(E|C)P(F|C)P(G|E).$$

Let us compute $P(A|e_G)$. We know that $P(A|e_G) = P(A, e_G)/P(e_G)$ and that $P(e_G) = \sum_A P(A, e_G)$. Hence:

$$P(A|e_G) = \frac{P(A, e_G)}{\sum_A P(A, e_G)}.$$

Moreover,

$$\begin{aligned} P(A, e_G) &= \sum_{B,C,D,E,F,G} P(A, B, C, D, E, F, G, e_G) \\ &= \sum_B \left(\sum_C \left(\sum_D \left(\sum_F \left(\sum_E \left(\sum_G P(A, B, C, D, E, F, G, e_G) \right) \right) \right) \right) \right), \end{aligned}$$

and, according to the decomposition of the joint probability distribution,

$$\begin{aligned} P(A, e_G) = \sum_B \sum_C \sum_D \sum_F \sum_E \sum_G & P(A)P(B)P(C|A, B)P(D|C)P(E|C) \\ & P(F|C)P(G|E)P(e_G|G). \end{aligned}$$

The computations of these summations can thus be performed as follows:

1. Summing out G amounts to calculate:

$$\begin{aligned} \alpha = \sum_G P(A, B, C, D, E, F, G, e_G) = \left(\sum_G P(G|E)P(e_G|G) \right) & P(A)P(B) \\ P(C|A, B)P(D|C)P(E|C)P(F|C). & \end{aligned}$$

But as e_G is independent of E conditionally to G (this is hypothesis 3.1),

$$P(G|E)P(e_G|G) = P(G|E)P(e_G|G, E) = P(G, e_G|E).$$

Hence $\sum_G P(G|E)P(e_G|G) = P(e_G|E)$ and:

$$\alpha = P(e_G|E)P(A)P(B)P(C|A, B)P(D|C)P(E|C)P(F|C).$$

2. Summing out E from α is computed by:

$$\beta = \sum_E \alpha = \left(\sum_E P(e_G|E)P(E|C) \right) P(A)P(B)P(C|A, B)P(D|C)P(F|C).$$

Here again, it can be shown that e_G is independent of C conditionally to E, so that:

$$P(e_G|E)P(E|C) = P(e_G, E|C).$$

Hence $\sum_E P(e_G|E)P(E|C) = \sum_E P(e_G, E|C) = P(e_G|C)$ and:

$$\beta = P(e_G|C)P(A)P(B)P(C|A, B)P(D|C)P(F|C).$$

3. Let us now sum out F:

$$\gamma = \sum_F \beta = \left(\sum_F P(F|C)\right) P(e_G|C)P(A)P(B)P(C|A,B)P(D|C)$$
$$= \mathbb{1}_C P(e_G|C)P(A)P(B)P(C|A,B)P(D|C),$$

where $\mathbb{1}_C$ is a vector of size $|C|$ filled only with ones.

4. Summation over D:

$$\delta = \sum_D \gamma = \left(\sum_D P(D|C)\right) \mathbb{1}_C P(e_G|C)P(A)P(B)P(C|A,B)$$
$$= \mathbb{1}_C \mathbb{1}_C P(e_G|C)P(A)P(B)P(C|A,B).$$

5. Summation over C:

$$\epsilon = \sum_C \delta = \left(\sum_C \mathbb{1}_C \mathbb{1}_C P(e_G|C)P(C|A,B)\right) P(A)P(B).$$

Here it may be worth mentioning that multiplying probability $P(e_G|C)$ by vector $\mathbb{1}_C$ will result in $P(e_G|C)$ simply because this product — as well as all the other products w.r.t. probabilities we perform — is a tensorial product. Hence:

$$\epsilon = \left(\sum_C P(e_G|C)P(C|A,B)\right) P(A)P(B).$$

Again, it can be shown that e_G is independent of A and B conditionally to C, so that:

$$\sum_C P(e_G|C)P(C|A,B) = \sum_C P(e_G|A,B,C)P(C|A,B)$$
$$= \sum_C P(e_G, C|A,B) = P(e_G|A,B).$$

Hence:

$$\epsilon = P(e_G|A,B)P(A)P(B).$$

6. Finally, let us marginalize out B:

$$\sum_B P(e_G|A,B)P(A)P(B) = \sum_B P(e_G|A,B)P(A)P(B|A)$$
$$= \sum_B P(e_G, B|A)P(A)$$
$$= P(e_G|A)P(A) = P(e_G, A).$$

Of course, computing $P(B|e_G)$ would be performed similarly and, more generally, this would apply to the computation of the *a posteriori* probability of any "cause" conditionally to the observation of some consequence (symptom, etc).

3.4 Diagnostic using the graphical part of the BN

As for section 3.2 the above computations can be illustrated on the graphical part of the BN: see Figure 3.5.

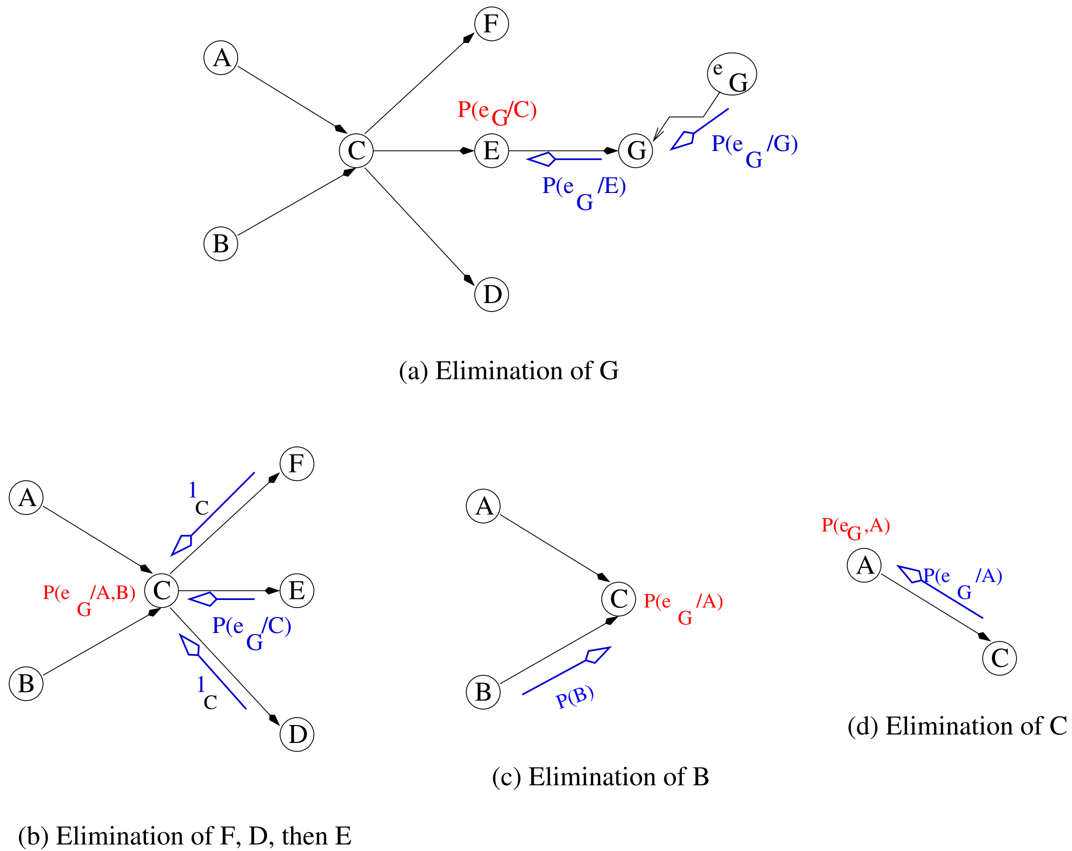


Figure 3.5: The diagnostic process.

Let us reconsider the computations of the preceding section in terms of messages sent along the edges of the graph:

1. Summing out G amounted to compute $\sum_G P(G|E)P(e_G|G)$. Of course, since both conditional probabilities are stored in G, no message need be sent and G can perform the product, the summation and store the result in its node.

2. Summing out E corresponds to the computation of $\sum_E P(e_G|E)P(E|C)$. As $P(e_G|E)$ is stored in G and $P(E|C)$ is in E, either G or E needs sending a message containing the conditional probability required by the other node to compute the product and the summation. Here, G will send to E a message containing $P(e_G|E)$ and E will perform the computation and store the result, i.e., $P(e_G|C)$, in its node. This is illustrated on Figure 3.5(a).

3. Let us now sum out F. This corresponds to $\sum_F P(F|C)$. As this probability is stored in F, the latter can compute the sum and store the result, i.e., $\mathbb{1}_C$ in F.

4. The summation over D ($\sum_D P(D|C)$) is performed similarly in D, so that D now stores vector $\mathbb{1}_C$.

5. The summation over C is a little more tricky. It amounts to:

$$\sum_C \mathbb{1}_C \mathbb{1}_C P(e_G|C)P(C|A,B).$$

   Vectors $\mathbb{1}_C$ were those obtained from marginalizing out F and D respectively. Hence, before computation, they are stored in F and D respectively. Moreover, $P(e_G|C)$ and $P(C|A,B)$ are kept in E and C respectively. So, to perform the product/sum, one node, say C must receive all the conditional probabilities stored in the other nodes and, thus, D, E and F will send to C messages $\mathbb{1}_C$, $\mathbb{1}_C$ and $P(e_G|C)$ respectively. C will then multiply these messages with $P(C|A,B)$, sum over C and keep the result, i.e., $P(e_G|A,B)$. The process is illustrated on Figure 3.5(b).

6. Finally, marginalizing out B is achieved performing first $\sum_B P(e_G|A,B)P(B)$ and then $P(e_G|A)P(A)$. In the first expression, $P(B)$ belongs to node B and $P(e_G|A,B)$ is kept in C, so B will send message $P(B)$ to C and C will compute $\sum_B P(e_G|A,B)P(B) = P(e_G|A)$ (see Figure 3.5(c)). Then C will send message $P(e_G|A)$ to A and the latter will finally compute $P(e_G|A)P(A) = P(e_G, A)$ (see Figure 3.5(d)).

As can be seen, messages sent and computations performed during a diagnostic inference process are quite similar to those of a prevision process. And, indeed, as we shall see in the next section, a general inference (a.k.a. propagation) scheme can be deduced from the above computations.

3.5 Pearl's architecture: the polytree algorithm

In this section, we present a variant of Pearl's algorithm [KP83] due to Peot and Shachter [PS91] and capable of computing the *a priori* and the *a posteriori* marginal probabilities of each node/variable in the network as long as this one is singly-connected. We postpone the problems raised by multiply-connected networks to the next section.

Pearl's propagation algorithm([KP83] and [Pea88]) can be explained in the subgraph of Figure 3.6. We assume that evidence e has been entered into the network. Consider a node X whose parents and children are respectively $U_1, \dots, U_n$ and $Y_1, \dots, Y_m$. Assume messages $\lambda_{Y_j}(X)$ (resp. $\pi_X(U_i)$) reflect how X is influenced by the subgraph separated from it by Y_j (resp. U_i)[1]. More precisely, $\lambda_{Y_j}(X)$ (resp. $\pi_X(U_i)$) transmits

[1]By convention, we will denote by π's messages sent to children and by λ's messages sent to parents.

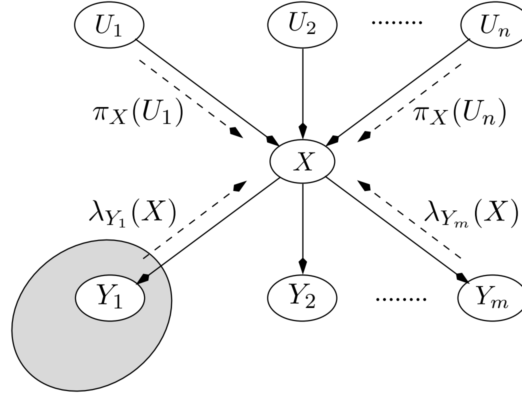


Figure 3.6: Messages during propagation.

to X all the informations coming from the lower part (resp. upper part) of the network that can be gathered at Y_j (resp. U_i). For instance, $\lambda_{Y_1}(X)$ gathers the influences on X of all the nodes in the shaded area of Figure 3.6. Knowing these influences, and assuming they are independent, the probability of X given all the evidence entered into the network can be computed using a formula which is simply a *function* of $P(X|Pa(X))$, the $\lambda_{Y_i}(X)$'s, and the $\pi_X(U_j)$'s:

$$P(X|e) = F(P(X|Pa(X)), \lambda_{Y_1}(X), \dots, \lambda_{Y_m}(X), \pi_X(U_1), \dots, \pi_X(U_n)).$$

Recall that $Pa(X)$ denotes the set of parents of X. Thus, the algorithm proposed by Pearl consists in sending π-λ messages to all the nodes in the network and then applying the above formula to compute the a posteriori probability of every node.

What are exactly the content of these $\pi - \lambda$ messages? Evidence may have been entered into several nodes of the network. So, let us call e^+_X (resp. e^-_X) the evidence connected to X by its parents U_i's (resp. children Y_j's), $e^+_{U_iX}$ (resp. $e^-_{XY_j}$) the evidence contained in the subnetwork on the tail side of arc (U_i, X) (resp. on the head side of arc (X, Y_j)). Peot and Shachter (see [PS91]) proved that using the following definitions for the $\pi_X(U_i)$-$\lambda_{Y_j}(X)$ messages:

$$\begin{aligned} \pi_X(U_i) &= P(U_i, e^+_{U_iX}), \\ \lambda_{Y_j}(X) &= P(e^-_{XY_j}|X), \end{aligned}$$

computing $P(X, e)$ can be achieved by using the following combination of the $\pi_X(U_i)$-$\lambda_{Y_j}(X)$'s and $P(X|Pa(X))$:

$$\begin{aligned} \pi(X) &= P(X, e^+_X) = \textstyle\sum_{U_1,\dots,U_n} P(X|U_1, \dots, U_n) \prod_{i=1}^n \pi_X(U_i); \\ \lambda(X) &= P(e^-_X|X) = \textstyle\prod_{j=1}^m \lambda_{Y_j}(X); \\ P(X, e) &= \lambda(X)\pi(X). \end{aligned}$$

In turn, once X has received the $\pi_X(U_i)$-$\lambda_{Y_j}(X)$ messages it needed, it sends to its

neighbors the following messages:

$$\begin{aligned} \pi_{Y_j}(X) &= \pi(X)\prod_{k\neq j}\lambda_{Y_k}(X); \\ \lambda_X(U_i) &= \sum_X\left[\lambda(X)\sum_{U_1,\ldots,U_n}P(X|U_1,\ldots,U_n)\prod_{k\neq i}\pi_{U_k}(X)\right]. \end{aligned} \tag{3.3}$$

At first sight the above messages seem quite complicated to produce and especially it is not very easy to see in which order they should be computed. However the following very simple scheme will i) ensure that all messages are sent appropriately and ii) make the computation of the messages very easy:

Algorithm 3.1 (Pearl's-like method)

1. *Select an arbitrary node, say X_i, as the current node in the Bayesian network. X_i is called the* root *of the algorithm.*
2. *inward pass: the current node asks its adjacent nodes for their λ or π messages depending on whether they are a child or a parent of the current node. In turn, they recursively ask their other adjacent nodes for π-λ 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 messages to its adjacent nodes; 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 the sum over all the variables except X_j (resp. X_k) of the product of $P(X_j|Pa(X_j))$ by all the messages sent to X_j except that sent by X_k.

Note that the prevision computations of Section 3.2 precisely corresponds to the application of the above algorithm with $X_i = G$. Similarly, the computations performed in Section 3.4 can be seen as an inward pass with $X_i = A$.

Example 3.1 Consider the Bayesian network of Figure 3.7 and assume that evidence e_A and e_G have been entered into nodes A and G. Let us apply Pearl's algorithm with $X_i = C$.

During the inward pass, C asks its adjacent nodes, i.e., A, B, D, E and F for their messages. As A has no other neighbor, it sends its message $\pi_C(A) = P(A)P(e_A|A) = P(A, e_A)$. Similarly, B sends $\pi_C(B) = P(B)$. F and D send messages $\lambda_F(C)$ and $\lambda_D(C)$ respectively. This is illustrated on Figure 3.8(a).

When E receives a query for a message, it cannot immediately send it as it has another neighbor: G. Hence E asks G for its message. G has no more neighbors, so it sends message $\lambda_G(E) = \sum_G P(G|E)P(e_G|G) = P(e_G|E)$ — see Figure 3.8(b). Now, E can sends to C message $\sum_E P(E|C)P(e_G|E) = \sum_E P(E|C)P(e_G|E,C) = \sum_E P(E,e_G|C) = P(e_G|C) = \lambda_E(C)$ — see Figure 3.8(c) — and the inward pass is completed as C has received messages from all of its neighbors.

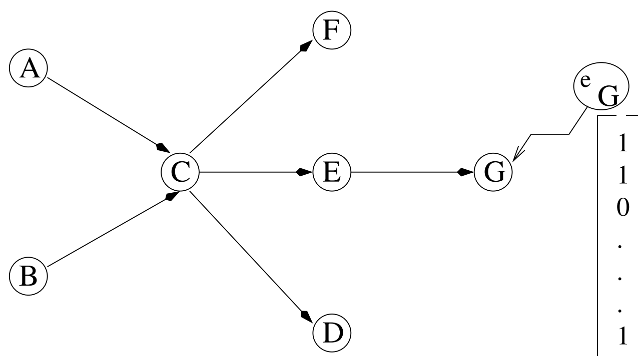


Figure 3.7: Example of Pearl's algorithm.

The beginning of the outward pass consists in C sending messages to A, B, D, E and F. The message sent to A is the sum over all the variables except A of the product of the conditional probability $P(C|A,B)$ by the messages sent to C by all its neighbors except A i.e.,

$$\begin{aligned}\lambda_C(A) &= \sum_{B,C} P(C|A,B)\pi_C(B)\lambda_D(C)\lambda_E(C)\lambda_F(C)\\ &= \sum_{B,C} P(C|A,B)P(B)\mathbb{1}_C P(e_G|C)\mathbb{1}_C\\ &= P(e_G|A).\end{aligned}$$

This is illustrated on Figure 3.8(d). Of course, the message sent to B — which is illustrated on Figure 3.8(e) — is similar, i.e,

$$\begin{aligned}\lambda_C(B) &= \sum_{A,C} P(C|A,B)\pi_C(A)\lambda_D(C)\lambda_E(C)\lambda_F(C)\\ &= \sum_{A,C} P(C|A,B)P(A,e_A)\mathbb{1}_C P(e_G|C)\mathbb{1}_C\\ &= P(e_A|B).\end{aligned}$$

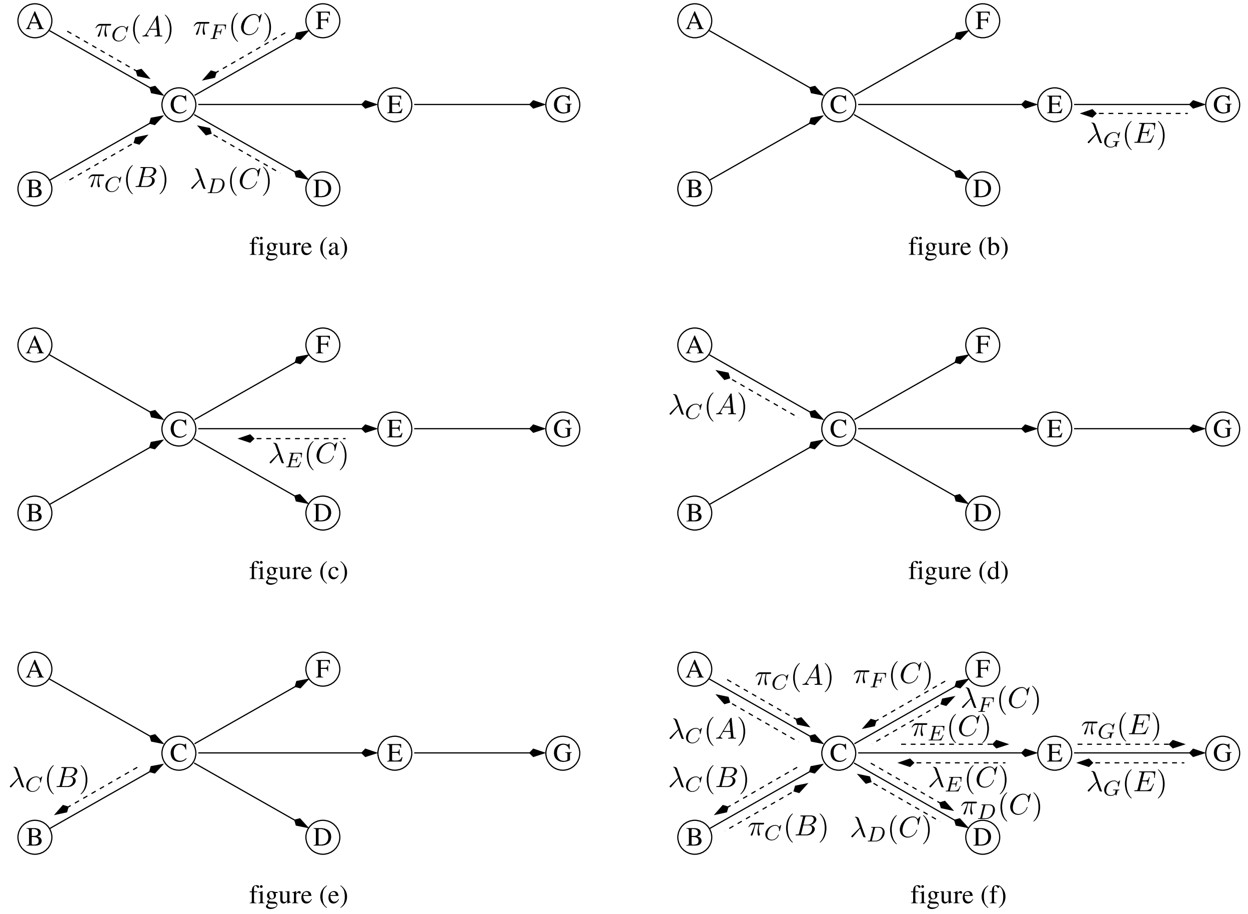


Figure 3.8: Messages in Pearl's algorithm.

the messages sent to D, F and E are respectively:

$$
\begin{aligned}
\pi_C(D) &= \sum_{A,B} P(C|A,B)\pi_C(A)\pi_C(B)\lambda_E(C)\lambda_F(C), \\
&= \sum_{A,B} P(C|A,B)P(A,e_A)P(B)P(e_G|C)\mathbb{1}_C = P(C,e_A,e_G), \\
\pi_C(E) &= \sum_{A,B} P(C|A,B)\pi_C(A)\pi_C(B)\lambda_D(C)\lambda_F(C), \\
&= \sum_{A,B} P(C|A,B)P(A,e_A)P(B)\mathbb{1}_C\mathbb{1}_C = P(C,e_A), \\
\pi_C(D) &= \sum_{A,B} P(C|A,B)\pi_C(A)\pi_C(B)\lambda_D(C)\lambda_E(C), \\
&= \sum_{A,B} P(C|A,B)P(A,e_A)P(B)\mathbb{1}_C P(e_G|C) = P(C,e_A,e_G).
\end{aligned}
$$

Finally, E sends the following message to G:

$$\pi_G(E) = \sum_C P(E|C)\pi_E(C) = \sum_E P(E|C)P(C, e_A) = P(E, e_A).$$

Figure 3.8(f) shows all the messages that are sent during both the inward and the outward phase. Note that, as was mentioned, on every arc (X, Y) the product of the π and and λ messages sent along this arc corresponds to $P(X, e)$, where e denotes all the evidence entered into the network — here $e = \{e_A, e_G\}$. ♦

As mentioned, the algorithm produces correct values for the *a posteriori* probabilities because the $e^-_{XY_j}$'s and the $e^+_{U_iX}$'s, i.e., all the evidence surrounding node X, are independent. This property always hold singly connected networks but is no more true in multiply connected networks. Hence such networks deserve a special treatment that shall be described in the next section.

3.6 Propagation in multiply connected networks

As we saw in Section 2.3 Bayesian networks are *Directed Acyclic Graphs*, that is, a Bayes net $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{P})$ may contain undirected cycles but not directed ones. A directed cycle is a sequence of arcs $\{(X_{i_j}, X_{i_{j+1}}) \in \mathcal{A},\ j \in \{1, \ldots, k\}\}$ such that $X_{i_1} = X_{i_{k+1}}$. Such directed cycles are forbidden for two reasons: i) because they imply a recurrent situation that leads to take several times the same evidence into account when computing *a posteriori* probabilities, thus preventing to obtain the correct results; and ii) because the Bayesian network simply reflects a simplification of the chain formula and, as the latter is obtained by applying recursively formula $P(A, B) = P(A|B)P(B)$ on every random variable, no variable A can belong to a directed cycle. However, undirected cycles (a.k.a. loops) are allowed because they just mean that a node X may influence several otherwise unrelated nodes that, in turn, have some influence on another node Y. These "chains" of influence are thus represented on the Bayesian network by different sequences of arcs (see Figure 3.9).

As we shall see in the following example, the propagation mechanism mentioned in the preceding section has a serious drawback: when applied on a multiply-connected network, i.e., a network containing undirected cycles, it will compute incorrect marginal probabilities. The reason is quite simple: the algorithm assumed that all the messages sent along the arcs of the network were independent, which does not hold when there are cycles.

Example 3.2 Let us consider the Bayesian network of Figure 3.9, that obviously contains the undirected cycle (loop) A, B, C, D. Assume that A is an arbitrary boolean random variable with probability 0.5 to be either 0 or 1, that B is a variable that always take the same value as A, and that C is always equal to $1 - A$. Finally, let D be the sum of B and C. Of course, as $B = A$ and $C = 1 - A$,

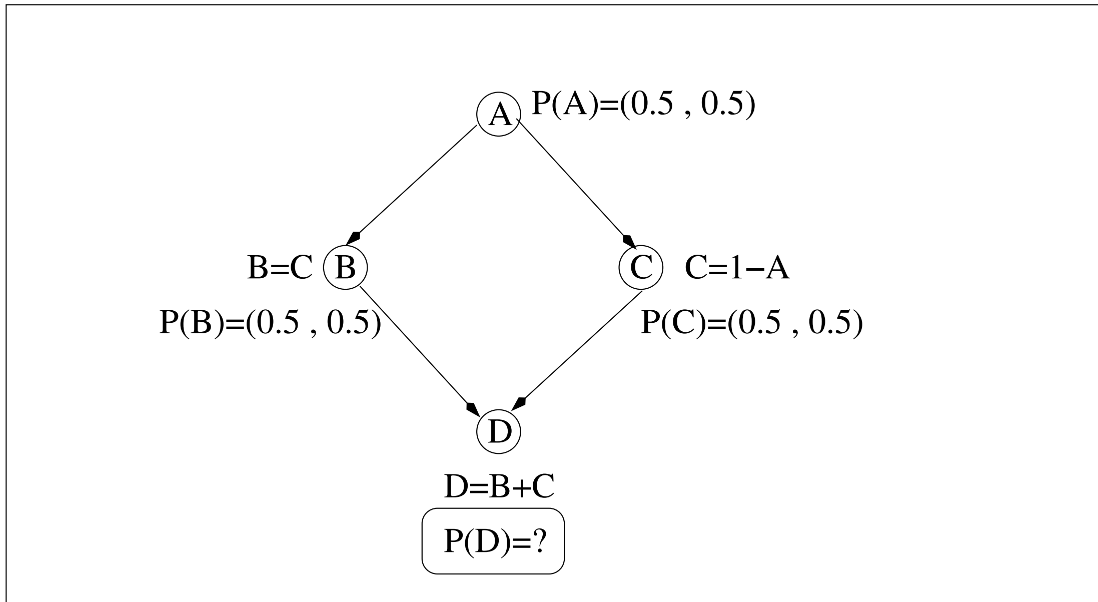


Figure 3.9: A graph with an undirected cycle.

$D = A + 1 - A = 1$. So, whatever the value of A, D should be equal to 1. Now, let us apply Pearl's algorithm for singly-connected networks and let us see that it will predict that D can take values 0,1 and 2.

Assume that the root of the algorithm is D, so that our computation of $P(D)$ will only require the inward pass. First, D asks its neighbors, B and C, for messages. In turn they ask A to send its message. Here, the algorithm should loop since B asking a message to A, the latter should ask C for a message before sending its own, and the algorithm will never terminate as every node would wait for another node to send a message before sending the one it was asked for. However, it is not unreasonable to think that a variant of Pearl's algorithm would be able to detect such infinite recurrence and that it would prevent A from asking for other messages before sending its own messages. Thus A would send to B and C a message containing $P(A)$, or equivalently a vector $(0.5, 0.5)$. Now, B can send its message:

$$P(B) = \sum_A P(B \mid A)P(A).$$

As $B = A$, probability $P(B \mid A)$ is equal to 1 when $B = A$ else 0. Thus

$$\begin{aligned} P(B = 0) &= \sum_A P(B = 0 \mid A)P(A) \\ &= P(B = 0|A = 0)P(A = 0) + P(B = 0|A = 1)P(A = 1) \\ &= 1 \times 0.5 + 0 \times 0.5 = 0.5. \end{aligned}$$

Similarly, $P(B = 1)$, so that $P(B) = (0.5, 0.5)$. In parallel, C can send to D the

following message:

$$\begin{aligned} P(C) &= \textstyle\sum_A P(C \mid A)P(A) \\ &= P(C|A=0)P(A=0) + P(C|A=1)P(A=1) = (0.5, 0.5). \end{aligned}$$

And D can thus compute its marginal *a priori* probability:

$$P(D) = \sum_{B,C} P(D \mid B, C)P(B)P(C).$$

But D is the sum of B and C, hence

$$P(D \mid B, C) = \begin{array}{c} \begin{array}{cccc} \multicolumn{2}{c}{B=0} & \multicolumn{2}{c}{B=1} \\ C=0 & C=1 & C=0 & C=1 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{l} D=0 \\ D=1 \\ D=2 \end{array}$$

Consequently, $P(D = 0) = 0.25$, $P(D = 1) = 0.5$ and $P(D = 2) = 0.25$, which is obviously not the correct result as we known that D equals 1 for sure. Why is there such a discrepancy between the correct result and the one obtained by Pearl? Well, this can be seen when expanding the expression $\sum_{B,C} P(D \mid B, C)P(B)P(C)$:

$$\begin{aligned} P(D) &= \textstyle\sum_{B,C} P(D \mid B, C) \; [P(B, A=0) + P(B, A=1)] \\ &\qquad\qquad\qquad\qquad [P(C, A=0) + P(C, A=1)] \\ &= \textstyle\sum_{B,C} P(D \mid B, C) \; [P(B, A=0)P(C, A=0)+ \\ &\qquad\qquad\qquad\qquad P(B, A=0)P(C, A=1)+ \\ &\qquad\qquad\qquad\qquad P(B, A=1)P(C, A=0)+ \\ &\qquad\qquad\qquad\qquad P(B, A=1)P(C, A=1)]. \end{aligned}$$

The expressions $P(B, A = 0)P(C, A = 1)$ and $P(B, A = 1)P(C, A = 0)$ are incoherent because they state that A takes at the same time two different values, 0 and 1. This explains why the value obtained for $P(D)$ is incorrect. The removal of the incoherent terms in the above equation, i.e., $P(B, A = 0)P(C, A = 1)$ and $P(B, A = 1)P(C, A = 0)$, would result in a correct value for $P(D)$. ♦

Fortunately, several modifications of the polytree algorithm have been proposed in the literature, that enable this algorithm to cope with loops. In the next section, we describe the modification advocated by Pearl [KP83], i.e., the cutset conditioning method, also called *Global conditioning*.

3.6.1 Global conditioning

Global conditioning consists in removing some arcs from the DAG, so that it contains no more loops and applying the polytree algorithm formulas with slight differences. Precisely, the dimension of messages are increased by that of the nodes at the tail of the removed arcs (forming the cutset).

Example 3.3 Consider the Bayesian network of Figure 3.10(a). As we saw, performing Pearl's algorithm within this graph will result in inconsistent results. However, assume that the value of A is known. For instance, say $A = a_1$. Then the arcs outgoing from A become useless for the propagation process. Indeed, consider arc (A, C): if A were to send a message to C, this message would be $P(A, e_A)$, where e_A = "A has the value a_1". But then arc (A, C) is useless, it is sufficient to substitute the conditional probability table stored in C by $P(C|A)P(A, e_A)$. Conversely, a message from C to A would be useless: it would simply contain the influence of C (and possibly some other nodes) on A. But there cannot be any influence since the value of A is already known for sure, so no new information is of any interest to A. Hence arc (A, C) is useless and can safely be dispensed with. Similarly, arc (A, B) could also be removed. However, the inference/propagation process is usually easier to perform if we keep the graph connected. Hence, here we can remove only arc (A, C) (but no information will be passed from B to A during the inference process) and perform the polytree algorithm.

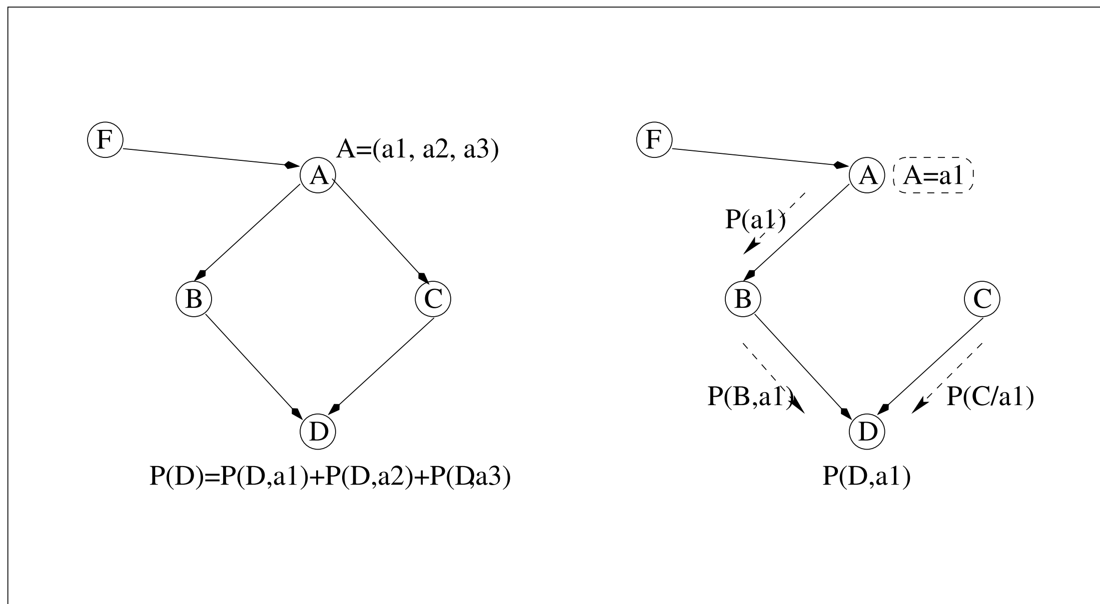


Figure 3.10: Graph with undirected cycle.

Of course, in general, there is not much chance that enough nodes will be instantiated for the graph to become singly-connected. However, if we observe that $P(D) = \sum_A P(D, A) = P(D, a_1) + P(D, a_2) + P(D, a_3)$, then it is sufficient to instantiate A successively to a_1, a_2 and a_3, to compute $P(D, a_1)$, $P(D, a_2)$, $P(D, a_3)$ and then to sum these values. All these operations can be performed in one step simply by imposing that the sizes (dimensions) of the $\lambda - \pi$ messages sizes be multiplied by $|A|$. Only at then end of the algorithm, when we multiply $\pi(X)$ by $\lambda(X)$, are we allowed to perform summations over A. ♦

As a conclusion, the global conditioning method (GC) requires the choice of a subset of arcs, the removal of which turns the graph into a polytree. Such a set always exists, but choosing the best set has been proved to be a NP-hard problem [SC90], and so heuristics have been proposed that produce good quality cutsets (see [BYGNR94], [BG96b]).

Property 3.1 *Global Conditioning conditions each node in the Bayesian network by the cutset, hence propagation is exponential in the cutset size.*

This property is of course undesirable, all the more that even computations performed in regions of the Bayesian network that do not belong to cycles are affected by the increase in the size of messages. For instance, in example 3.10, the messages transmitted along arc (F, A) would not be of size $|F|$ but of size $|F| \times |A|$. This seems quite inefficient since increasing the size of the messages is just a trick to cope with cycles, and (F, A) does not belong to any cycle. This led researchers to investigate other, more efficient, types of conditioning and *Local Conditioning* quite naturally arose.

3.6.2 Local conditioning

Peot and Shachter [PS91] introduced two important improvements to the polytree algorithm. They defined a *knot* as a portion of the network that cannot be made unconnected by removing only one edge. They proved that cutset conditioning need not be performed on the whole BN, but that it could be conducted separately on each knot. Thus, in their method, each knot is endowed with a cutset of its own, and an algorithm is introduced that computes probabilities using *knot*-cutsets.

Knot Conditioning (KC) is an improvement of global conditioning since cutset variables are used only in knots and not in the whole graph. Diez [Die96] later noticed that this local property could be further generalized by observing that even within knots some cutsets did not affect the whole knot but only part of it. This led to the concept of *Local Conditioning* (LC), which was later justified mathematically by [FJ00].

[FJ00]'s algorithm is similar in spirit to the global conditioning method, i.e., it amounts to apply algorithm 3.1 while keeping the dimensions related to some instantiated variables. In other words, summations on those instantiated variables are performed only at the end of the algorithm, when computing marginal probabilities as the product of downward and upward messages. To understand what are precisely the messages dimensions and which summations shall be performed both in singly and multiply connected networks, we will label arcs with the dimension of their corresponding messages. First, remark that when the graph contains no cycle, for instance the graph of Figure 3.11, $\pi_Y(X)$ and $\lambda_Y(X)$ have only one dimension, that of X, since they are equal to $P(X, e^+_{XY})$ and $P(e^-_{XY}|X)$ respectively, where e^+_{XY} (resp. e^-_{XY}) stands for the evidence available on X's (resp. Y's) side of arc (X, Y). In a singly-connected network, labels are thus similar to those of Figure 3.12(a).

In other words, if C_{XY} denotes the label of arc (X, Y), then Peot and Shachter's formula (Equation 3.3) can be expressed as follows:

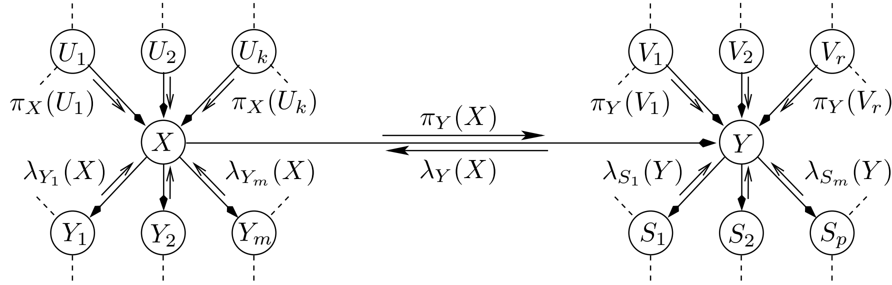


Figure 3.11: An arc (X, Y) and its neighbors.

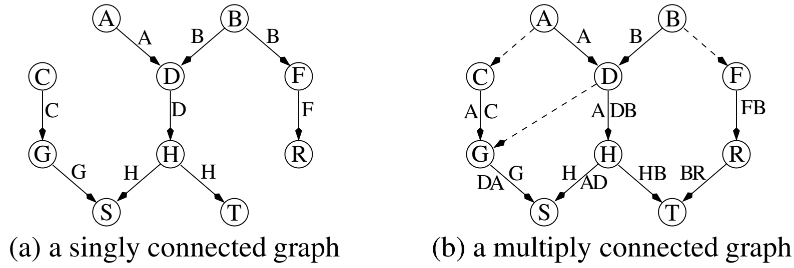


Figure 3.12: Labeling arcs in the network.

Definition 3.1 ($\pi - \lambda$ messages in terms of labels)

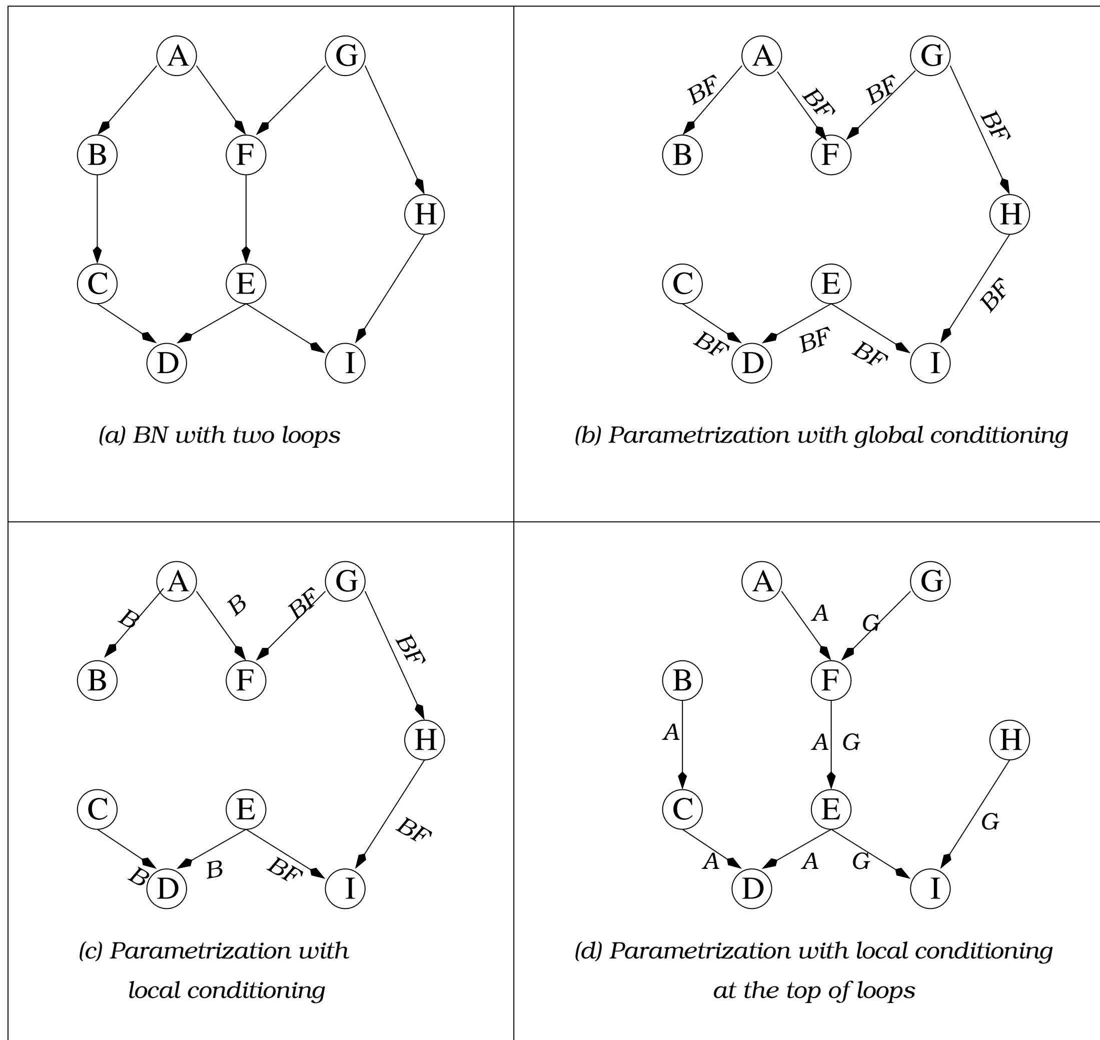
- $$\pi_Y(X) = \sum_{Z \in \mathcal{V} \setminus C_{XY}} \left[P(X, e_X | \mathrm{Pa}(X)) \times \prod_{i=1}^{k} \pi_X(U_i) \times \prod_{j=1}^{m} \lambda_{Y_j}(X) \right];$$
- $$\lambda_Y(X) = \sum_{Z \in \mathcal{V} \setminus C_{XY}} \left[P(Y, e_Y | \mathrm{Pa}(Y)) \times \prod_{i=1}^{r} \pi_Y(V_i) \times \prod_{j=1}^{p} \lambda_{S_j}(Y) \right].$$

In the case of multiply connected networks, as we saw in the preceding section, enough cutset nodes must be selected to remove all cycles. For instance, on Figure 3.12(b), we chose arbitrarily nodes A, B and D. Faÿ and Jaffray [FJ00] showed that performing algorithm 3.1 (with labeled messages) using the labels provided by the algorithm below (leading to the labels of Figure 3.12(b)) results in correct a posteriori probabilities.

Algorithm 3.2 (labeling algorithm) *Let $(\mathcal{V}, \mathcal{A}, \mathcal{P})$ be a BN. Assign to each arc (X, Y) of $\mathcal{A}$ label X. Then, while there remain cycles, select one of them and cut one of its arcs, call it (A, B), and add to the labels of the other arcs of the cycle arc (A, B)'s label.*

Local Conditioning decreases messages dimensions of Global conditioning messages by avoiding unnecessary parameters. For instance, consider the graphs of Figure 3.13. We can notice that the labels induced by Global Conditioning (Figure 3.13(b))

are supersets of those induced by Local Conditioning. Hence the latter will perform computations faster than the former. Note that in both cases, we began by cutting the loop $ABCDEF$ at arc (B, C), then we cut loop $EFGHI$ at arc (F, E).



(a) BN with two loops

(b) Parametrization with global conditioning

(c) Parametrization with local conditioning

(d) Parametrization with local conditioning at the top of loops

Figure 3.13: Resulting polytrees from a given BN.

Chapter 4

Undirected methods

In Chapter 2, it was shown that, as is, undirected graphs are not expressive enough to represent faithfully probabilistic independencies since the same graph could represent several incompatible sets of dependencies/independencies. Remember for instance the two examples we used in Chapter 2:

1. people's reading ability (variable $Ability$) and their shoe sizes (variable $Shoes$) are dependent random variables. However, conditionally to variable Age, they become independent.

2. D_1 and D_2 are two dice. We throw them and denote by Sum the sum of their values. The dice are not loaded and are independent. However, knowing the value of Sum, they become dependent since knowing the value of one die results in knowing the value of the other die.

The very basic idea consisting in adding edges between dependent variables led us to the undirected graphs of Figure 4.1, which was not satisfying since the graphs were identical and yet represented incompatible sets of dependencies/independencies. This led us to add orientations to edges so as to be able to discriminate between these different sets. However, this is not the only way such discrimination can be achieved and, in this chapter, we shall see that modifying slightly the undirected graphs of Figure 4.1 also enable the aforementioned discrimination.

Consider again the problem of the dice. As we saw Figure 4.2(a) could not be used because it already represented the set of dependencies/independencies of the age/ability/shoe problem. However, adding a new edge between die D_1 and die D_2 produces a graph that is not conflicting with that of age/ability/shoe and we will see in this chapter how this can be generalized to apply to any set of dependencies/independencies.

To do so, we will first review some basic Markov properties. Then these properties will be exploited to produce join trees and junction trees. Finally it will be shown how these trees can be used to perform evidence propagation.

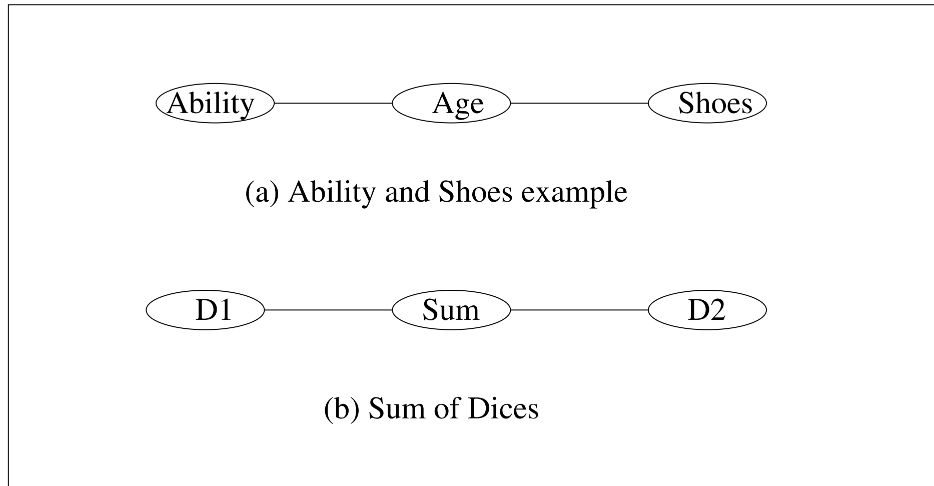


Figure 4.1: Some graphs representing dependencies/independencies between random variables.

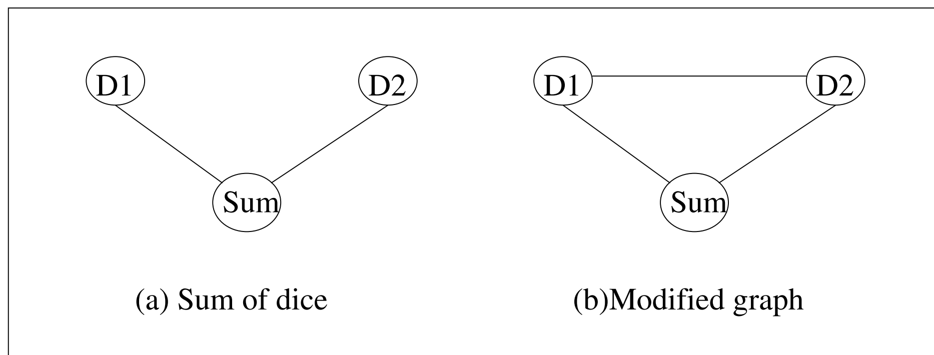


Figure 4.2: Modified undirected graph.

4.1 Markov Networks

Before explaining how the construction of graphs such as those of Figure 4.2(b) can be done, one must wonder why the simple graph of Figure 4.2(a) could not faithfully represent the set of dependencies/independencies of the dice example. When we designed it in chapter 2, we tried to represent independence between pairs of variables by a lack of edge between these variables and, conversely, dependence between pairs of variables by edges. However, such method does not take full advantage of the expressive power of graphical representations in that the structure of the graph has no special significance except that connected pairs are dependent whereas unconnected ones are independent. Yet this structure should offer a distinction between direct and indirect connections: for instance, in Figure 4.1(a), the graphical representation should tell us intuitively that the reading's ability is found to be related to shoe size because both variables depend on the age of the person but it should also tell us that, when the age

is known, there is no relation between ability and shoes. This suggests that the graphical structure should represent conditional independencies, that is, the lack of an edge between two nodes/variables should indicate an independence between these variables *conditionally* to the variables that are in between (here the age). Such graphs will be called *Markov networks* and this section is devoted to their study.

As we are about to represent sets of independencies by graphs, we should first give a few words about probabilistic independencies. Let $\mathcal{V}$ be a set of random variables and let X, Y, Z be a partition of $\mathcal{V}$. Let us denote the independence of sets X and Y conditionally to set Z by $I(X, Z, Y)_P$. Then $I(\cdot, \cdot, \cdot)_P$ satisfies the following well-known properties [Lau82]:

Property 4.1 (Conditional independence)

$$I(X, Y, Z)_P \iff P(X, Y|Z) = P(X|Z)P(Y|Z) \tag{4.1}$$

$$I(X, Y, Z)_P \iff \exists\, f, g : \; P(X, Y, Z) = f(X, Z)g(Y, Z) \tag{4.2}$$

$$I(X, Y, Z)_P \iff P(X, Y, Z) = P(X|Z)P(Y, Z) \tag{4.3}$$

As mentioned above, we would like to represent sets of conditional independencies by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that if Z is a set of nodes between sets X and Y — actually if all paths between any node in X and any node in Y pass through a node in Z — then it means that $I(X, Z, Y)_P$. Let us denote the fact that Z is "in-between" X and Y by $\langle X|Z|Y\rangle_{\mathcal{G}}$. Then, our aim is to create a graph $\mathcal{G}$ such that:

$$I(X, Y, Z)_P \iff \langle X|Z|Y\rangle_{\mathcal{G}}. \tag{4.4}$$

Unfortunately, a result by Pearl and Paz [PP85] shows that no graphical structure can satisfy the above equation but there do exit graphical structures satisfying:

$$I(X, Y, Z)_P \Longleftarrow \langle X|Z|Y\rangle_{\mathcal{G}}. \tag{4.5}$$

These are called *I-maps* [Pea88]. Of course, Equation (4.5) is less satisfying than Equation (4.4) since it states that if set Z intercepts all paths between the nodes in X and the nodes in Y, then X and Y are independent conditionally to Z, but it does not guarantee that if there exist some paths between nodes in X and nodes in Y that are not intercepted by Z then X and Y are dependent conditionally to Z whereas Equation (4.4) offers this guarantee. As a consequence, *I*-maps guarantee that nodes separated by some set Z are indeed independent conditionally to Z but there may exist connected nodes that are truly independent.

As *I*-maps do not encode all independencies, a given set of conditional independencies $I(X, Z, Y)_P$'s may be representable by different *I*-maps. For instance, the complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, i.e., the graph in which for every pair of node in $\mathcal{V}$ there exists an edge in $\mathcal{E}$ linking these nodes — is an *I*-map for any set of conditional independencies since it states that every pair of nodes *may* be dependent. Hence it is legitimate to wonder if there exists for any set of probabilistic conditional independencies a minimal *I*-map (in the sense that removing any edge from the graph would make it cease

to be an *I*-map). The answer was given by Pearl and Paz [PP85] for strictly positive probabilities:

Theorem 4.1 *Any strictly positive distribution P has a unique minimal I-map $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ produced by connecting only those pairs of nodes X,Y in $\mathcal{V}$ for which $I(X, \mathcal{V}\backslash\{X, Y\}, Y)_P$ is false, i.e.,*

$$(X, Y) \notin \mathcal{E} \Longleftrightarrow I(X, \mathcal{V}\backslash\{X, Y\}, Y)_P.$$

Using this theorem, we can now create the minimal *I*-map for the dice problem: we know that not($I(D_1, Sum, D_2)_P$) since knowing the value of the sum of the dice gives us the value of D_2 when we know the value of D_1. Consequently, edge (D_1, D_2) should exist. Naturally, not($I(D_1, D_2, Sum)_P$) and not($I(D_2, D_1, Sum)_P$) also hold since, when the value x of one die is known, Sum is equal to x plus the value of the other die. Consequently, our minimal *I*-map should contain edges (D_1, Sum) and (D_2, Sum). This is precisely what we obtained on Figure 4.2(b).

Naturally, as for Bayesian networks, once the structure of the graph has been established, there remains to quantify the strengths of the dependencies represented by the edges. The solution to this problem is provided by the theory of Markov fields [Ish81, Lau82] and the following algorithm:

Algorithm 4.1 *Let $\mathcal{V} = \{X_1, \dots, X_n\}$ be a set of random variables.*

1. *Identify the cliques of the I-map $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, i.e., the largest subgraphs whose nodes are all adjacent to each other.*

2. *For each clique C_i, assign a nonnegative compatibility function $g_i(c_i)$, which measures the relative degree of compatibility associated with the value assignment c_i to the variables included in C_i.*

3. *Form the product $\prod_i g_i(c_i)$ of the compatibility functions over all the cliques.*

4. *Normalize the product over all possible value combinations of the variables in the graph. The normalized product constitutes a joint probability distribution of $\mathcal{V}$ that satisfies all the conditional independencies in the undirected graph:*

$$P(\mathcal{V}) = \frac{\displaystyle\prod_i g_i(c_i)}{\displaystyle\sum_{x_1,\dots,x_n} \prod_i g_i(c_i)}.$$

The above algorithm provides a safe way to retrieve a probability distribution that is compatible with the *I*-map $\mathcal{G}$, i.e., this distribution encodes a compatible set of conditional independencies. However, it is not very satisfying because the interpretation of the $g_i(C_i)$'s is not always straightforward. Indeed, consider the following example:

Example 4.1 Let A, B, C, D be four people. A is friend with B and C, B is friend

with A and D, C is friend with A and D as well, and D is only friend with B and C, so that the relations between these four individuals can be represented by the graph of Figure 4.3.

A contagious disease appeared in the region where the four individuals live and we may wonder what is the probability that a given person will catch this disease. Assume that there is a joint probability distribution P over A, B, C, D of catching the disease and that this distribution is equal to:

$$P(A,B,C,D) = \frac{g_1(A,B)g_2(A,C)g_3(B,D)g_4(C,D)}{\sum_{A,B,C,D} g_1(A,B)g_2(A,C)g_3(B,D)g_4(C,D)}, \tag{4.6}$$

which is compatible with the *I*-map of Figure 4.3. Now, extracting functions $g_i(\cdot)$'s from a database or from an expert is not a trivial task. Indeed, $g_1(A,B)$ should reflect the risk of the pair A, B catching the disease, but it is not easy to isolate this risk from the risk involved by the rest of the population as A often sees C, and B often sees D (see Figure 4.3). Hence an expert would find it difficult to estimate functions $g_i(\cdot)$'s. Even computing these functions from a database would be difficult as Equation (4.6) is a nonlinear equation. ♦

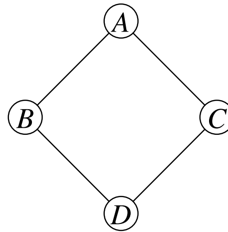


Figure 4.3: The relations between A, B, C and D.

However not all sets of independencies/Markov networks suffer from this quantification problem: as described in the theorem below, when Markov networks are chordal, the $g_i(\cdot)$'s can be interpreted as probabilities, which usually simplifies their elicitation.

Definition 4.1 (chordality) *An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is said to be chordal if every cycle of length four or more has a chord, i.e., an edge linking two nonadjacent nodes in the cycle.*

Definition 4.2 (decomposability) *Let P be a probability distribution. P is said to be decomposable if it has a minimal I-map that is chordal. P is said to be decomposable relative to an undirected graph $\mathcal{G}$ if $\mathcal{G}$ is an I-map of P and $\mathcal{G}$ is chordal.*

Theorem 4.2 *Let P be a probability distribution decomposable relative to a graph $\mathcal{G}$ then the joint distribution of P can be written as a product of the cliques of $\mathcal{G}$ divided by a product of the distributions of their intersections.*

Example 4.2 Consider the chordal graph of Figure 4.4. The cliques of this graph are: AB, BCD, CDE, EF and EG. Here, the intersections between cliques are B (this is the intersection between AB and BCD), CD (intersection of BCD and CDE) and E (intersection between CDE, EF and EG). Hence Theorem 4.2 guarantees that any probability distribution over A, B, C, D, E, F, G compatible with this graph can be decomposed as:

$$P(A, B, C, D, E, F, G) = \frac{P(A, B)P(B, C, D)P(C, D, E)P(E, F)P(E, G)}{P(B)P(C, D)P(E)}.$$

Algorithm 4.1 stated that there exist functions $g_i(\cdot)$'s such that:

$$P(A, B, C, D, E, F, G) \propto g_1(A, B)g_2(B, C, D)g_3(C, D, E)g_4(E, F)g_5(E, G).$$

By assigning:

$$\begin{array}{rlrl} g_1(A, B) = P(A, B) & & g_2(B, C, D) = P(C, D|B) \\ g_3(C, D, E) = P(E|C, D) & & g_4(E, F) = P(E, F, G), \end{array}$$

it can easily be seen that both expressions of P are similar. ♦

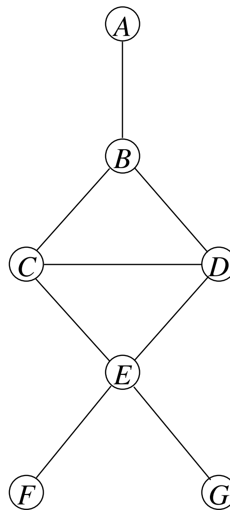


Figure 4.4: A chordal *I*-map.

4.2 From Markov networks to join trees and junction trees

Naturally, not all probability distributions are decomposable, but we shall see in this section how, dropping some independencies, that is, occulting the knowledge that they

do exist, we can transform any probability into a decomposable one. This transformation will be performed by triangulating an *I*-map of the distribution. However, although *I*-maps encode many independencies underlying the joint probability distribution, they are not very well suited for evidence propagation because these independencies are not readily available from these graphs but need be extracted from them (for instance finding the cliques of the network). Hence, it is often a good idea to transform these undirected graphs into secondary structures more suited for evidence propagation. In Shafer-Shenoy's algorithm [Sha96] the secondary structure is called a **join tree** and in Jensen's scheme it is called a **junction tree**. Both structures result from a triangulation process: an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is said to be triangulated if and only if it is chordal. As shown in [Ros70], triangulating a graph amounts to apply the following algorithm:

Algorithm 4.2 (Triangulation)
Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V} = \{X_1, \dots, X_n\}$. Denote by $\mathrm{adj}(X_i)$ the set of vertices adjacent to X_i. A vertex $X_i \in \mathcal{V}$ is said to be eliminated from graph $\mathcal{G}$ when:

1. *the edges $(\mathrm{adj}(X_i) \times \mathrm{adj}(X_i)) \backslash \mathcal{E}$ are added to $\mathcal{E}$ such that $\mathrm{adj}(X_i) \cup \{X_i\}$ becomes a clique;*
2. *the edges between X_i and its neighbors are removed from $\mathcal{E}$, as well as X_i from $\mathcal{V}$.*

Let σ be a permutation of $\{1, \dots, n\}$. Let us eliminate $X_{\sigma(1)}, X_{\sigma(2)}, \dots, X_{\sigma(n)}$ successively and call $\mathcal{E}_T$ the set of edges added to graph $\mathcal{G}$ by these eliminations. Then graph $\mathcal{G}_T = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_T)$ is chordal.

Thus triangulating a graph simply amounts to applying an elimination sequence. As an illustration, applying sequence S, C, G, R, T, B, D, H, F, A on the undirected graph of Figure 4.5(a) results in the triangulated graph of Figure 4.5(b). Note that applying elimination sequence σ to a graph already triangulated by σ keeps the graph unaffected. Of course, different sequences induce different triangulated graphs. The elimination sequence shall thus be carefully chosen. Finding the best one is unfortunately NP-hard [Yan81], but heuristics exist that usually provide good sequences [Kjæ90, BG96b, BG96a, SG97].

Note that Figure 4.5(a) simply states that the joint probability distribution over random variables $A, B, C, D, F, G, H, R, S, T$ can be decomposed as:

$$\begin{aligned} P(A,B,C,D,F,G,H,R,S,T) = {} & f_1(A,B,D) f_2(A,C) f_3(B,F) f_4(C,G) \\ & f_5(D,H) f_6(F,R) f_7(G,H,S) f_8(H,R,T), \end{aligned} \tag{4.7}$$

whereas Figure 4.5(b) states that $P(A, B, C, D, F, G, H, R, S, T)$ is decomposable as:

$$\begin{aligned} P(A,B,C,D,F,G,H,R,S,T) = {} & g_1(A,B,D,F) g_2(A,C,G) g_3(G,H,S) \\ & g_4(A,G,H) g_5(H,F,R,T) g_6(A,D,F,H). \end{aligned}$$

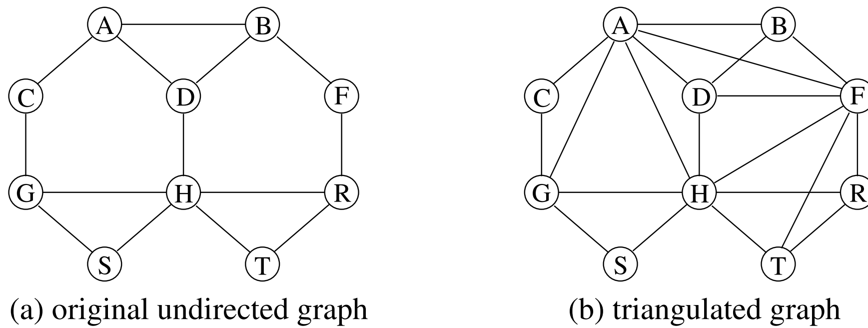


Figure 4.5: The moralization and triangulation steps.

Note that the latter decomposition is compatible with that of Equation (4.7). Indeed, if:

- $g_1(A, B, D, F) = f_1(A, B, D)f_3(B, F)$,
- $g_2(A, C, G) = f_2(A, C)f_4(C, G)$,
- $g_3(G, H, S) = f_7(G, H, S)$,
- $g_4(A, G, H) = 1$,
- $g_5(H, F, R, T) = f_6(F, R)f_8(H, R, T)$,
- $g_6(A, D, F, H) = f_5(D, H)$,

then the latter decomposition of $P(\cdot)$ equals that of Equation (4.7).

From chordal graphs we can extract elimination trees and junction trees that are very convenient for evidence propagation:

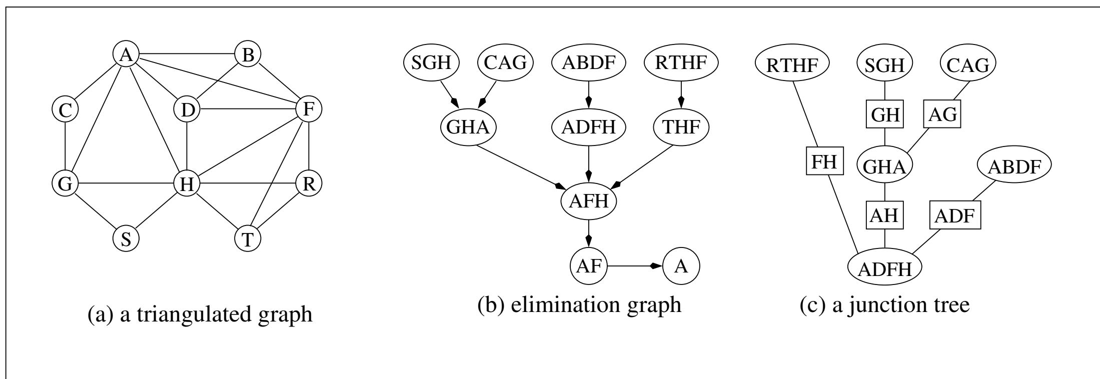


Figure 4.6: Elimination and junction trees.

Definition 4.3 (Elimination tree) *Let $\mathcal{G}_T = (\mathcal{V}, \mathcal{E}_T)$ be a chordal graph produced by an elimination sequence σ. Let $D_{\sigma(i)} = adj(X_{\sigma(i)}) \cup \{X_{\sigma(i)}\}$ be the clique adjacent to $X_{\sigma(i)}$ in $\mathcal{G}_T$ just before $X_{\sigma(i)}$ is eliminated, and let $\mathcal{D} = \{D_{\sigma(1)}, \ldots, D_{\sigma(n)}\}$. The elimination tree of $\mathcal{G}_T$ is graph $\mathcal{G}_E = (\mathcal{D}, \mathcal{A}_E)$, where $\mathcal{A}_E$ is the set of arcs $(D_{\sigma(i)}, D_{\sigma(j)})$, $i < j$, such that $j = \min\{k : X_{\sigma(k)} \in D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}\}$. Each arc $(D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}_E$, $i < j$, which is called the separator arc of node $D_{\sigma(i)}$, is labeled by $D_{\sigma(i)} \cap D_{\sigma(j)}$, which is called its separator content.*

Definition 4.4 (Junction tree) *Let $\mathcal{C}$ be the set of cliques of a triangulated graph $\mathcal{G}_T$. A junction tree of $\mathcal{G}_T$ is a tree $\mathcal{G}_J = (\mathcal{C}, \mathcal{E}_J)$ having the running intersection property, i.e., such that for any pair $C_1, C_2 \in \mathcal{C}$, $C_1 \cap C_2 \subseteq C_i$ for all the nodes $C_i \in \mathcal{C}$ on the unique path in $\mathcal{G}_J$ between C_1 and C_2. For any two adjacent nodes $C_1, C_2 \in \mathcal{C}$, $C_1 \cap C_2$ is called the separator of nodes C_1 and C_2.*

Elimination trees are similar in spirit to junction trees except that some of their vertices may be nonmaximal complete subgraphs of $\mathcal{G}_T$. Note that for a given triangulated graph, there exists only one elimination tree whereas there may exist different junction trees. However, as shown in [JJ94], all these junction trees contain precisely the same cliques and separators. As an illustration, Figure 4.6 represents the elimination tree and one possible junction tree for the triangulated graph of Figure 4.5(c).

The way we derived junction trees, one of their key aspects of is that they represent a decomposition of a joint probability. Indeed, it can be shown [CDLS99, chapter 5] that if $\mathcal{G}_J = (\mathcal{C}, \mathcal{E}_J)$ and $\mathcal{G}_E = (\mathcal{D}, \mathcal{A}_E)$ are respectively a junction tree and the elimination tree of a triangulated graph $\mathcal{G}_T = (\mathcal{V}, \mathcal{E})$, then the joint probability on $\mathcal{V}$ is equal to the product of the joint probabilities of the cliques divided by those of the separators, i.e.,

$$P(\mathcal{V}) = \frac{\prod_{C_i \in \mathcal{C}} P(C_i)}{\prod_{(C_i, C_j) \in \mathcal{E}_J} P(C_i \cap C_j)} = \frac{\prod_{D_i \in \mathcal{D}} P(D_i)}{\prod_{(D_i, D_j) \in \mathcal{A}_E} P(D_i \cap D_j)}.$$

As the decomposition of the joint probability distribution is strongly related to both Bayesian networks and junction trees, it is legitimate to wonder whether the latter can be derived from Bayesian networks (and conversely). The answer consists in applying three steps:

1. the Bayesian network is moralized, hence resulting in a new graph called the moral graph — actually, this is an *I*-map;

2. the moral graph is triangulated;

3. the triangulated graph is transformed into a junction tree.

Algorithm 4.3 (Moralization) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a directed graph. The moral graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E})$ results from the the application of the following two steps:*

1. *for any vertex X in $\mathcal{V}$, add (undirected) edges between all pairs of parents of X in $\mathcal{A}$;*
2. *replace all remaining arcs (X, Y)'s by (undirected) edges (X, Y)'s.*

Moralizing a Bayesian network naturally transforms it into an *I*-map. For instance, the graph of Figure 4.7(a) decomposes the joint probability distribution as:

$$P(\mathcal{V}) = P(T|H,R)P(S|G,H)P(R|F)P(H|D)P(G|C) \\ P(F|B)P(D|A,B)P(C|A)P(B)P(A),$$

By assigning:

$$\begin{aligned} a(T,H,R) &= P(T|H,R) & b(S,G,H) &= P(S|G,H) \\ c(R,F) &= P(R|F) & d(H,D) &= P(H|D) \\ e(G,C) &= P(G|C) & f(F,B) &= P(F|B) \\ g(A,B,D) &= P(D|A,B)P(B)P(A) & h(C,A) &= P(C|A), \end{aligned}$$

$$P(\mathcal{V}) = a(T,H,R)b(S,G,H)c(R,F)d(H,D)e(G,C) \\ f(F,B)g(D,A,B)h(C,A),$$

which corresponds precisely to the decomposition represented by the moral graph of Figure 4.7(b).

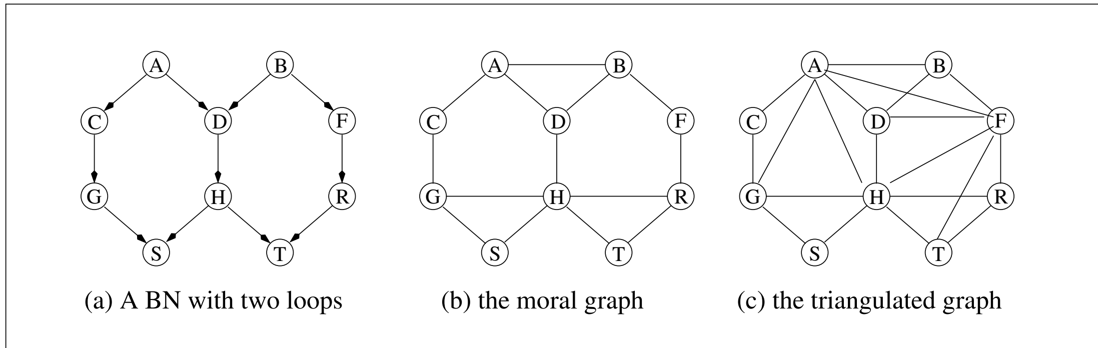


Figure 4.7: The moralization and triangulation steps.

Of course, steps 2 and 3 of the BN-junction tree conversion are just the transformation of the *I*-map into an *I*-map of a decomposable probability.

Now we shall see in the next sections how these graphs can be exploited to perform evidence propagations.

4.3 Jensen's inference method

Assume now that we have constructed a junction tree representing a probability distribution P. As we saw in the preceding section, the joint distribution P is equal to the product of the joint probabilities of the cliques divided by those of the separators. Hence assume that we populate cliques and separators with their corresponding joint probabilities hereafter called *potentials*. Here we will denote by ϕ (resp. ψ) the potentials of the separators (resp. of the cliques).

To understand Jensen's inference engine, consider the part of a junction tree of Figure 4.8. Assume that the potentials in C_i, C_j and S are:

$$\begin{aligned}\psi(C_i) &= P(C_i),\\ \psi(C_j) &= P(C_j),\\ \phi(S) &= P(S)\end{aligned}$$

respectively. Since S is a separator, $S = C_i \cap C_j$ and consequently:

$$\phi(S) = \sum_{C_i \setminus S} P(C_i) = \sum_{C_j \setminus S} P(C_j).$$

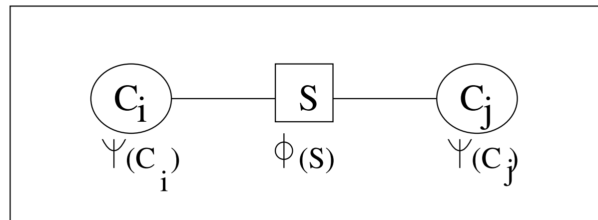


Figure 4.8: Jensen's (HUGIN) propagation algorithm.

The junction tree can be said to be in equilibrium in the sense that the potentials in cliques and separators are consistent. Now, new evidence can be entered into the network. This can be a piece of information e_X indicating that a variable X, the possible values of which were a priori $x_1, \dots, x_k$, cannot have value x_1 anymore, or the observation that Y has only one possible value left: y. In any case, as for Chapter 3, we will always assume the following holds:

Hypothesis 4.1 *Any evidence e_X related to a random variable X is independent of the other variables of the network conditionally to X.*

Assume now that taking into account a new evidence on a variable in clique C_i leads to change $\psi(C_i)$ into $\psi^*(C_i)$. The equilibrium of the junction tree is lost since:

$$\phi(S) \neq \sum_{C_i \setminus S} \psi^*(C_i).$$

To restore it, it is sufficient to apply the following two steps, that are called *absorption of C_j from C_i*:

Algorithm 4.4 (absorption of C_j from C_i)

1. *potential $\phi(S)$ associated to S is replaced by* $\phi^*(S) = \sum_{C_i \setminus S} \psi^*(C_i)$;

2. *potential $\psi(C_j)$ associated to C_j is replaced by* $\psi^*(C_j) = \psi(C_j) \times \dfrac{\phi^*(S)}{\phi(S)}$.

Jensen's inference algorithm consists in applying in a systematic way the above absorption process. [Jen96] proves that the equilibrium can be restored in the whole junction tree by selecting arbitrarily a clique, say C, and applying successively on this clique the two functions *Collect-Evidence* and *Distribute-Evidence* below. Once the junction tree is again in equilibrium, a posteriori marginal probabilities of each random variable X_i can be computed by selecting any clique C containing X_i and computing:

$$\sum_{C \setminus \{X_i\}} \psi^*(C)$$

or any separator S containing X_i and computing:

$$\sum_{S \setminus \{X_i\}} \phi^*(S).$$

Algorithm 4.5 (Collect-Evidence on clique C_i) *For all cliques C_j adjacent to C_i except, if any, that which called* Collect-Evidence *on C_i, do:*

1. *call* Collect-Evidence *on C_j,*

2. *perform the absorption of C_i from C_j.*

Algorithm 4.6 (Distribute-Evidence on clique C_i) *For all cliques C_j adjacent to C_i except, if any, that which called* Distribute-Evidence *on C_i, do:*

1. *perform the absorption of C_j from C_i,*

2. *call* Distribute-Evidence *on C_j.*

Note that the complexity of the absorption process corresponds to the size of the cliques, so that the complexity of the overall inference algorithm is the sum of the sizes of all the cliques of the junction tree.

To complete this section, there remains to describe how junction trees are initialized: indeed until now we have always supposed that the potentials of the junction trees

used for evidence propagation initially contain joint probabilities. However, in practical situations, these are not available upon the start as all that is known at the beginning is a Bayesian network, and hence only a set of conditional probabilities is available. Fortunately [Jen96] shows that application of the following algorithm results in all the potentials being equal to joint probabilities:

Algorithm 4.7 (Junction tree initialization)

1. *Fill all potentials with 1's.*
2. *For each conditional probability in the Bayesian network, find a clique containing all the variables of this probability and multiply the potential of this clique by this conditional probability.*
3. *Select a clique arbitrarily, say C, and apply Collect-Evidence on C and then Distribute-Evidence on C.*

Example 4.3 Consider the Bayesian network of Figure 4.9(a). The result of the moralization and triangulation processes (using elimination sequence H, G, F, E, D, C, B, A) is the Markov network of Figure 4.9(b). Finally, the junction tree of Figure 4.9(c) is constructed from the Markov network. After performing step 1 of Algorithm 4.7, we may have the following potentials:

$$\begin{array}{ll}
\psi(ABC) = P(B|A)P(C|A)P(A) = P(A,B,C) & \phi(BC) = \mathbb{1}_{BC} \\
\psi(BCD) = P_B(D|C) & \phi(BD) = \mathbb{1}_{BD} \\
\psi(BDE) = P_D(E|B) & \phi(DE) = \mathbb{1}_{DE} \\
\psi(DEF) = P_E(F|D) & \phi(EF) = \mathbb{1}_{EF} \\
\psi(EFG) = P_F(G|E) & \phi(FG) = \mathbb{1}_{FG} \\
\psi(FGH) = P(H|F,G), &
\end{array}$$

where, for any set X of random variables, $\mathbb{1}_X$ denotes a vector/matrix of size $|X|$ filled only with 1's and where $P_X(Y|Z)$ stands for the matrix of size $|X| \times |Y| \times |Z|$ filled with $|X|$ replicas of conditional probability $P(Y|Z)$.
Let $\psi^*(ABC) = \psi(ABC) = P(A,B,C)$ and $\psi^*(FGH) = \psi(FGH) = P(H|F,G)$. Applying Collect-Evidence on clique BDE results in the following computations:

- separator BC: $\phi^*(BC) = \sum_A \psi^*(ABC) = \sum_A P(A,B,C) = P(B,C)$;
- clique BCD: $\psi^*(BCD) = \psi(BCD) \times \frac{\phi^*(BC)}{\phi(BC)} = P_B(D|C) \times \frac{P(B,C)}{\mathbb{1}_{BC}} = P(B,C,D)$;
- separator BD: $\phi^*(BD) = \sum_C \psi^*(BCD) = \sum_C P(B,C,D) = P(B,D)$;
- clique BDE: $\psi'(BDE) = \psi(BDE) \times \frac{\phi^*(BD)}{\phi(BD)} = P_D(E|B) \times \frac{P(B,D)}{\mathbb{1}_{BD}} = P(B,D,E)$;

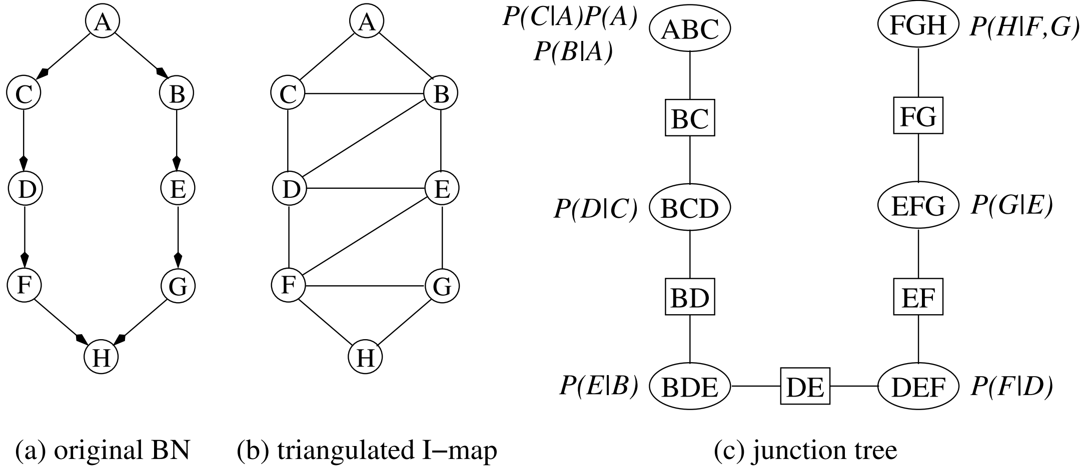


Figure 4.9: Junction tree initialization.

- separator FG: $\phi^*(FG) = \sum_H \psi^*(FGH)$
  $= \sum_H P(H|F,G) = \mathbb{1}_{FG}$;
- clique EFG: $\psi^*(EFG) = \psi(EFG) \times \frac{\phi^*(FG)}{\phi(FG)}$
  $= P_F(G|E) \times \frac{\mathbb{1}_{FG}}{\mathbb{1}_{FG}} = P_F(G|E) = P(G|E,F)$;
- separator EF: $\phi^*(EF) = \sum_G \psi^*(EFG)$
  $= \sum_G P_F(G|E) = \mathbb{1}_{EF}$;
- clique DEF: $\psi^*(DEF) = \psi(DEF) \times \frac{\phi^*(EF)}{\phi(EF)}$
  $= P_E(F|D) \times \frac{\mathbb{1}_{EF}}{\mathbb{1}_{EF}} = P_E(F|D) = P(F|D,E)$;
- separator DE: $\phi^*(DE) = \sum_F \psi^*(DEF)$
  $= \sum_F P_E(F|D) = \mathbb{1}_{DE}$;
- clique BDE: $\psi^*(BDE) = \psi'(BDE) \times \frac{\phi^*(DE)}{\phi(DE)}$
  $= P(B,D,E) \times \frac{\mathbb{1}_{DE}}{\mathbb{1}_{DE}} = P(B,D,E)$.

Let $\psi^{**}(BDE) = \psi^*(BDE) = P(B,D,E)$. Applying Distribute-Evidence on clique BDE results in the following computations:

- separator BD: $\phi^{**}(BD) = \sum_E \psi^{**}(BDE)$
  $= \sum_E P(B,D,E) = P(B,D)$;
- clique BCD: $\psi^{**}(BCD) = \psi^*(BCD) \times \frac{\phi^{**}(BD)}{\phi^*(BD)}$
  $= P(B,C,D) \times \frac{P(B,D)}{P(B,D)} = P(B,C,D)$;
- separator BC: $\phi^{**}(BC) = \sum_D \psi^{**}(BCD)$
  $= \sum_D P(B,C,D) = P(B,C)$;
- clique ABC: $\psi^{**}(ABC) = \psi^*(ABC) \times \frac{\phi^{**}(BC)}{\phi^*(BC)}$
  $= P(A,B,C) \times \frac{P(B,C)}{P(B,C)} = P(A,B,C)$;

- separator DE: $\phi^{**}(DE) = \sum_B \psi^{**}(BDE) = \sum_B P(B,D,E) = P(D,E)$;
- clique DEF: $\psi^{**}(DEF) = \psi^{*}(DEF) \times \frac{\phi^{**}(DE)}{\phi^{*}(DE)} = P(F|D,E) \times \frac{P_{(D,E)}}{\mathbb{1}_{DE}} = P(D,E,F)$;
- separator EF: $\phi^{**}(EF) = \sum_D \psi^{**}(DEF) = \sum_D P(D,E,F) = P(E,F)$;
- clique EFG: $\psi^{**}(EFG) = \psi^{*}(EFG) \times \frac{\phi^{**}(EF)}{\phi^{*}(EF)} = P(G|E,F) \times \frac{P_{(E,F)}}{\mathbb{1}_{EF}} = P(E,F,G)$;
- separator FG: $\phi^{**}(FG) = \sum_E \psi^{**}(EFG) = \sum_E P(E,F,G) = P(F,G)$;
- clique FGH: $\psi^{**}(FGH) = \psi^{*}(FGH) \times \frac{\phi^{**}(FG)}{\phi^{*}(FG)} = P(H|F,G) \times \frac{P_{(F,G)}}{\mathbb{1}_{FG}} = P(F,G,H)$.

As we can see, at the end of the Distribute-Evidence process, all potentials contain joint probabilities. ♦

4.4 Shafer-Shenoy's inference method

At the end of Section 4.2 we described how a Bayesian network could be converted into a junction tree (a.k.a. join tree). The first step, that is the moralization, ensured that the conditional probabilities stored into each node of the Bayes net could also be stored into a compatibility function in the resulting *I*-map. If we refer again to the Bayesian network of Figure 4.10(a), the joint probability distribution P can be decomposed as:

$$\begin{aligned} P(\mathcal{V}) = {} & P(T|H,R)P(S|G,H)P(R|F)P(H|D)P(G|C) \\ & P(F|B)P(D|A,B)P(C|A)P(B)P(A) \end{aligned}$$

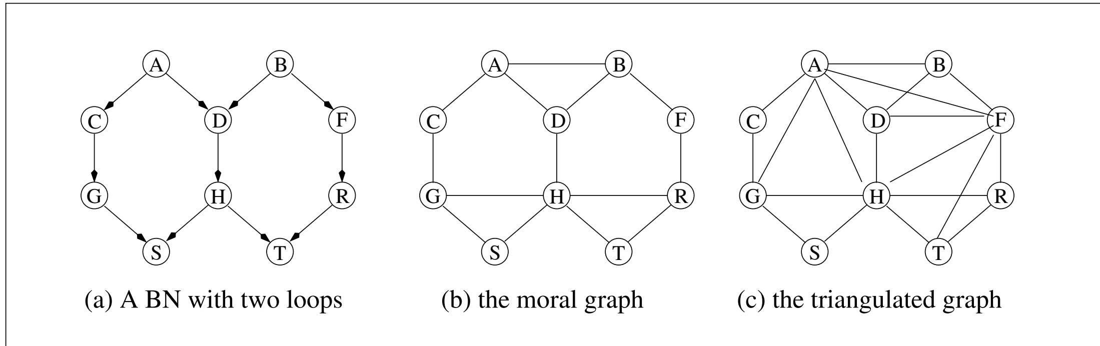
and, if we assign the following values to the compatibility functions of the moral graph of Figure 4.10(b) (the resulting *I*-map):

$$\begin{aligned} a(T,H,R) &= P(T|H,R) & b(S,G,H) &= P(S|G,H) \\ c(R,F) &= P(R|F) & d(H,D) &= P(H|D) \\ e(G,C) &= P(G|C) & f(F,B) &= P(F|B) \\ g(A,B,D) &= P(D|A,B)P(B)P(A) & h(C,A) &= P(C|A), \end{aligned}$$

the joint probability distribution can also be expressed as the product of the compatibility functions, that is:

$$\begin{aligned} P(\mathcal{V}) = {} & a(T,H,R)b(S,G,H)c(R,F)d(H,D)e(G,C) \\ & f(F,B)g(D,A,B)h(C,A). \end{aligned}$$

The graph resulting from triangulation, i.e., that of Figure 4.10(c), has fewer cliques but they contain the cliques of the *I*-map of Figure 4.10(b) — actually, the new cliques



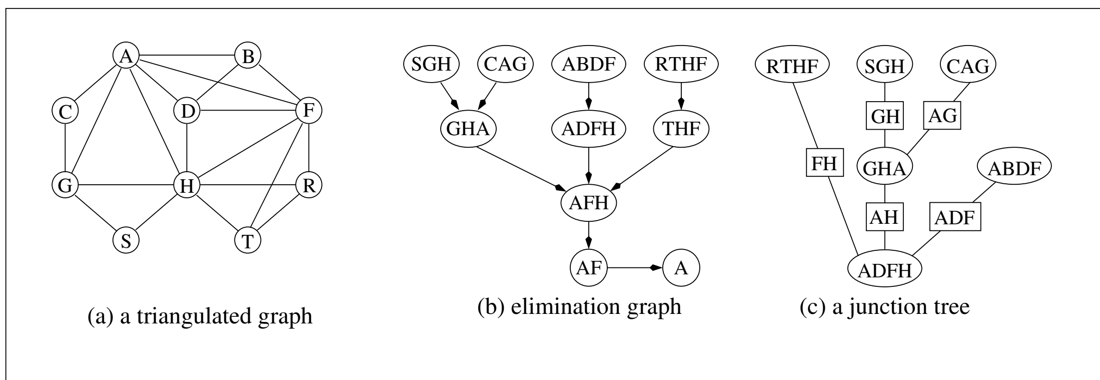
(a) A BN with two loops (b) the moral graph (c) the triangulated graph

Figure 4.10: From a Bayesian network to a triangulated *I*-map.

are $RTFH$, SGH, CAG, GHA, $ABDF$, $ADFH$ — hence the conditional probabilities of the Bayesian network can still be included in the compatibility functions of the triangulated graph. Indeed, if we assign:

$$\begin{array}{ll} a(R,T,F,H) = P(T|H,R)P(R|F) & b(S,G,H) = P(S|G,H) \\ c(A,D,F,H) = P(H|D) & d(G,H,A) = P(A) \\ e(A,B,D,F) = P(D|A,B)P(F|B)P(B) & f(C,A,G) = P(G|C)P(C|A), \end{array}$$

then the joint probability distribution can be expressed as the product of the compatibility functions of the triangulated graph. Now, remember that a junction tree (or join tree) is a tree where the cliques (ellipses) correspond to the cliques of the triangulated graph and the separators are the intersection between adjacent cliques (see Figure 4.11(c)). Consequently, if the potentials we store in the cliques are equal to the compatibility functions we assigned to the triangulated graph, the joint probability distribution P is equal to the product of the potentials of the cliques of the join tree.



(a) a triangulated graph (b) elimination graph (c) a junction tree

Figure 4.11: Elimination and junction trees.

Actually, if we fill the separators with vectors/matrices filled with 1's, the joint distribution over all the random variables represented by the join tree is the product

of all the potentials of the cliques and separators in the graph. The idea in Shafer-Shenoy's propagation algorithm is to propagate evidence while preserving this property. Consider for instance, the Bayesian network of Figure 4.12, which expresses the joint probability distribution over A, C, D, F, G, I as:

$$P(A, C, D, F, G, I) = P(A)P(C|A)P(F|C)P(D)P(G|D)P(I|F, G).$$

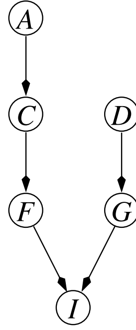


Figure 4.12: A simple Bayesian network.

Triangulating this graph according to elimination sequence A, C, D, F, G, I and constructing the join tree, we obtain the graph of Figure 4.13, where 1_C, 1_F and 1_G stand for vectors and matrices filled with 1's. Note that the product of all these potentials is indeed equal to $P(A, C, D, F, G, I)$.

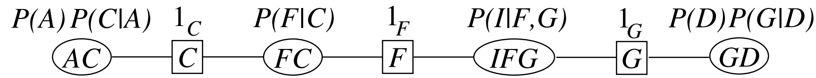


Figure 4.13: A join tree resulting from elimination sequence A, C, D, F, G, I.

Now, assume we want to compute the marginal probability of F. Then we must sum the joint probability distribution over A, C, D, G, I. Consider summing over A:

$$\begin{aligned}\sum_A P(A, C, D, F, G, I) &= \left(\sum_A P(A)P(C|A)\right) P(F|C)P(D)P(G|D)P(I|F, G)\\ &= P(C)P(F|C)P(D)P(G|D)P(I|F, G).\end{aligned}$$

$P(A)P(C|A)$ is stored into the leftmost clique of the join tree, so this clique can perform both the product and the sum, i.e., compute $P(C)$, and it can send this value to its adjacent separator. Removing clique AC from the join tree results in the graph of Figure 4.14. Note that the product of all the potentials is still equal to the joint distribution of the random variables of the graph, i.e., $P(C, D, F, G, I)$.

The same process can be used to sum over C:

$$\begin{aligned}\sum_C P(C, D, F, G, I) &= \left(\sum_C P(C)P(F|C)\right) P(D)P(G|D)P(I|F, G)\\ &= P(F)P(D)P(G|D)P(I|F, G).\end{aligned}$$

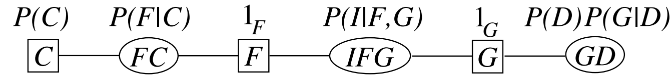


Figure 4.14: The graph after removing clique AC.

$P(C)$ is stored into separator C and $P(F|C)$ is stored into clique FC, hence separator C will send a message $P(C)$ to clique FC. The latter now knows both $P(C)$ and $P(F|C)$, hence it can compute $\sum_C P(C)P(F|C) = P(F)$ and send this probability to its adjacent separator F. Then, both the separator C and clique FC can be removed, thus resulting in the graph of Figure 4.15. Note that, again, the product of all the potentials is equal to the joint distribution of the random variables of the graph, i.e., $P(D, F, G, I)$.

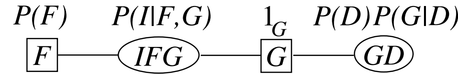


Figure 4.15: The graph marginalizing out C.

The basic property preserved during the whole algorithm is always that the product of all the potentials remaining after each sequence of marginalization is equal to a joint probability distribution. For instance, if we decided to remove the two shaded areas from the join tree of Figure 4.16, we would send the messages described in Figure 4.17 and the product of the remaining potentials would be:

$$P(C)P(F|C)\mathbb{1}_F P(I|F, G)P(G),$$

which is indeed equal to $P(I, C, F, G)$, that is, the joint probability distribution of the remaining variables.

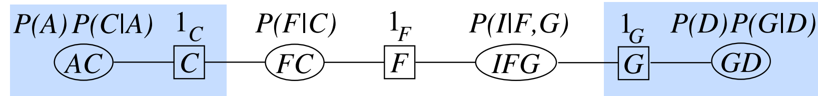


Figure 4.16: Removing parts of the join tree.

As a consequence, if we remove recursively the outer cliques —those that have only one neighbor— as well as their adjacent separators, the last clique remaining must contain the joint probability distribution of its own random variables. Then, if we are interested only in the marginal probability of one of these variables, there just remains to sum the clique's potential over the other variables of the cliques. Naturally, no constraint is imposed on the choice of the last remaining clique, it can be any of those of the original join tree. Hence applying this process sequentially for each of the cliques in the original join tree ensures that the marginal probability of each random variable can be computed. Now, consider the graphs of Figures 4.18 and 4.19. In these graphs, we represented by arrows the messages sent between cliques and separators.

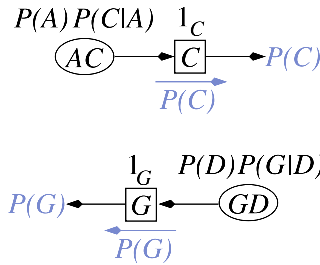


Figure 4.17: Messages in the removed parts of the join tree.

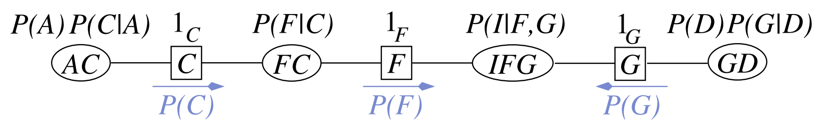


Figure 4.18: Messages when the last remaining clique is IFG.

It is noticeable that most messages are identical, so that applying the above elimination algorithm first to clique IFG and then to clique FC is very inefficient as many computations will be performed twice. Two remarks will help us factorize these computations:

1. note that toward the end of the propagation algorithm, more specifically when there remains only one clique — that of interest — and its adjacent separators, the joint probability of the clique is equal to the product of the potential stored in the clique by those sent by its adjacent separators. For instance, in Figure 4.18, the joint probability $P(I, F, G)$ is equal to the product of $P(I|F, G)$ by $P(F)$ and $P(G)$. This property is local. Consequently, not deleting during the algorithm the cliques/separators from the graph just after they send their messages does not change the fact that the joint probability of the last remaining clique is equal to the product of the potential stored in that clique by those sent by its adjacent separators.

2. note that when a clique C_i sends a message to a separator S, this message is always the sum over the variables in $C_i \backslash S$ of the product of the potential stored in C_i by the messages sent by all separators adjacent to C_i except S. As a consequence, messages sent in opposite directions to a separator are independent. This suggests that instead of keeping only one potential in each separator, we should store two potentials: one for each message in each direction.

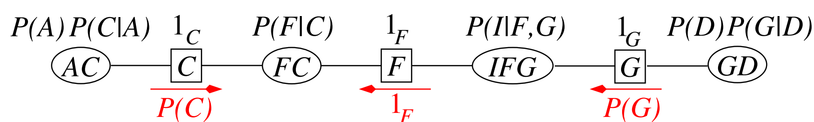


Figure 4.19: Messages when the last remaining clique is FC.

The last remark suggests that if an algorithm can be found that ensures that both messages are computed, then, by remark 1, it will be sufficient to multiply, for each clique of interest, the messages sent to this clique by its internal potential to get the joint probability of the clique. In our example, this would lead to consider sending all the messages of Figure 4.20 using only one algorithm.

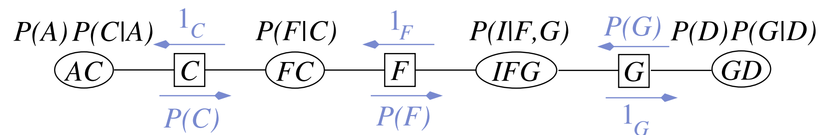


Figure 4.20: Efficient messages sent in the join tree.

The following algorithm [Sha96, SS90a, SS90b] will ensure that all separators will receive messages in both directions:

Algorithm 4.8 (Shafer-Shenoy's propagation method) *Let $(\mathcal{V}, \mathcal{A}, P)$ be a Bayesian network and let $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ be a join tree for this network.*

1. *Clique initialization: store the conditional probabilities of the P's decomposition into the potentials of the cliques of $\mathcal{C}$ (if some cliques received no probability, let their potential be filled with 1's).*

2. *Separator initialization: assign to each separator 2 potentials, that is one for each direction, filled with 1's. Now, the product of all the cliques/separators potentials should be equal to P.*

3. *Propagation: Each node (clique/separator) of the join tree sends messages toward all of its neighbors using the following two rules:*

   (a) *Before sending a message toward its neighbor X, node Y waits for messages coming from all its other neighbors;*

   (b) *the message sent by clique Y to separator X is the sum over the variables in $Y \backslash X$ of the product of the potential stored in Y by the messages received by Y from all of its neighbors except X. The message sent by separator X to a clique Z is simply the message it received from its other neighbor.*

To highlight the connection with Jensen's algorithm, Shafer-Shenoy's method can be implemented by selecting arbitrarily a clique we will call the *root* and applying functions *Collect-Evidence* and *Distribute-Evidence* below on this clique:

Algorithm 4.9 (Collect-Evidence on clique C_i)

1. *For all cliques C_j adjacent to a separator S_{ji} adjacent to C_i except, if any, that which called* Collect-Evidence *on C_i, call* Collect-Evidence *on C_j.*

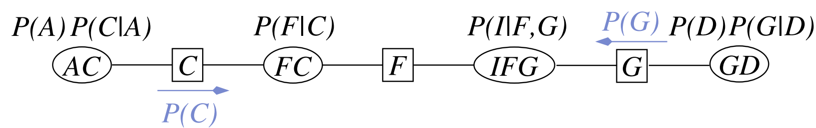
2. *If C_i is not the root, let C_k be the clique that called Collect-Evidence on clique C_i. Compute the product of the potential in C_i by the potentials sent by the C_j's to the S_{ij}'s and sum over $C_i \backslash C_k$. Send the result to S_{ik}.*

Algorithm 4.10 (Distribute-Evidence on clique C_i)

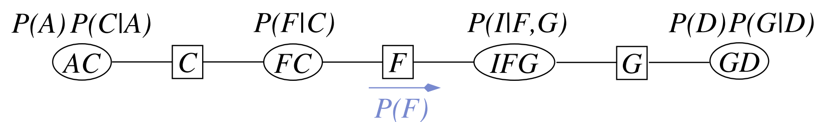
1. *For all cliques C_j adjacent to a separator S_{ij} adjacent to C_i except, if any, that which called* Distribute-Evidence *on C_i, send a message to S_{ij} that is the sum over $C_i \backslash S_{ij}$ of the product of the potential in C_i by messages coming from every separator except S_{ij}.*

2. *Call* Distribute-Evidence *on every C_j's.*

Example 4.4 If we consider again the join tree of Figure 4.20, and if we select (arbitrarily) IFG to be the root, the collect phase above consists in IFG asking cliques FC and GD for messages. Rule (a) in Shafer-Shenoy prevents node FC to send any message. So, it must ask its adjacent node, i.e., AC, for a message. As the latter has no more neighbors, it can send the message it was asked for (see Figure 4.21(a)).



(a) first messages



(b) last message

Figure 4.21: The collect phase.

As described in Algorithm Collect-Evidence, the message sent by clique AC is equal to the sum over $\{A, C\} \backslash \{C\} = \{A\}$ of the product of the potential stored in clique AC, i.e., $P(A)P(C|A)$ by the messages sent by all the separators adjacent to AC except the separator between AC and FC. Here there is no other

separator, hence AC sends message:

$$P(C) = \sum_A P(A)P(C|A).$$

Similarly, GD sends to separator G message $P(G) = \sum_D P(G|D)P(D)$. Now FC has received all the messages it was waiting for and, in turn, it can send its own message to clique IFG (Figure 4.21(b)). This message is equal to the sum over $\{F, C\}\backslash\{C\} = \{F\}$ of the product of its own potential, i.e., $P(F|C)$, by the messages sent by all FC's adjacent separators except the one between FC and IFG. Here there is only one separator: C. Hence FC sends message:

$$P(F) = \sum_C P(C)P(F|C).$$

The distribute phase is similar: clique IFG sends messages to its adjacent nodes, i.e., to FC and GD, see Figure 4.22(a). The message sent to FC is equal to:

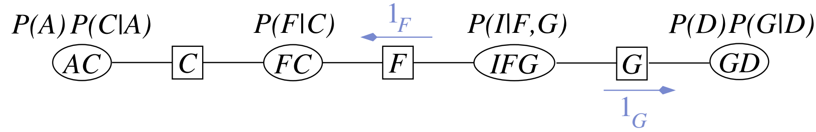
$$\sum_{IG} P(I|F,G)P(G) = \sum_{IG} P(I,G|F) = \mathbb{1}_F$$

and that sent to GD is equal to:

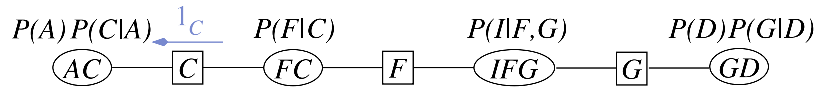
$$\sum_{IF} P(I|F,G)P(F) = \sum_{IF} P(I,F|G) = \mathbb{1}_G.$$

Finally, FC sends the following message to its neighbor (see Figure 4.22(b)):

$$\sum_F P(F|C)\mathbb{1}_F = \sum_F P(F|C) = \mathbb{1}_C.$$



(a) first messages



(b) last message

Figure 4.22: The distribute phase.

If we write on the same graph all the messages sent by the collect-distribute algorithm, we get that of Figure 4.23 which contains all the expected messages. ♦

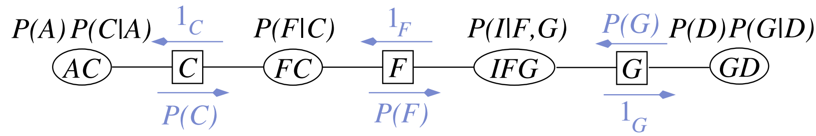


Figure 4.23: All messages sent by the collect-distribute algorithm.

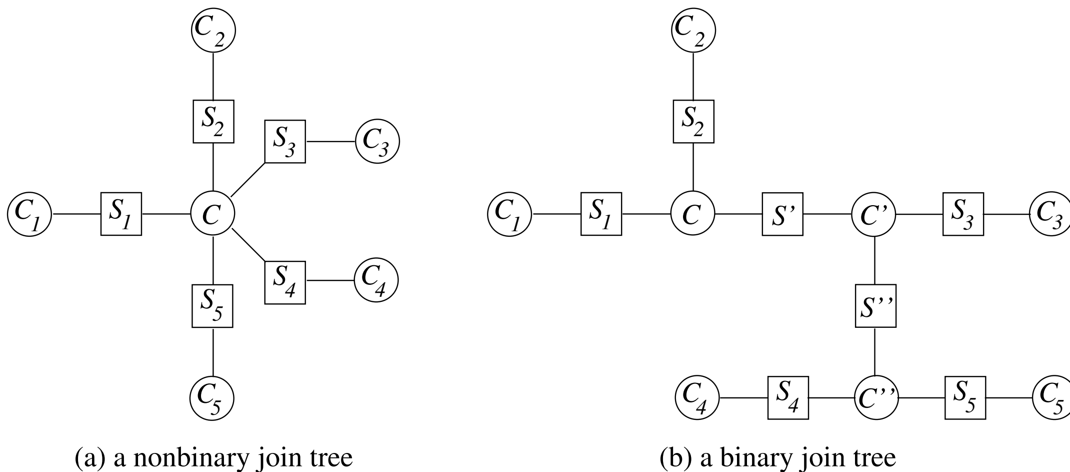
4.5 Binary join trees

A close observation of Shafer-Shenoy's algorithm shows that it can be significantly improved by factorizing some of its local computations. Consider for instance the join tree of Figure 4.24(a) and let us denote by $\phi_{C_j}(S_i)$ the potential stored in separator S_i to be sent to clique C_j and by $\psi(C_j)$ the potential stored in clique C_j. Then, in Shafer-Shenoy's algorithm, message from clique C to clique C_1 is computed as follows:

$$\phi_{C_1}(S_1) = \sum_{C \setminus S_1} \psi(C)\phi_C(S_2)\phi_C(S_3)\phi_C(S_4)\phi_C(S_5).$$

Similarly, a message sent by C to S_2 would be computed using the following expression:

$$\phi_{C_2}(S_2) = \sum_{C \setminus S_2} \psi(C)\phi_C(S_1)\phi_C(S_3)\phi_C(S_4)\phi_C(S_5).$$



(a) a nonbinary join tree (b) a binary join tree

Figure 4.24: A nonbinary join tree, and a corresponding binary join tree.

Note that both messages require computing product $\phi_C(S_3)\phi_C(S_4)\phi_C(S_5)$. Hence if during the distribute phase node C needs sending messages to both C_1 and C_2 —and possibly to other C_i's—, numerous identical multiplications will be computed several times, thus limiting the efficiency of the algorithm. It would be more clever to compute such products only once, store the products somewhere in the computer's memory and use them whenever needed. Binary join trees are a simple graphical tool for storing the

results of these "temporary" multiplications. The idea is to modify the topology of the join tree so that no clique has more than 3 adjacent separators. Consider for instance the graph of Figure 4.24(b), where $C' = C_3 \cup C_4 \cup C_5$ and $\psi(C') = \mathbb{1}_{C'}$, and $C'' = C_4 \cup C_5$ and $\psi(C'') = \mathbb{1}_{C''}$. The messages sent by C to C_1 and C_2 are now

$$\begin{aligned} \phi_{C_1}(S_1) &= \sum_{C \setminus S_1} \psi(C)\phi_C(S_2)\phi_C(S'), \\ \phi_{C_2}(S_2) &= \sum_{C \setminus S_2} \psi(C)\phi_C(S_1)\phi_C(S') \end{aligned}$$

respectively. There is no redundant computation. Actually, $\phi_C(S')$ is equal to product $\phi_C(S_3)\phi_C(S_4)\phi_C(S_5)$. Indeed:

$$\begin{aligned} \phi_{C'}(S'') &= \sum_{C'' \setminus S''} \psi(C'')\phi_{C''}(S_4)\phi_{C''}(S_5) \text{ and} \\ \phi_C(S') &= \sum_{C' \setminus S'} \psi(C')\phi_{C'}(S_3)\phi_{C'}(S''). \end{aligned}$$

$S'' = C'' \cap C'$ but as $C'' \subset C'$, $S'' = C''$. Consequently,

$$\phi_{C'}(S'') = \phi_{C''}(S_4)\phi_{C''}(S_5) = \phi_C(S_4)\phi_C(S_5).$$

Similarly, as $C' = C_3 \cup C_4 \cup C_5$:

$$\phi_C(S') = \phi_{C'}(S_3)\phi_{C'}(S'') = \phi_{C'}(S_3)\phi_C(S_4)\phi_C(S_5) = \phi_C(S_3)\phi_C(S_4)\phi_C(S_5).$$

Now, given an arbitrary join tree, is it possible to convert it into a binary join tree? The answer is positive and Shenoy [She97] proposes the following algorithm:

Algorithm 4.11 (Conversion to a binary join tree) *Let $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ be a join tree. While there remains some clique C whose neighbor set $\mathcal{S} = \{C_1, \dots, C_k\}$ has more than 3 cliques, apply the following steps until $|\mathcal{S}| \leq 3$:*

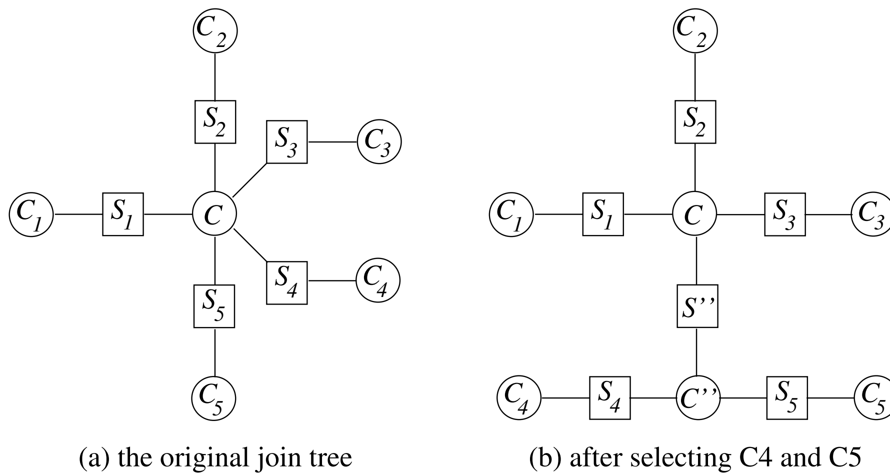
1. *Select arbitrarily two elements C_i and C_j in $\mathcal{S}$ such that for all pairs $C_p, C_q \in \mathcal{S}$, $|C_i \cup C_j| \leq |C_p \cup C_q|$;*

2. *Add a new clique $C_{k'} = C_i \cup C_j$ and link it (through a separator) to C.*

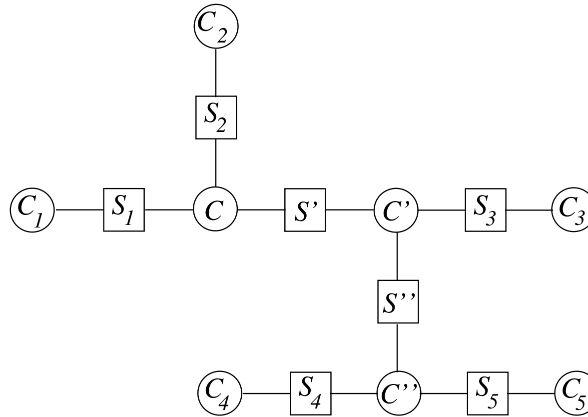
3. *Let S_i and S_j be the separators between C and C_i, and C and C_j, respectively. Remove edges (S_i, C) and (S_j, C) from $\mathcal{G}$ and add edges $(S_i, C_{k'})$ and $(S_j, C_{k'})$. Finally, remove C_i and C_j from $\mathcal{S}$.*

Actually, the above algorithm is not in general optimal because the new cliques created at step 2 may be bigger than what is really necessary: to be optimal, $C_{k'}$ should be equal to $[C_i \cap C_j] \cup [(C_i \cup C_j) \cap (\cup_{C_p \in \mathcal{S} \setminus \{C_i, C_j\}} C_p)]$. However, the way Shenoy constructs his join trees ensures that these two formulas give the same result.

Example 4.5 Consider the join tree of Figure 4.25(a). It is not a binary join tree since clique C has 5 neighbors. Hence we shall apply the above algorithm to transform it into a binary join tree. At the beginning of the algorithm, $\mathcal{S} = \{C_1, C_2, C_3, C_4, C_5\}$. We compare pairs of cliques until we find cliques $C_i, C_j \in \mathcal{S}$ such that for all pairs $C_p, C_q \in \mathcal{S}$, $|C_i \cup C_j| \leq |C_p \cup C_q|$. Assume that C_4 and C_5 are such cliques. Then by step 1 we shall add clique $C'' = C_4 \cup C_5$ to $\mathcal{G}$ and link it to C. Moreover, by step 2, we shall substitute edges (S_4, C) and (S_5, C) by edges (S_4, C'') and (S_5, C'') (see Figure 4.25(b). As for $\mathcal{S}$, we shall update it by adding C'' and deleting C_4 and C_5. Thus $\mathcal{S} = \{C_1, C_2, C_3, C''\}$.



(a) the original join tree (b) after selecting C4 and C5



(c) final binary join tree

Figure 4.25: From a join tree to a binary join tree.

$|\mathcal{S}| > 3$, so we must apply the algorithm again. Assume that now C_3 and C'' are such that for all pairs $C_p, C_q \in \mathcal{S}$, $|C'' \cup C_3| \leq |C_p \cup C_q|$. Then a new clique $C' = C_3 \cup C''$ must be added to $\mathcal{G}$ and it must be linked through a separator, say S', to C. Step 2: edges (S'', C) and (S_3, C) are substituted by (S'', C') and (S_3, C'). Finally, $\mathcal{S}$ becomes $\{C_1, C_2, C'\}$. As $|\mathcal{S}| \leq 3$, the algorithm terminates

and the resulting graph is a binary join tree. ♦

Chapter 5

A comparison between Pearl's and Jensen's algorithms

The last two chapters described propagation methods based on very different concepts: in Chapter 3 Pearl performed computations on a directed graph whereas in Chapter 4 Jensen used an undirected graph. Moreover, these propagation methods also managed cycles in the Bayesian network very differently. Hence it is legitimate to wonder whether one is better than the other. By better we mean a faster algorithm or an algorithm using less memory storage. The answer to this question is not straightforward as both propagation methods have their own advantages and disadvantages. In this section, we will compare both algorithms on two different aspects:

1. How do these algorithms manage cycles in the Bayesian network? Does one method always outperform the other? This will be investigated in Section 5.1.
2. Can the graphical structure give us some clue as to which computations are necessary for evidence propagation and which are not? This topic is developed in Section 5.2.

5.1 Coping with loops in the Bayesian network

As a starting point consider a Bayesian network containing some loops, for instance that of Figure 5.1.

Pearl's algorithm copes with loops using local cutsets. Assume for instance that B is chosen as the custset. Then this algorithm consists in sending through the Bayesian network local messages the sizes of which are indicated on Figure 5.2(a). More precisely, if vertex D is chosen as the root of the algorithm, messages are computed as described below and as illustrated on Figures 5.2(b) and 5.2(c): Vertices B and E first send to A and G respectively messages:

$$\begin{aligned}\lambda_B(A) &= P(B|A),\\ \pi_G(E) &= P(E|B).\end{aligned}$$

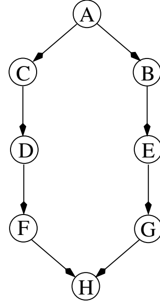
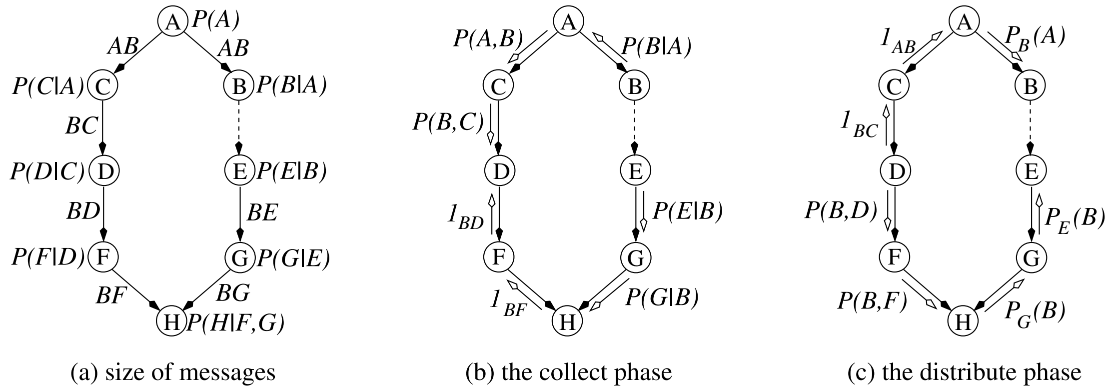


Figure 5.1: A Bayesian network containing a cycle.



(a) size of messages (b) the collect phase (c) the distribute phase

Figure 5.2: A local Cutset and its collect/distribute phases.

Then A and G send to C and H respectively messages:

$$\begin{aligned} \pi_C(A) &= P(A)\lambda_B(A) = P(A)P(B|A) = P(A,B), \\ \pi_H(G) &= \sum_E P(G|E)\pi_G(E) = \sum_E P(G,E|B) = P(G|B). \end{aligned}$$

In turn, C and H send to D and F respectively messages:

$$\begin{aligned} \pi_D(C) &= \sum_A P(C|A)\pi_C(A) = \sum_A P(A,B,C) = P(B,C), \\ \lambda_H(F) &= \sum_{G,H} P(H|F,G)\pi_H(G) = \sum_{G,H} P(H,G|B,F) = \mathbb{1}_{BF}. \end{aligned}$$

The collect phase is completed when F sends to D message:

$$\lambda_F(D) = \sum_F P(F|D)\mathbb{1}_{BF} = \mathbb{1}_{BD}.$$

Now the distribute phase can begin with D sending to C and F the following messages:

$$\begin{aligned} \lambda_D(C) &= \sum_D P(D|C)\mathbb{1}_{BD} = \mathbb{1}_{BC}, \\ \pi_F(D) &= \sum_C P(D|C)P(B,C) = \sum_C P(B,C,D) = P(B,D). \end{aligned}$$

C and F send to A and H respectively messages:

$$\begin{aligned}\lambda_C(A) &= \sum_C P(C|A)\mathbb{1}_{BC} = \mathbb{1}_{AB}, \\ \pi_H(F) &= \sum_D P(F|D)P(B,D) = \sum_D P(B,D,F) = P(B,F).\end{aligned}$$

A sends to B message $P(A)\mathbb{1}_{AB} = P_B(A)$, where $P_B(A)$ is a matrix of size $|A| \times |B|$ constituted by $|B|$ replicas of vector $P(A)$. Similarly, H sends to G the following message:

$$\begin{aligned}\lambda_H(G) &= \sum_{F,H} P(H|F,G)P(B,F) \\ &= \sum_{F,H} P(H|F,G)P(F|B,G)P(B) = \sum_{F,H} P(H,F|B,G)P(B) = P_G(B).\end{aligned}$$

And finally, G sends to E message:

$$\lambda_G(E) = \sum_G P(G|E)P_G(B) = \mathbb{1}_E P(B) = P_E(B).$$

Assume now that each random variable A, B, C, D ,E, F, G and H, can take 10 possible different values. Then the number of summations and products performed by Pearl's algorithm with local conditioning (and B as cutset) is summarized in the table below and it amounts to an overall[1] of 28000 summations and 28200 multiplications.

computation	nb $\times$	nb $+$
$\lambda_B(A) = P(B\|A)$	0	0
$\pi_B(A) = P(A)\mathbb{1}_{AB}$	100	0
$\lambda_C(A) = \sum_C P(C\|A)\mathbb{1}_{BC}$	1000	1000
$\pi_C(A) = P(A)P(B\|A)$	100	0
$\lambda_D(C) = \sum_D P(D\|C)\mathbb{1}_{BD}$	1000	1000
$\pi_D(C) = \sum_A P(C\|A)P(A,B)$	1000	1000
$\lambda_F(D) = \sum_F P(F\|D)\mathbb{1}_{BF}$	1000	1000
$\pi_F(D) = \sum_C P(D\|C)P(B,C)$	1000	1000
$\lambda_G(E) = \sum_G P(G\|E)P_G(B)$	1000	1000
$\pi_G(E) = P(E\|B)$	0	0
$\lambda_H(F) = \sum_{G,H} P(H\|F,G)P(G\|B)$	10000	10000
$\pi_H(F) = \sum_D P(F\|D)P(B,D)$	1000	1000
$\lambda_H(G) = \sum_{F,H} P(H\|F,G)P(B,F)$	10000	10000
$\pi_H(G) = \sum_E P(G\|E)P(E\|B)$	1000	1000

[1] Here, for simplicity, even multiplications with vectors $\mathbb{1}$ were taken into account, although such superfluous computations can be detected and avoided using a simple graph search based algorithm.

Jensen's algorithm on the other hand first transforms the original Bayesian network of Figure 5.1 into a junction tree and then computations are performed in this secondary structure. To do so, the Bayes net is moralized and triangulated (see Figure 5.3(a)) and, then, a junction tree is constructed from this triangulated *I*-map (see Figure 5.3(b)). Once the secondary structure is established, the conditional probabilities stored in the original Bayesian network are inserted into the potentials of the cliques and separators as described below:

$$\begin{array}{rlr} \psi(ABC) &= P(B|A)P(C|A)P(A) = P(A,B,C) & \phi(BC) = \mathbb{1}_{BC} \\ \psi(BCD) &= P_B(D|C) & \phi(BD) = \mathbb{1}_{BD} \\ \psi(BDF) &= P_B(F|D) & \phi(BF) = \mathbb{1}_{BF} \\ \psi(BEG) &= P(E|B)P(G|E) = P(E,G|B) & \phi(BG) = \mathbb{1}_{BG} \\ \psi(BFGH) &= P_B(H|F,G) & \end{array}$$

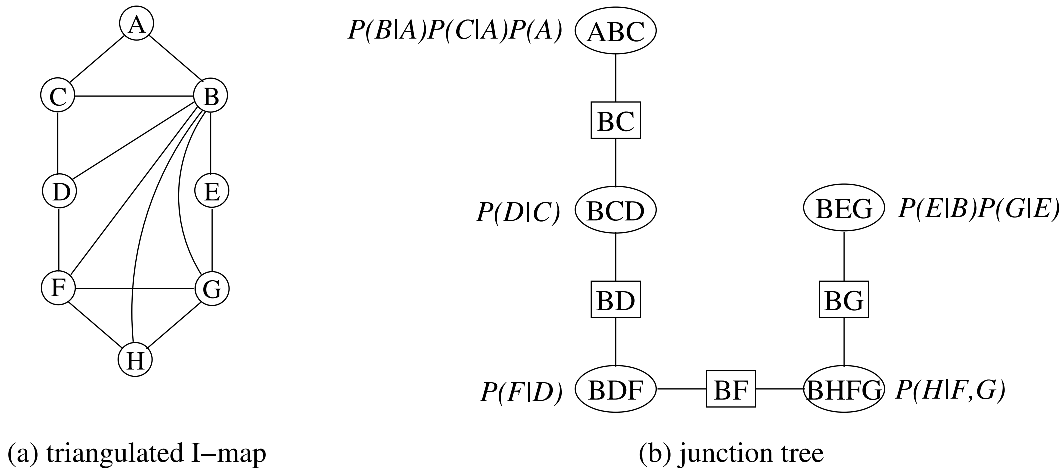


Figure 5.3: A transformation of the Bayes net into a junction tree.

Assume that clique BDF is chosen as the root of the collect and distribute algorithms. The former amounts to performing the following operations:

- Let $\psi^*(ABC) = \psi(ABC) = P(A,B,C)$;
- separator BC computes $\phi^*(BC) = \sum_A \psi^*(ABC) = P(B,C)$;
- clique BCD computes $\psi^*(BCD) = \psi(BCD) \times \frac{\phi^*(BC)}{\phi(BC)} = P(B,C,D)$;
- separator BD computes $\phi^*(BD) = \sum_C \psi^*(BCD) = P(B,D)$;
- Let $\psi^*(BEG) = \psi(BEG) = P(E,G|B)$;
- separator BG computes $\phi^*(BG) = \sum_E \psi^*(BEG) = P(G|B)$;
- clique $BFGH$ computes $\psi^*(BFGH) = \psi(BFGH) \times \frac{\phi^*(BG)}{\phi(BG)} = P(G,H|F,B)$;
- separator BF computes $\phi^*(BF) = \sum_{G,H} \psi^*(BFGH) = \mathbb{1}_{BF}$;
- clique BDF computes $\psi^*(BDF) = \psi(BDF) \times \frac{\phi^*(BD)}{\phi(BD)} \times \frac{\phi^*(BF)}{\phi(BF)} = P(B,D,F)$.

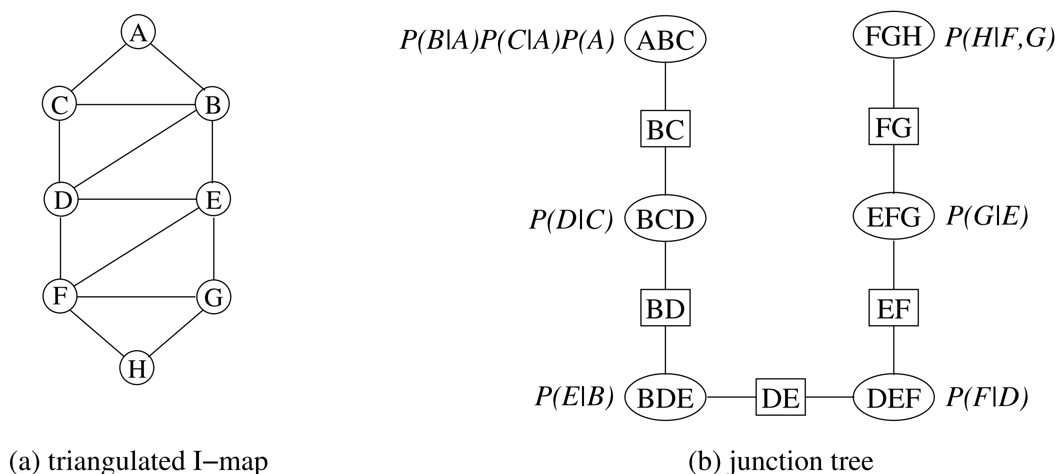
The distribute phase is achieved by:
- Let $\psi^{**}(BDF) = \psi^{*}(BDF)$;
- separator BD computes $\phi^{**}(BD) = \sum_F \psi^{**}(BDF) = P(B,D)$;
- clique BCD computes $\psi^{**}(BCD) = \psi^{*}(BCD) \times \frac{\phi^{**}(BD)}{\phi^{*}(BD)} = P(B,C,D)$;
- separator BC computes $\phi^{**}(BC) = \sum_D \psi^{**}(BCD) = P(B,C)$;
- clique ABC computes $\psi^{**}(ABC) = \psi^{*}(ABC) \times \frac{\phi^{**}(BC)}{\phi^{*}(BC)} = P(A,B,C)$;
- separator BF computes $\phi^{**}(BF) = \sum_D \psi^{**}(BDF) = P(B,F)$;
- clique $BFGH$ computes $\psi^{**}(BFGH) = \psi^{*}(BFGH) \times \frac{\phi^{**}(BF)}{\phi^{*}(BF)} = P(B,F,G,H)$;
- separator BG computes $\phi^{**}(BG) = \sum_{F,H} \psi^{**}(BFGH) = P(B,G)$;
- clique BEG computes $\psi^{**}(BEG) = \psi^{*}(BEG) \times \frac{\phi^{**}(BG)}{\phi^{*}(BG)} = P(B,E,G)$.

The overall number of operations performed by the collect and distribute steps are 26000 additions, 28100 multiplications and 800 divisions. When compared to the 28000 summations and 28200 multiplications required by Pearl, we can conclude that, for this particular cutset and triangulation Jensen's and Pearl's algorithms are roughly equivalent.

Naturally, neither the cutset nor the triangulation are unique and, to complete our comparison, we should study how the propagation algorithms behave using different cutsets and triangulations. As all the variables can take exactly 10 possible values, it is easily seen that whatever the cutset chosen, Pearl's method will always perform 28000 summations and 28200 multiplications. On the contrary, depending on the triangulation, Jensen can outperform significantly Pearl. Consider for instance the junction tree of Figure 5.4(b) and the clique's/separator's potentials:

$$\begin{array}{ll}
\psi(ABC) = P(B|A)P(C|A)P(A) = P(A,B,C) & \phi(BC) = \mathbb{1}_{BC} \\
\psi(BCD) = P_B(D|C) & \phi(BD) = \mathbb{1}_{BD} \\
\psi(BDE) = P_D(E|B) & \phi(DE) = \mathbb{1}_{DE} \\
\psi(DEF) = P_E(F|D) & \phi(EF) = \mathbb{1}_{EF} \\
\psi(EFG) = P_F(G|E) & \phi(FG) = \mathbb{1}_{FG} \\
\psi(FGH) = P(H|F,G). &
\end{array}$$

Let FGH be the root. The collect phase is achieved by:
- Let $\psi^{*}(ABC) = \psi(ABC) = P(A,B,C)$;
- separator BC computes $\phi^{*}(BC) = \sum_A \psi^{*}(ABC) = P(B,C)$;
- clique BCD computes $\psi^{*}(BCD) = \psi(BCD) \times \frac{\phi^{*}(BC)}{\phi(BC)} = P(B,C,D)$;
- separator BD computes $\phi^{*}(BD) = \sum_C \psi^{*}(BCD) = P(B,D)$;
- clique BDE computes $\psi^{*}(BDE) = \psi(BDE) \times \frac{\phi^{*}(BD)}{\phi(BD)} = P(B,D,E)$;
- separator DE computes $\phi^{*}(DE) = \sum_B \psi^{*}(BDE) = P(D,E)$;
- clique DEF computes $\psi^{*}(DEF) = \psi(DEF) \times \frac{\phi^{*}(DE)}{\phi(DE)} = P(D,E,F)$;
- separator EF computes $\phi^{*}(EF) = \sum_D \psi^{*}(DEF) = P(E,F)$;
- clique EFG computes $\psi^{*}(EFG) = \psi(EFG) \times \frac{\phi^{*}(EF)}{\phi(EF)} = P(E,F,G)$;
- separator FG computes $\phi^{*}(FG) = \sum_E \psi^{*}(EFG) = P(F,G)$;
- clique FGH computes $\psi^{*}(FGH) = \psi(FGH) \times \frac{\phi^{*}(FG)}{\phi(FG)} = P(F,G,H)$.



(a) triangulated I–map (b) junction tree

Figure 5.4: Another transformation of the Bayes net into a junction tree.

The distribute phase is achieved by:

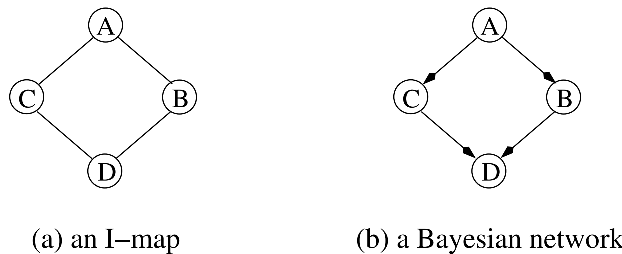
- Let $\psi^{**}(FGH) = \psi^{*}(FGH)$;
- separator FG computes $\phi^{**}(FG) = \sum_H \psi^{**}(FGH) = P(F,G)$;
- clique EFG computes $\psi^{**}(EFG) = \psi^{*}(EFG) \times \frac{\phi^{**}(FG)}{\phi^{*}(FG)} = P(E,F,G)$;
- separator EF computes $\phi^{**}(EF) = \sum_G \psi^{**}(EFG) = P(E,F)$;
- clique DEF computes $\psi^{**}(DEF) = \psi^{*}(DEF) \times \frac{\phi^{**}(EF)}{\phi^{*}(EF)} = P(D,E,F)$;
- separator DE computes $\phi^{**}(DE) = \sum_F \psi^{**}(DEF) = P(D,E)$;
- clique BDE computes $\psi^{**}(BDE) = \psi^{*}(BDE) \times \frac{\phi^{**}(DE)}{\phi^{*}(DE)} = P(B,D,E)$;
- separator BD computes $\phi^{**}(BD) = \sum_E \psi^{**}(BDE) = P(B,D)$;
- clique BCD computes $\psi^{**}(BCD) = \psi^{*}(BCD) \times \frac{\phi^{**}(BD)}{\phi^{*}(BD)} = P(B,C,D)$;
- separator BC computes $\phi^{**}(BC) = \sum_D \psi^{**}(BCD) = P(B,C)$;
- clique ABC computes $\psi^{**}(ABC) = \psi^{*}(ABC) \times \frac{\phi^{**}(BC)}{\phi^{*}(BC)} = P(A,B,C)$.

With this junction tree, Jensen's algorithm requires only 10000 additions, 10000 multiplications and 1000 divisions. It thus outperforms significantly Pearl's algorithm. Actually, this is a general property: Shachter, Andersen and Szolovits [SAS94] have shown that cutset conditioning is equivalent to a triangulation of a very special type[2] and their conclusion was that Jensen's algorithm was always as least as fast as Pearl's algorithm when applied on multiply-connected Bayesian networks (on singly-connected networks, both algorithms are equivalent). The special type of triangulations mentioned above corresponds to those where all fill-ins are adjacent to the cutset variable used by Pearl (in our example, this corresponds to B), see Figure 5.3(a).

[2] In their paper, Shachter, Andersen and Szolovits only considered Global Conditioning, but the property still holds for Local Conditioning.

5.2 *d*-separation analysis

It is well-known that the sets of independencies representable by an *I*-map on one hand and by a Bayesian network on the other hand are not identical: some independencies are representable by an *I*-map but not by a Bayesian network (see Figure 5.5(a), where $B \amalg C|(A, D)$ but $B \not\amalg C|A$) and conversely (see Figure 5.5(b), where $B \amalg C|A$ but $B \not\amalg C|(A, D)$). Hence *a fortiori* junction trees and Bayes nets do not encode the same independencies. Of course, the latter can be used to reduce the computational burden of evidence propagation by avoiding computations that are known to be unnecessary. For instance, when we applied Jensen's algorithm on the junction tree of Figure 5.4(b), we could have avoided the distribute phase as it provided no new information (no potential was affected by this phase). It is thus natural to wonder whether one graphical representation is more prone than the other to provide informations that can be used to reduce the computational burden of evidence propagation.



(a) an I–map (b) a Bayesian network

Figure 5.5: Independencies representable by *I*-maps and Bayesian networks.

To answer this question, let us consider the Bayesian network of Figure 5.6 and one of its junction trees (Figure 5.7). The independencies represented by the structure of the junction tree alone can be captured through its separators. Indeed, assume that evidence e_C has been entered into the tree and that it provides the information that C's value is known for sure, e.g., "$C = c_i$". Let $\phi(\cdot)$ and $\psi(\cdot)$ denote the potentials stored in the separators and cliques respectively after Jensen's algorithm has propagated evidence e_C. Then $\phi(C) = P(C, e_C)$. Now, let e_L be an additional piece of evidence concerning node L and denote by $\phi^*(\cdot)$ and $\psi^*(\cdot)$ the potentials taking into account both e_C and e_L. How do potentials $\phi(\cdot)$ and $\psi(\cdot)$ need be updated to get $\phi^*(\cdot)$ and $\psi^*(\cdot)$? As random variable L is only contained in clique HL, evidence e_L must be entered into this clique. When propagating e_L throughout the junction tree, there is no need to compute anything past separator C, that is cliques AC, ABJ, BK and BE, as well as separators A and B, cannot be affected by e_L. Indeed, before entering e_L separator C contained $P(C, e_C)$ which stated that C could only take value c_i. Knowing the value of a random variable is the maximal information that can be gathered on this variable, hence the new potential to be stored in C can only be $\phi^*(C) = P(C, e_C, e_L) = P(C, e_C)$ and this probability simply states that C can only take value c_i. Consequently, the updating w.r.t. e_L of the potentials by Jensen's algorithm does affect C's potential. Thus $\psi^*(AC)$ that should be equal to $\psi(AC)\frac{\phi^*(C)}{\phi(C)}$ must be equal to $\psi(AC)$. By induction, the potential of separator

A is kept unchanged as well as that of clique ABJ, of separator B and of cliques BK and BE.

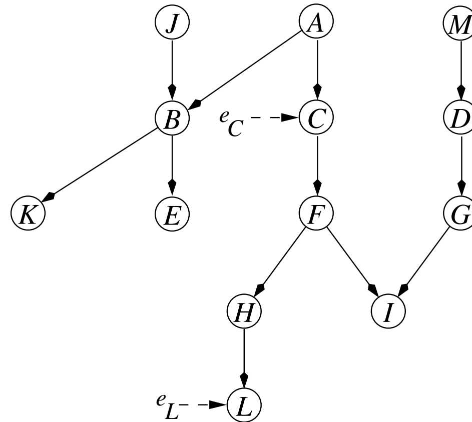


Figure 5.6: A simple Bayesian network.

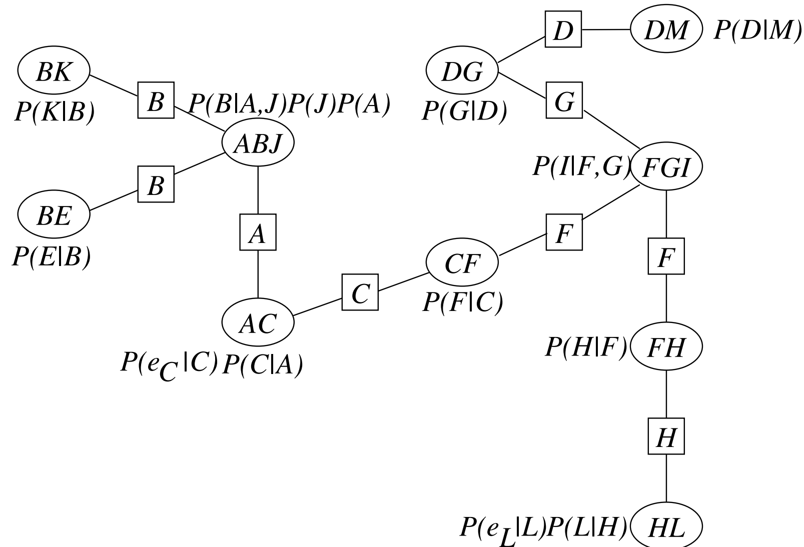


Figure 5.7: A junction tree for the Bayesian network of Figure 5.6.

This is actually a general property: once the value of all the random variables of a separator are known, during further updatings, no information can pass from one side of the separator to the other and thus Jensen's algorithm can be applied separately on the subtrees on each side of the separator. As a consequence, knowing the value of C, it can be deduced graphically —that is, prior to any quantitative computation— that only random variables M, D, F, G, H and I can be affected by evidence e_L. Naturally, when evidence e_C is first entered into the graph, according to the junction tree, all random variables can affected by e_C since, prior to entering e_C, C is not known for sure and

the potential in separator C is vector $\mathbb{1}_C$ —that is, a vector filled with 1's— instead of a vector containing only 0's except for one 1.

Now, is Pearl's algorithm able to do better? Well, the answer is yes and can be understood intuitively using the following argument: Jensen's algorithm encodes the independencies represented by a triangulated *I*-map, that is an undirected graph whereas Pearl's algorithm uses a directed graph. It turns out that the arc orientations is meaningful and provides more informations than simply stating a dependence between two random variables. Hence directed structures provide "more" independence informations and, thus, Pearl's algorithm is more likely than Jensen's method to exploit graphical structure to reduce the computational burden of evidence propagation. As for the independence informations provided by the Bayesian network, consider the possible 2-connections between three random variables. According to Section 2.3:

Figure 5.8(a): $X \amalg Y$ and $X \not\amalg Y|Z$;
any evidence on Z, even one that does not provides the exact value taken by Z makes X and Y dependent.

Figure 5.8(b): $X \not\amalg Y$ and $X \amalg Y|Z$;
only an evidence specifying the exact value of Z makes X and Y independent.

Figure 5.8(c): $X \not\amalg Y$ and $X \amalg Y|Z$;
only an evidence specifying the exact value of Z makes X and Y independent.

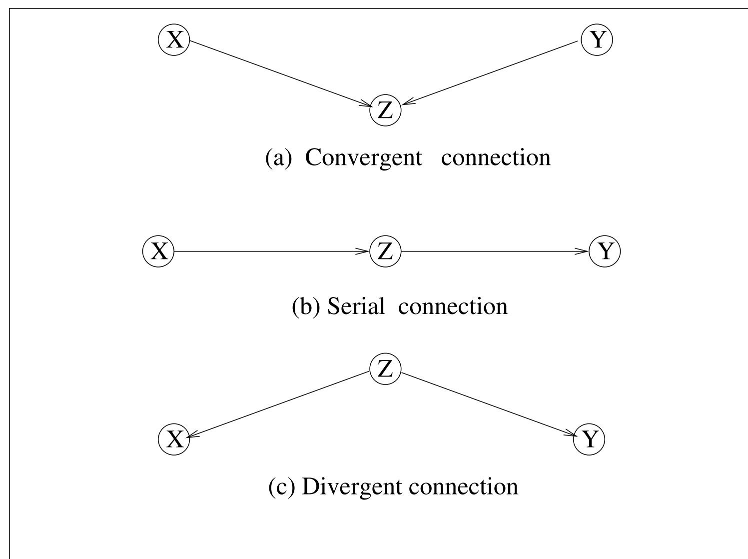


Figure 5.8: Possible connections between random variables.

By induction on these three relations, given a Bayesian network $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ and a set of evidence, the question of whether two arbitrary nodes X and Y are independent can be answered graphically using the following definition [Pea88]:

Definition 5.1 (d-separation) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a Bayesian network and let $\mathcal{E}$ be a set of evidence. Node/variable X is said to be d-separated from Y given $\mathcal{E}$ —or equivalently X is independent from Y— if* **for all** *trails between X and Y, that is, all sequences of arcs linking X and Y, not taking into account the orientations of these arcs, the following two conditions hold:*

1. *There exists no convergent connection on the trail, that is, the trail contains no arcs (A, C) and (D, C), for some arbitrary A, B, C, such that C or one of its descendants in the Bayes net received any evidence in $\mathcal{E}$.*
2. *There exists a divergent or a serial connection, that is, the trail contains some arcs (C, A) and (C, B), or (A, C) and (C, B), such that node C received some evidence in $\mathcal{E}$ specifying its exact value.*

Now, let us come back to the Bayesian network of Figure 5.6. When propagating e_C into the network, using Definition 5.1, it can be deduced that there is no need to compute messages $\lambda_B(J)$, $\lambda_I(G)$, $\lambda_G(D)$ and $\lambda_D(M)$ as random variables J, G, D and M are independent from node C conditionally to e_C. Indeed, the trail between J and C passes through a convergent connection toward B and neither B nor its descendants K and E received any evidence. Similarly, the trails between C and G, C and D, and C and M, pass through I which did not received any evidence. Thus, updating $P(C)$ to $P(C, e_C)$ can affect neither J, nor G, D or M. As a consequence, where Jensen's algorithm needed to send messages throughout the whole of the junction tree Pearl's algorithm can detect the unnecessary computations. The latter, if detected by Jensen's method, would have led to avoiding for instance computing all the messages between clique FGI and separator G, between separator G and clique DG, between clique DG and separator D, and finally between separator D and clique DM.

Consider now adding new information e_L into the network. By Definition 5.1, it can be easily seen that e_L can only affect nodes H, F and I. Hence the updating by Pearl only requires computing messages $\lambda_L(H)$, $\lambda_H(F)$ and $\pi_I(F)$. In comparison, even if Jensen detects that no message is necessary beyond separator C, it still computes messages between cliques FGI and separator G, between separator G and clique DG, between clique DG and separator D, and between separator D and clique DM.

Moreover, one of the main advantages of *d*-separation is that it can be used even when there is no evidence or when evidence does not specify the exact value taken by a random variable. In such cases, separators cannot used. For instance if e_C were the information that $C = c_1$ or $C = c_2$, then by rule 1 of Definition 5.1, it is still true that random variables J, G, D and M are unaffected by e_C, and, when propagating both e_C and e_L it is still true that messages $\lambda_B(J)$, $\lambda_I(G)$, $\lambda_G(D)$ and $\lambda_D(M)$ can be safely dispensed with. On the other hand, when Jensen first propagates e_C, it needs sending messages everywhere in the junction tree. Worst, when updating the potentials to take into account both e_L and e_C, messages still need be sent throughout the whole junction tree as, the value of C being uncertain, separator C cannot break the junction tree into two separate parts.

To conclude this section, *d*-separation is more powerful than separators to identify in the graphical structure independencies used for propagation. This is especially true when evidence does not provide the exact values taken by random variables or when separators contain more than one random variable: passing messages from one side of a separator to the other is unnecessary only if the exact values of *all* the random variables are known for sure. So if only the value one variable is not known for sure, messages need be sent throughout the separator. *d*-separation, by its very definition, avoids this problem.

5.3 Conclusion

To summarize this chapter, the junction tree structure, by way of its triangulation process, is more suited than cutset conditioning for coping with cycles in the Bayesian network. This may be explained by the fact that the former is not afraid to change the very structure of the network, that is, to modify the knowledge about the set of independencies, whereas the latter is reluctant to do so. However, this is the right thing to do as the computations needed to obtain marginal probabilities require multiplying independent messages. But in multiply-connected networks, there is not enough independence for this property to be naturally true. Hence local cutsets cut cycles, thus introducing new independencies, and Pearl's message-passing algorithm can be applied. Now, why should there be only one cut in each cycle? Why not cutting part of the cycle with one cutset and another part with another cutset? This is precisely what happens with junction trees: to enable different cutsets, cycles are first broken into smaller cycles by adding new edges —this is the triangulation process— and, when cycles are small enough, some cutsets are chosen —these are the separators. As we saw in Section 5.1, this proves to be a very efficient technique.

But is this technique only possible in undirected cycles? At first sight, there is no obvious objection to apply it directly in Bayesian networks and, indeed, as we shall see in Chapter 6, the triangulation technique used in Markov networks can be adapted to Bayesian networks. Thus Pearl's algorithm with triangulated BN should be able to manage cycles as well as Jensen's method. Now, what's the point of trying to find such an improvement Pearl's algorithm? Well, Section 5.2 answered this question: *d*-separation can be used by Pearl to reduce the computational burden of evidence propagation by avoiding unnecessary computations. So, if we could triangulate Bayesian networks, that is, perform a "directed" triangulation, we should be able to provide a directed propagation method that is at least as fast as Jensen's method. The next chapters will explain in details how this can be achieved.

Chapter 6

Triangulated Bayesian Network

As we saw in chapter 3, directed propagation methods — for instance the polytree algorithm — use a message-passing scheme directly on the original Bayesian network whereas undirected propagation methods first construct a secondary structure that is more suited for the computations. This is the very use of this secondary structure that makes undirected algorithms much more efficient than directed ones. Thus, one may wonder whether a directed secondary structure similar — at least in spirit — to join/junction trees may improve Pearl's algorithm sufficiently to make it competitive with Jensen's algorithm. The aim of this chapter is to provide a positive answer to this question. Thus, the new variant of Pearl's algorithm we consider here performs computations in two steps, exactly as in Jensen's scheme:

1. Construction of a new directed graph,
2. Propagation based on local conditioning in this new structure.

6.1 Constructing a new DAG: the triangulated BN

In this section, it will be shown that for any elimination tree and any junction tree, there exists a directed graph that precisely captures the same conditional independencies. Although Jensen's algorithm does not use elimination trees, those are important here because the decomposition of the joint probability they provide is closely related to a product of conditional probabilities, which is precisely what would be provided by a directed acyclic graph.

By [CDLS99, p.59], if $\mathcal{G}_T = (\mathcal{V}, \mathcal{E}_T)$ is a triangulated undirected graph induced by an elimination sequence σ, and if $\mathcal{G}_J = (\mathcal{C}, \mathcal{E}_J)$ and $\mathcal{G}_E = (\mathcal{D}, \mathcal{A}_E)$ are a junction tree and the elimination tree of $\mathcal{G}_T$ respectively, then $\mathcal{C} \subseteq \mathcal{D}$. So, for simplicity of notation, in the rest of this section, we assume the cliques in $\mathcal{C}$ and $\mathcal{D}$ are indexed similarly to the node eliminated when they were created, e.g., $\mathcal{C} = \cup_{i \in N} \{C_{\sigma(i)}\}$ for some set N, where $C_{\sigma(i)}$ was created by eliminating node $X_{\sigma(i)}$.

Lemma 6.1 *Let $\mathcal{G}_T = (\mathcal{V}, \mathcal{E}_T)$ be a triangulated graph induced by an elimination sequence σ, and let $\mathcal{G}_J = (\mathcal{C}, \mathcal{E}_J)$ and $\mathcal{G}_E = (\mathcal{D}, \mathcal{A}_E)$ be a junction tree and the elimination tree of $\mathcal{G}_T$ respectively. Consider the following expressions:*

$$\Pi_J = \frac{\prod_{C_{\sigma(i)} \in \mathcal{C}} P(C_{\sigma(i)})}{\prod_{(C_{\sigma(i)}, C_{\sigma(j)}) \in \mathcal{E}_J} P(C_{\sigma(i)} \cap C_{\sigma(j)})}, \quad \Pi_E = \frac{\prod_{D_{\sigma(i)} \in \mathcal{D}} P(D_{\sigma(i)})}{\prod_{(D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}_E} P(D_{\sigma(i)} \cap D_{\sigma(j)})}.$$

Then all the elements in the expression of Π_J are also contained in the expression of Π_E. Moreover, after removing from Π_E all the elements of Π_J, the remaining expressions on the numerator of Π_E are the same as those on the denominator, with the same multiplicities.

Since Π_J is known to be a decomposition of the joint probability on $\mathcal{V}$ (see [JLO90] or [CDLS99, p.59]), the above lemma has the following consequence:

Corollary 6.1 *Let $\mathcal{G}_T = (\mathcal{V}, \mathcal{E}_T)$ be a triangulated graph induced by an elimination sequence σ, and let $\mathcal{G}_J = (\mathcal{C}, \mathcal{E}_J)$ and $\mathcal{G}_E = (\mathcal{D}, \mathcal{A}_E)$ be a junction tree and the elimination tree of $\mathcal{G}_T$ respectively. Then the joint probability on $\mathcal{V}$ factorizes as:*

$$P(\mathcal{V}) = \frac{\prod_{D_{\sigma(i)} \in \mathcal{D}} P(D_{\sigma(i)})}{\prod_{(D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}_E} P(D_{\sigma(i)} \cap D_{\sigma(j)})}.$$

Moreover, $\mathcal{G}_E$ captures precisely the same conditional independencies as $\mathcal{G}_J$.

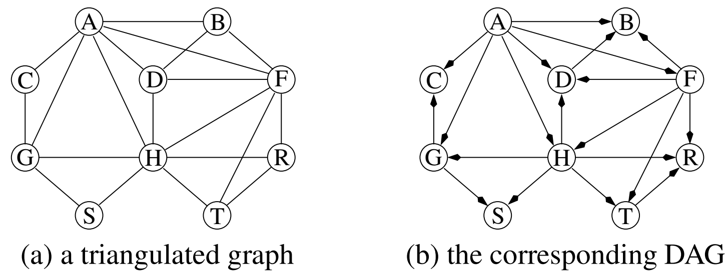
One key aspect of elimination trees that follows directly from their definitions is that if $(D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}_E$, then $D_{\sigma(i)} \cap D_{\sigma(j)} = D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}$ (see the proof of property 1 of lemma 6.2) and $P(D_{\sigma(i)})/P(D_{\sigma(i)} \cap D_{\sigma(j)}) = P(X_{\sigma(i)} | D_{\sigma(i)} \backslash \{X_{\sigma(i)}\})$. Moreover, nodes in $\mathcal{D}$ have at most one child. Consequently, as shown in the following corollary, the decomposition provided by elimination trees is a product of conditional probabilities, and thus can be precisely captured by a directed graph.

Corollary 6.2 *Let $\mathcal{G}_T = (\mathcal{V}, \mathcal{E}_T)$ be a triangulated graph induced by an elimination sequence σ, and let $\mathcal{G}_J = (\mathcal{C}, \mathcal{E}_J)$ and $\mathcal{G}_E = (\mathcal{D}, \mathcal{A}_E)$ be a junction tree and the elimination tree of $\mathcal{G}_T$ respectively. Then DAG $\mathcal{G}' = (\mathcal{V}, \mathcal{A}')$, where $\mathcal{A}' = \{(X_{\sigma(j)}, X_{\sigma(i)}) : X_{\sigma(j)} \in D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}\}$, precisely captures the same conditional independencies as $\mathcal{G}_E$ and $\mathcal{G}_J$. Such a DAG will be called a "triangulated" DAG.*

This corollary provides a simple algorithm for constructing $\mathcal{G}'$: as $D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}$ is the set of vertices in $\mathcal{G}_T$ adjacent to $X_{\sigma(i)}$ before the latter is eliminated, $\mathcal{A}'$ can

be deduced from $\mathcal{E}_T$ by: i) setting $\mathcal{A}' = \mathcal{E}_T$ and ii) orienting in $\mathcal{A}'$ undirected edges adjacent to each $X_{\sigma(i)}$, selected according to elimination sequence σ, toward $X_{\sigma(i)}$.

Example 6.1 The triangulated graph of Figure 6.1(a), resulting from elimination sequence S, C, G, R, T, B, D, H, F, A, yields both the junction tree of Figure 4.11(c) and the directed graph of Figure 6.1(b). Indeed, in the triangulated graph, vertex S's neighbors are G and H. Hence edges (G, S) and (H, S) are replaced by arcs (G, S) and (H, S) in the directed graph. C's elimination involves replacing edges (A, C) and (G, C) by arcs (A, C) and (G, C). When eliminating G, we need only replace edges (A, G) and (H, G) because (G, C) and (G, S) are already directed. The process goes on until all vertices have been eliminated (according to the elimination sequence). ♦



(a) a triangulated graph (b) the corresponding DAG

Figure 6.1: From triangulated graphs to directed graphs.

Example 6.2 Consider the graph of Figure 6.2(b). It can be thought of as triangulated since there does not exists any "undirected" cycle of length 4 or more without chords. If this graph were undirected, it would correspond to a triangulated graph resulting from an elimination sequence such as $\sigma = (G, F, A, C, B, D, E)$. Thus, as shown by corollary 6.2, to get the orientations shown in Figure 6.2(b), it is sufficient, for every node, say X, in σ and in the order specified by σ, to orientate each undirected edge adjacent to X toward X.

Let us compare the TBN to the junction tree of Figure 6.2(c). Note that both graphs encode precisely the same conditional independencies. ♦

The directed acyclic graph obtained in corollary 6.2 has a serious drawback: the conditional probabilities of its nodes given their parents differ from those stored in the original BN. However they are needed to provide a complete description of the joint probability of the random variables. Fortunately, the following section will provide a way for retrieving these new conditional probabilities.

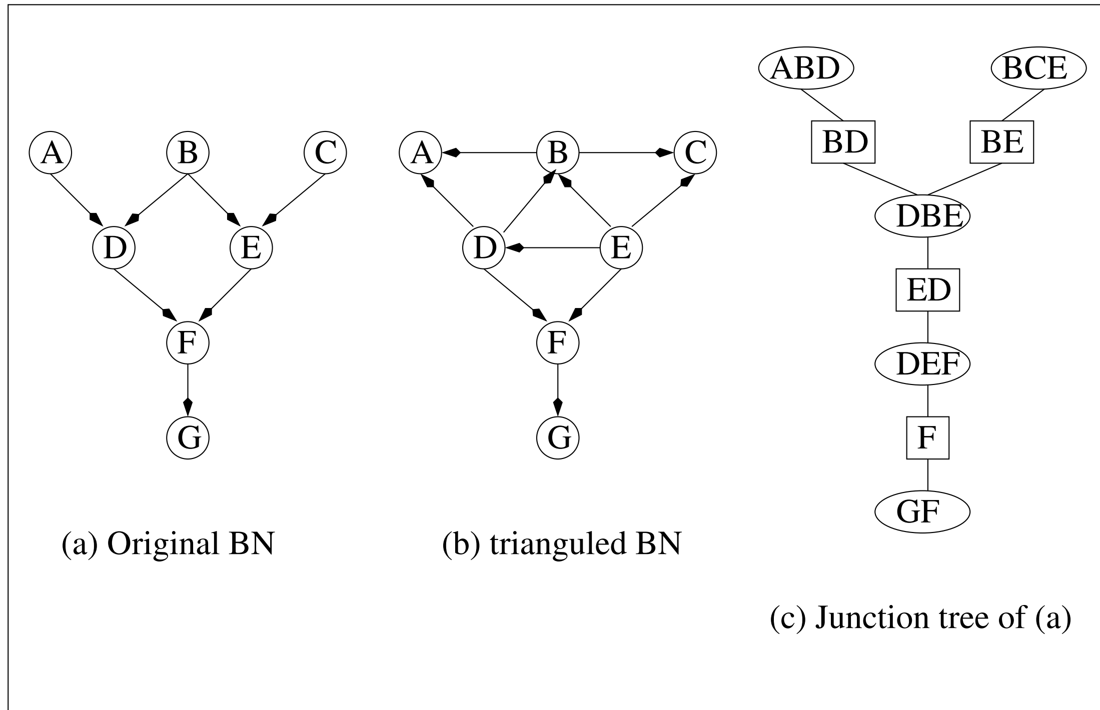


Figure 6.2: From BN to triangulated BN and to junction tree.

6.2 Arc reversal and the generation of new probabilities

To compute the new conditional probabilities mentioned in the preceding section, remark that the differences between the original directed graph and the last one are:

1. the addition of some arcs;
2. the reversal of other arcs.

The update of conditional probabilities after arc additions is simple: their dimension is increased and their content duplicated. For instance adding arc (X, Y) changes $P(Y|A, B)$ into $P(Y|A, B, X)$ and for all values x of X, $P(Y|A, B, X = x) = P(Y|A, B)$. Arc reversals are more subtle to handle. Fortunately, the following theorem due to [Sha86] indicates how the update shall be performed:

Theorem 6.1 (Reversal of arc (X, Y)) *Let $(\mathcal{V}, \mathcal{A}, P)$ be a BN in which node X is a parent of Y (see Figure 6.3.a) and in which there is no other path from X to Y. Let* Pa *denote the set of parents of a node in this BN, i.e.,* $\mathrm{Pa}(X) = \{Z_i : i \in I\} \cup \{V_j : j \in J\}$ *and* $\mathrm{Pa}(Y) = \{X\} \cup \{V_j : j \in J\} \cup \{W_k : k \in K\}$. *Then triple $(\mathcal{V}, \mathcal{A}', P')$ resulting from the reversal of arc (X, Y) into (Y, X) and from the addition of arcs (W_k, X), $k \in K$, and (Z_i, Y), $i \in I$, is a Bayesian network. Moreover, if* Pa' *and* P' *denote sets of parents and probabilities in the resulting BN, the following properties hold:*

- $Pa'(Y) = (\mathrm{Pa}(X) \cup \mathrm{Pa}(Y)) \backslash \{X\}$;
- $Pa'(X) = (\mathrm{Pa}(X) \cup \mathrm{Pa}(Y) \cup \{Y\}) \backslash \{X\}$;
- $P'(Y|\mathrm{Pa}'(Y)) = \sum_X P(Y|\mathrm{Pa}(Y))P(X|\mathrm{Pa}(X))$;
- $P'(X|\mathrm{Pa}'(X)) = \dfrac{P(Y|\mathrm{Pa}(Y)P(X|\mathrm{Pa}(X))}{P'(Y|\mathrm{Pa}'(Y))}$.

Finally, both BN represent the same probability distribution, i.e., $P'(\mathcal{V}) = P(\mathcal{V})$ for all values of the random variables in $\mathcal{V}$.

(a) before arc reversal (b) after arc reversal

Figure 6.3: The reversal of arc (X, Y).

At first sight, a recursive application of theorem 6.1 (and maybe the addition of a few arcs) shall be sufficient to transform an original BN into a "triangulated" DAG. This is true but care must be taken in the sequence of applications because arc reversal cannot be performed if there exists multiple paths from X to Y (else this would create directed cycles). Fortunately, the following proposition exhibits good candidates for arc reversal:

Proposition 6.1 *Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a DAG, $\mathcal{V} = \{X_1, \ldots, X_n\}$, and let σ be an elimination sequence inducing an undirected and a directed triangulated graphs, say $\mathcal{G}_E = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}_T = (\mathcal{V}, \mathcal{A}_T)$. Let i_0 be the highest index in $\{1, \ldots, n\}$ such that, for all $i < i_0$, the set of arcs adjacent to $X_{\sigma(i)}$ in $\mathcal{A}$ equals that in $\mathcal{A}_T$. Assume that $\mathcal{V}_{\sigma(i_0)} = \{Y : (X_{\sigma(i_0)}, Y) \in \mathcal{A} \backslash \mathcal{A}_T\}$ is non empty. Then there exists a node $Z \in \mathcal{V}_{\sigma(i_0)}$ without ancestor in $\mathcal{V}_{\sigma(i_0)}$ and theorem 6.1 can be applied in $\mathcal{G}$ on arc $(X_{\sigma(i_0)}, Z)$. Moreover, if the resulting graph is triangulated again with sequence σ, then the triangulated graphs obtained are still $\mathcal{G}_E$ and $\mathcal{G}_T$.*

A recursive application of this proposition justifies the following algorithm:

Proposition 6.2 (Directed triangulation of a BN) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a DAG, $\mathcal{V} = \{X_1, \ldots, X_n\}$, and let σ be an elimination sequence inducing a directed triangulated graph $\mathcal{G}_T = (\mathcal{V}, \mathcal{A}_T)$. Applying the function below successively to all the variables in $\mathcal{V}$ in the order of their elimination transforms $\mathcal{G}$ into $\mathcal{G}_T$:*

function elimination(*node* X)
1 **while** $\mathcal{V}' = \{Y : (X, Y) \in \mathcal{A} \backslash \mathcal{A}_T\}$ *is non empty* **do**
2 *choose any $Z \in \mathcal{V}'$ without ancestor in $\mathcal{V}'$*
3 *apply theorem 6.1 to reverse arc (X, Z) in $\mathcal{A}$*
4 *update conditional probabilities according to the formulas of theorem 6.1*
5 **done**
6 **for all** *arcs $(Y, X) \in \mathcal{A}_T \backslash \mathcal{A}$ adjacent to X* **do**
7 *add these arcs to $\mathcal{A}$*
8 *update X's conditional probability table duplicating it as many times as Y's number of values, i.e., add a dimension in Y to the table*
9 **done**

Example 6.3 As an illustration of this proposition, let us transform the graph of Figure 6.4(a) into that of Figure 6.4(f). Remember that the latter resulted from elimination sequence S, C, G, R, T, B, D, H, F, A. Applying function "elimination" on S neither reverses nor adds any arc since all the arcs adjacent to S are properly directed. Applying "elimination" on C reverses arc (C, G), thus implying the addition of a new arc: (A, G). Note that the resulting graph (Figure 6.4(b)) contains no directed cycle. C's conditional probability table is updated using the formulas of theorem 6.1, i.e., $P(C|A)$ becomes $P(C|A, G)$. The application of function "elimination" on G does not reverse any arc, but it adds one: arc (H, G) was indeed missing. Of course, G's conditional probability table must be updated accordingly, i.e., $P(G|A)$ is duplicated to become $P(G|A, H)$. Applying function "elimination" on R results in the reversal of arc (R, T), which in turn involves the addition of arcs (F, T) and (H, R) (Figure 6.4(d)). Elimination on T keeps the graph unchanged. The application of elimination on B reverses arcs (B, F) and (B, D), which induces the addition of arc (A, B) (see Figure 6.4(e)). And so on until elimination has been applied on all nodes, which results in the graph of Figure 6.4(f). ♦

6.3 An efficient variation of the polytree algorithm

At this stage, we have a new multiply-connected Bayesian network with the appropriate conditional probabilities. Propagating informations within this new network can be performed using the algorithms described in Section 3, for instance using Local Conditioning. Let us recall briefly how the latter works:

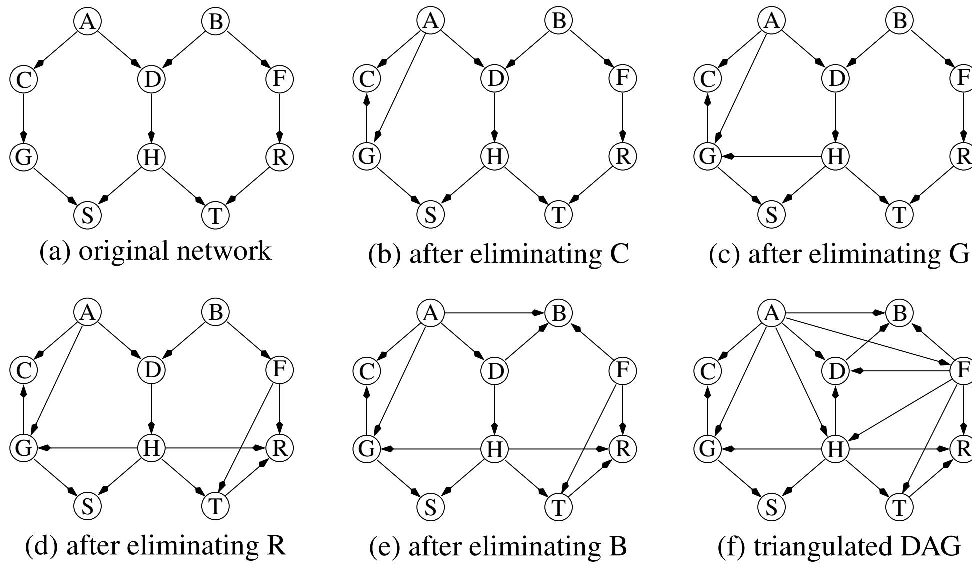


Figure 6.4: Applying arc reversal and arc addition.

Algorithm 6.1 (polytree algorithm with local conditioning) *Let* $(\mathcal{V}, \mathcal{A}, P)$ *be a BN. To compute a posteriori marginal probabilities of all the random variables in* $\mathcal{V}$*, label the arcs using algorithm 3.2, choose arbitrarily a node* R *and apply successively to* R *functions* Collect-Evidence *and* Distribute-Evidence *below:*

Collect-Evidence on node X:

1. *for all nodes* Y *adjacent to* X *except, if any, that which called* Collect-Evidence *on* X*, call* Collect-Evidence *on* Y*,*
2. *if* $X \neq R$*, send to the node* Y *that called* Collect-Evidence *on* X *message* $\pi_Y(X)$ *(resp.* $\lambda_X(Y)$*) if* X *is a parent (resp. a child) of* Y *(see below the expressions for* π *and* λ*).*

Distribute-Evidence on node X: *for all nodes* Y *adjacent to* X *except, if any, that which called* Distribute-Evidence *on* X*, do:*

1. *send to* Y *message* $\pi_Y(X)$ *(resp.* $\lambda_X(Y)$*) if* X *is a parent (resp. a child) of* Y*,*
2. *call* Distribute-Evidence *on* Y.

using labeled $\pi - \lambda$ *messages (see Figure 6.5):*

- $$\pi_Y(X) = \sum_{Z \in \mathcal{V} \backslash C_{XY}} \left[P(X, e_X | \text{Pa}(X)) \times \prod_{i=1}^{k} \pi_X(U_i) \times \prod_{j=1}^{m} \lambda_{Y_j}(X) \right];$$
- $$\lambda_Y(X) = \sum_{Z \in \mathcal{V} \backslash C_{XY}} \left[P(Y, e_Y | \text{Pa}(Y)) \times \prod_{i=1}^{r} \pi_Y(V_i) \times \prod_{j=1}^{p} \lambda_{S_j}(Y) \right].$$

The λ*'s are messages from children to parents and the* π*'s are messages from parents to children.*

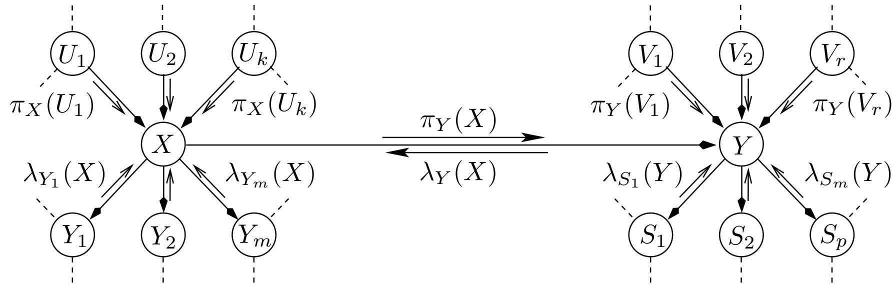


Figure 6.5: A generic graph for the the $\pi - \lambda$ messages.

We will now show that the above algorithm applied on the triangulated DAG resulting from the application of proposition 6.2 has a complexity at most equal to that of HUGIN. Since triangulated DAG usually contain cycles, we shall select variables for cutset conditioning. To help us in this choice, remember that, for every node X_i, X_i and its parents form a clique. Consider now a clique such as that of Figure 6.6(a). Since

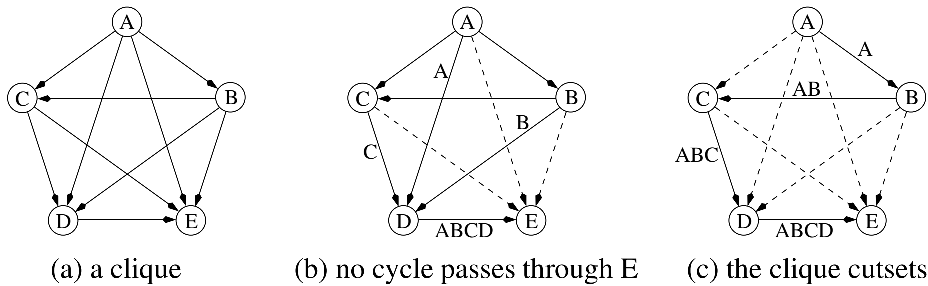


Figure 6.6: Cutsetting within cliques.

the graph is directed and complete, there exists an order on the nodes (a node X is after a node Y in this order if there exists an arc (Y, X)), here the order is A, B, C, D, E. Since the graph is complete, there exist some arcs between all nodes in every triple of nodes, in particular every triple containing nodes D and E: ADE, BDE and CDE. It is easily seen that after cutting arcs (A, E), (B, E) and (C, E), no more cycle passes through E. It is remarkable that selecting these arcs does not affect the labels of any other arc except the last remaining one adjacent to E (in Figure 6.6(b), labels are written beside their associated arcs) and, moreover, the label of the latter is included in the set of variables of the clique. This property always hold when cutting arc (X, Z) in triples (X, Y, Z) such that X and Y are parents of Z and X is parent of Y. A process similar to that used to remove cycles passing through E can be applied to the other nodes, D, C, etc, and leads to the graph of Figure 6.6(c).

During the creation of the triangulated DAG, each time a node was eliminated, this node as well as its parents formed a clique. Hence, applying the above paragraph to all the nodes in the order of their elimination ensures that no cycle will remain. Moreover, labels will be included in cliques, which will guarantee that the complexity of Faÿ and Jaffray's algorithm with these labels is at most equal to that of HUGIN. This motivates the following proposition:

Proposition 6.3 (A new variation of Pearl's method) *Let* $(\mathcal{V}, \mathcal{A}, P)$*,* $\mathcal{V} = \{X_1, \ldots, X_n\}$*, be a BN and let* σ *be an elimination sequence. Let* $\mathcal{G}' = (\mathcal{V}, \mathcal{A}')$ *be the triangulated DAG of proposition 6.2. The application on* $\mathcal{G}'$ *of* Collect-Evidence *and* Distribute-Evidence *together with the cutset algorithm below computes correctly a posteriori marginal probabilities. Moreover, the computational complexity is at most equal to that of HUGIN (with the same elimination sequence).*

Cutset algorithm:

For i *varying from* 1 *to* n **do**

Let j *be the smallest index such that arc* $(X_{\sigma(j)}, X_{\sigma(i)})$ *belongs to* $\mathcal{A}'$ *in all cycles (triangles)* $X_{\sigma(i)}X_{\sigma(j)}X_{\sigma(k)}$*. Cut all arcs* $(X_{\sigma(k)}, X_{\sigma(i)})$*'s*

done

Finally, messages $\pi - \lambda$ *are equal to:*

$$\pi_Y(X) = P(C_{XY}, e^+_{XY}) \quad \textit{and} \quad \lambda_Y(X) = P(e^-_{XY}|C_{XY}).$$

The construction of our secondary structure relies more heavily on elimination sequences than junction trees. Consider for instance a complete graph of three nodes A, B and C. All elimination sequences will lead to clique ABC, but not to the same DAG because the arc's directions depend on the elimination order. Thus, for a given junction tree, there may certainly be a "best" triangulated DAG, or equivalently a best elimination sequence. Probably, this shall be that which changes as less as possible the original network, as it both minimizes the effort to compute the required conditional probabilities and it enables *d*-separation analysis (the analysis showing where informations have an impact in the network, that can be used to avoid unnecessary inference computations). This shall be promising for future works.

The computations presented here are similar to those of [SS90b] in that they never perform any division. In this respect, they outperform HUGIN. However, [SS90b]'s algorithm may be thought of as better than the above one because it uses the conditional probabilities of the original network whereas the above one requires a preprocess computing those of the secondary structure. In terms of memory consumption, HUGIN outperforms our method and [SS90b]'s, since it uses only one message per edge whereas ours uses two messages per arc.

6.4 Proofs

Lemma 6.2 (Properties of the elimination tree) *Let $\mathcal{G}_T = (\mathcal{V}, \mathcal{E}_T)$ be a triangulated graph induced by an elimination sequence σ, and let $\mathcal{G}_J = (\mathcal{C}, \mathcal{E}_J)$ and $\mathcal{G}_E = (\mathcal{D}, \mathcal{A}_E)$ be a junction tree and the elimination tree of $\mathcal{G}_T$ respectively. Then the following properties hold:*

1. *For any $D_{\sigma(i)} \in \mathcal{D} \backslash \mathcal{C}$ and any arc $E = (D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}_E$, the content of E, is equal to $D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}$;*

2. *Let $D_{\sigma(i)}$ be a node in $\mathcal{D} \backslash \mathcal{C}$. Then $D_{\sigma(i)}$ has an ancestor in $\mathcal{C}$;*

3. *Let $D_{\sigma(i)}$ be a node in $\mathcal{D} \backslash \mathcal{C}$ that has only one parent in $\mathcal{G}_E$, say $D_{\sigma(j)}$. Then $D_{\sigma(i)} = D_{\sigma(j)} \backslash \{X_{\sigma(j)}\}$;*

4. *Let $D_{\sigma(i)}$ be a node in $\mathcal{D} \backslash \mathcal{C}$ that has multiple parents in $\mathcal{G}_E$, say $D_{\sigma(i_1)}, \ldots, D_{\sigma(i_p)}$. Then, there exists an index $j \in \{1, \ldots, p\}$ such that, for all $k \in \{1, \ldots, p\}$, $D_{\sigma(i)} \cap D_{\sigma(i_k)} \subseteq D_{\sigma(i)} \cap D_{\sigma(i_j)}$. Moreover $D_{\sigma(i)} = D_{\sigma(i_j)} \backslash \{X_{\sigma(i_j)}\}$.*

Proof of Proof of lemma 6.2.: 1/ We will prove property 1/ for all $D_{\sigma(i)} \in \mathcal{D}$. Let $E = (D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}_E$. Then, by definition, $j = \min\{k : X_{\sigma(k)} \in D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}\}$. Thus $X_{\sigma(j)}$ is the first node in $D_{\sigma(i)}$ to be eliminated after $X_{\sigma(i)}$ itself has been eliminated. When this happens, since no node in $D_{\sigma(i)} \backslash \{X_{\sigma(i)}, X_{\sigma(j)}\}$ has been removed yet and since all the nodes in this set are adjacent (because $D_{\sigma(i)}$ is a clique and only edges adjacent to eliminated nodes are removed), $D_{\sigma(j)} \supseteq D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}$. Moreover, $X_{\sigma(i)} \notin D_{\sigma(j)}$ because $X_{\sigma(j)}$ is eliminated after $X_{\sigma(i)}$. Hence $D_{\sigma(i)} \cap D_{\sigma(j)}$, the content of E, equals $D_{\sigma(i)} \backslash \{X_{\sigma(i)}\}$.
2/ Let $D_{\sigma(i)} \in \mathcal{D} \backslash \mathcal{C}$. Assume $D_{\sigma(i)}$ has no parent in $\mathcal{G}_E$. Let $\mathcal{D}'$ be the set of cliques in $\mathcal{D}$ containing $D_{\sigma(i)}$. $\mathcal{D}' \neq \emptyset$ because $D_{\sigma(i)} \in \mathcal{D} \backslash \mathcal{C}$. Consider any $D_{\sigma(j)} \in \mathcal{D}'$. Then $j < i$, else $X_{\sigma(i)} \notin D_{\sigma(j)}$. Since $D_{\sigma(i)}$ is not a child of $D_{\sigma(j)}$, $r = \min\{k : X_{\sigma(k)} \in D_{\sigma(j)} \backslash \{X_{\sigma(j)}\}\} < i$. $D_{\sigma(j)}$ being a clique, $X_{\sigma(r)}$ is adjacent to all the nodes in $D_{\sigma(j)}$, and *a fortiori* to all the nodes in $D_{\sigma(i)}$. Hence $D_{\sigma(i)} \subset D_{\sigma(r)}$. Consequently, if $D_{\sigma(i)}$ has no parent in $\mathcal{G}_E$, for any j such that $D_{\sigma(j)} \in \mathcal{D}'$, there exists r, $j < r < i$, such that $D_{\sigma(r)} \in \mathcal{D}'$ which, by induction, proves to be impossible since $\mathcal{D}'$ is a finite set. So any $D_{\sigma(i)} \in \mathcal{D} \backslash \mathcal{C}$ has at least one parent. $\mathcal{G}_E$ having a finite number of nodes and arcs, $D_{\sigma(i)}$ must have an ancestor in $\mathcal{C}$.
3/ Let $D_{\sigma(i)}$ be a node in $\mathcal{D} \backslash \mathcal{C}$ that has only one parent, say $D_{\sigma(j)}$, in $\mathcal{G}_E$. Assume that $D_{\sigma(i)} \not\subset D_{\sigma(j)}$. As for the proof of property 2, consider the set $\mathcal{D}'$ of all the cliques in $\mathcal{D}$ containing $D_{\sigma(i)}$. Then $D_{\sigma(j)} \notin \mathcal{D}'$. Now for any $D_{\sigma(k)} \in \mathcal{D}'$, as there exists no arc $(D_{\sigma(k)}, D_{\sigma(i)}) \in \mathcal{A}_E$, there exists an index $r > k$ such that $(D_{\sigma(k)}, D_{\sigma(r)}) \in \mathcal{A}_E$ and $D_{\sigma(i)} \subset D_{\sigma(r)}$ since $r < i$, which is again impossible because $\mathcal{D}'$ would be infinite.

Hence $D_{\sigma(i)} \subset D_{\sigma(j)}$. Since we already know that $D_{\sigma(i)} \supseteq D_{\sigma(j)} \backslash \{X_{\sigma(j)}\}$ by definition of the arcs in $\mathcal{A}_E$, property 3 obtains.

4/ Assume that there exists no such j. Then there exist two different parents of $D_{\sigma(i)}$, say $D_{\sigma(i_r)}$ and $D_{\sigma(i_s)}$, such that there exists no $i_k \neq i_r$ such that $D_{\sigma(i_r)} \cap D_{\sigma(i)} \subseteq D_{\sigma(i_k)} \cap D_{\sigma(i)}$ and no $i_k \neq i_s$ such that $D_{\sigma(i_s)} \cap D_{\sigma(i)} \subseteq D_{\sigma(i_k)} \cap D_{\sigma(i)}$, and of course $D_{\sigma(i_r)} \cap D_{\sigma(i)} \neq D_{\sigma(i_s)} \cap D_{\sigma(i)}$. By property 1, $D_{\sigma(i)} \supseteq S_{rs} = (D_{\sigma(i_r)} \cup D_{\sigma(i_s)}) \backslash \{X_{\sigma(i_r)}, X_{\sigma(i_s)}\}$. There exist other cliques containing S_{rs} in $\mathcal{D}$ since $D_{\sigma(i)} \in \mathcal{D} \backslash \mathcal{C}$. By the running intersection property, the trail between $D_{\sigma(i)}$ and such cliques passes through $D_{\sigma(i)}$'s child (because no parent of $D_{\sigma(i)}$ contains the whole set S_{rs}), hence these cliques cannot contain $X_{\sigma(i)}$. So $D_{\sigma(i)}$ is the only clique in $\mathcal{D}$ containing both $X_{\sigma(i)}$ and S_{rs}, and thus it belongs to $\mathcal{C}$, a contradiction. So there exists j such that, for all $k \in \{1, \ldots, p\}$, $D_{\sigma(i)} \cap D_{\sigma(i_k)} \subseteq D_{\sigma(i)} \cap D_{\sigma(i_j)}$.

By property 1, $D_{\sigma(i)} \cap D_{\sigma(i_j)} = D_{\sigma(i_j)} \backslash \{X_{\sigma(i_j)}\}$, so $D_{\sigma(i)} \supseteq D_{\sigma(i_j)} \backslash \{X_{\sigma(i_j)}\}$. Assume there exists a node $X_{\sigma(k)}$ in $D_{\sigma(i)}$ but not in $D_{\sigma(i_j)}$ then, by the preceding paragraph, no parent of $D_{\sigma(i)}$ can contain it. Again, since $D_{\sigma(i)} \in \mathcal{D} \backslash \mathcal{C}$, there exist other cliques that contain $D_{\sigma(i)}$, which is impossible since the trail between $D_{\sigma(i)}$ and such cliques passes through $D_{\sigma(i)}$'s child, preventing them to contain $X_{\sigma(i)}$. Hence $D_{\sigma(i)} = D_{\sigma(i_j)} \backslash \{X_{\sigma(i_j)}\}$. ♦

Proof of Proof of lemma 6.1.: In this proof, we will transform $\mathcal{G}_E$ into a junction tree $\mathcal{G}'_J$ satisfying the lemma's property. Each step of the transformation will preserve properties 1 to 4 of lemma 6.2. By [JJ94, corollary 1], all junction trees have the same cliques and the same separators (including multiplicity). Hence $\mathcal{G}_J$ and $\mathcal{G}_E$ will also satisfy the property.

Start from a graph $\mathcal{G}'_J = (\mathcal{D}', \mathcal{A}'_J) = \mathcal{G}_E$. Let

$$\Pi'_J = \frac{\prod_{D_{\sigma(i)} \in \mathcal{D}'} P(D_{\sigma(i)})}{\prod_{(D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}'_J} P(D_{\sigma(i)} \cap D_{\sigma(j)})}.$$

By definition of the elimination tree, $\mathcal{G}'_J$ has the running intersection property. Assume there exist some nodes in $\mathcal{D} \backslash \mathcal{C}$ having at least two parents in $\mathcal{A}'_J$. Then there exists a minimal index r such that $D_{\sigma(r)}$ has this property. By property 4 of lemma 6.2, there exists a parent $D_{\sigma(s)}$ such that $D_{\sigma(r)} = D_{\sigma(s)} \backslash \{X_{\sigma(s)}\}$. Select this parent and another one. Either they belong to $\mathcal{C}$ or, by property 2 of lemma 6.2, they have ancestors in $\mathcal{C}$. Hence, there exist two sequences of nodes $N_1 = \{D_{\sigma(i_1)}, D_{\sigma(i_2)}, \ldots, D_{\sigma(i_p)}\}$ and $N_2 = \{D_{\sigma(j_1)}, D_{\sigma(j_2)}, \ldots, D_{\sigma(j_q)}\}$ such that:

1. $D_{\sigma(i_1)}$ and $D_{\sigma(j_1)}$ belong to $\mathcal{C}$ and all the other nodes belong to $\mathcal{D} \backslash \mathcal{C}$;
2. $D_{\sigma(i_p)} = D_{\sigma(j_q)} = D_{\sigma(r)}$ and $D_{\sigma(j_{q-1})} = D_{\sigma(s)}$;
3. for all k, $(D_{\sigma(i_k)}, D_{\sigma(i_{k+1})}) \in \mathcal{A}'_J$ and $(D_{\sigma(j_k)}, D_{\sigma(j_{k+1})}) \in \mathcal{A}'_J$.

By construction, when $(D_{\sigma(j)}, D_{\sigma(k)}) \in \mathcal{A}'_J$, $D_{\sigma(k)}$ belonging to $\mathcal{D} \backslash \mathcal{C}$, then $j < k$. Hence, since among the cliques that have multiple parents, r has the lowest index,

no node in sequence $\{D_{\sigma(i_2)}, \ldots, D_{\sigma(i_{p-1})}\}$ has more than one parent. Consequently, removing all nodes $D_{\sigma(i_2)}, \ldots, D_{\sigma(i_{p-1})}$ as well as their outgoing arc, and replacing arc $(D_{\sigma(i_1)}, D_{\sigma(i_2)})$ by $(D_{\sigma(i_1)}, D_{\sigma(j_1)})$ does not change the connectedness of the graph. Moreover, all arcs $(D_{\sigma(j)}, D_{\sigma(k)}) \in \mathcal{A}'_J$ such that either $D_{\sigma(j)} \notin \mathcal{C}$ or $D_{\sigma(k)} \notin \mathcal{C}$, still satisfy $j < k$. If the new arc is labeled by $D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\}$, then the running intersection property is preserved. Indeed we know that cliques $D_{\sigma(i_2)}, \ldots, D_{\sigma(i_{p-1})}$ have only one parent so, applying recursively property 3 of lemma 6.2, $D_{\sigma(i_1)} \cap D_{\sigma(i_{p-1})} = D_{\sigma(i_{p-1})}$. Moreover, by property 1 of lemma 6.2, $D_{\sigma(i_{p-1})} \cap D_{\sigma(r)} = D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\}$ hence, by the running intersection property, $D_{\sigma(i_1)} \cap D_{\sigma(r)} = D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\}$. Similarly, by property 3 of lemma 6.2, $D_{\sigma(j_1)} \cap D_{\sigma(j_{q-1})} = D_{\sigma(j_{q-1})}$ and by property 4 of lemma 6.2, $D_{\sigma(j_{q-1})} \cap D_{\sigma(r)} = D_{\sigma(r)}$ so that, by the running intersection property, $D_{\sigma(j_1)} \cap D_{\sigma(r)} = D_{\sigma(r)} \supseteq D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\}$. Consequently, by the running intersection property, $D_{\sigma(i_1)} \cap D_{\sigma(j_1)} = D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\}$. Note that the above modification resulted in replacing product:

$$\prod_{k \in \{1,\ldots,p-1\}} \frac{P(D_{\sigma(i_k)}}{P(D_{\sigma(i_k)} \cap D_{\sigma(i_{k+1})})} = \prod_{k \in \{1,\ldots,p-2\}} \frac{P(D_{\sigma(i_k)})}{P(D_{\sigma(i_{k+1})})} \times \frac{P(D_{\sigma(i_{p-1})})}{P(D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\})}$$

by

$$\frac{P(D_{\sigma(i_1)})}{P(D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\})}$$

in Π'_J, or in other words in removing all the $P(D_{\sigma(i_k)})/P(D_{\sigma(i_k)})$, $k \in \{2, \ldots, p-1\}$. Note also that if, after the modification in $\mathcal{G}'_J$, $D_{\sigma(r)}$ has only one parent left, then property 3 of lemma 6.2 is preserved.

Applying recursively the above transformations guarantees that $\mathcal{G}'_J$ has no more clique in $\mathcal{D} \backslash \mathcal{C}$ having more than one parent. Thus by property 2 of lemma 6.2, all cliques in $\mathcal{D} \backslash \mathcal{C}$ have exactly one parent in $\mathcal{G}'_J$. Assume now that there exist two cliques $C_{\sigma(i)}$ and $C_{\sigma(j)}$ in $\mathcal{C}$, $i < j$, that are joined by a trail containing only cliques in $\mathcal{D} \backslash \mathcal{C}$, say $N = \{D_{\sigma(i_1)} = C_{\sigma(i)}, D_{\sigma(i_2)}, \ldots, D_{\sigma(i_p)} = C_{\sigma(j)}\}$. Then, since all cliques in $\mathcal{D} \backslash \mathcal{C}$ have exactly one parent, this trail is a path from $C_{\sigma(i)}$ to $C_{\sigma(j)}$. But then, as in the above paragraph, removing all the nodes in $\{D_{\sigma(i_2)}, \ldots, D_{\sigma(i_{p-1})}\}$ as well as their outgoing arc, and replacing arc $(C_{\sigma(i)}, D_{\sigma(i_2)})$ by an arc $(C_{\sigma(i)}, C_{\sigma(j)})$ labeled with $D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\}$ keeps the running intersection property (see property 3 of lemma 6.2) and modifies expression Π'_J replacing

$$\prod_{k \in \{1,\ldots,p-1\}} \frac{P(D_{\sigma(i_k)})}{P(D_{\sigma(i_k)} \cap D_{\sigma(i_{k+1})})} = \prod_{k \in \{1,\ldots,p-2\}} \frac{P(D_{\sigma(i_k)})}{P(D_{\sigma(i_{k+1})})} \times \frac{P(D_{\sigma(i_{p-1})})}{P(D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\})}$$

by expression

$$\frac{P(D_{\sigma(i_1)})}{P(D_{\sigma(i_{p-1})} \backslash \{X_{\sigma(i_{p-1})}\})},$$

or in other words it removes all $P(D_{\sigma(i_k)})/P(D_{\sigma(i_k)})$'s, $k \in \{2, \ldots, p-1\}$. Apply recursively this paragraph to all such pairs of cliques $(C_{\sigma(i)}, C_{\sigma(j)})$.

Now, assume there exists a clique in $\mathcal{C}$, say $C_{\sigma(i)}$, the outgoing arc of which points toward a clique in $\mathcal{D}\backslash\mathcal{C}$. Then all the descendants of $C_{\sigma(i)}$ belong to $\mathcal{D}\backslash\mathcal{C}$ (by the preceding paragraph) and form a path (since no node in $\mathcal{D}\backslash\mathcal{C}$ has more than one parent). Let $N = \{D_{\sigma(i_1)} = C_{\sigma(i)}, D_{\sigma(i_2)}, \ldots, D_{\sigma(i_p)}\}$ be this path. Then removing all nodes in $\{D_{\sigma(i_2)}, \ldots, D_{\sigma(i_p)}\}$ as well as their outgoing arc (note that $D_{\sigma(i_p)}$ is the only clique without outgoing arc) keeps the running intersection property. Moreover, the modification resulting in Π'_J corresponds to replacing expression

$$\frac{\prod_{k\in\{1,\ldots,p\}} P(D_{\sigma(i_k)})}{\prod_{k\in\{1,\ldots,p-1\}} P(D_{\sigma(i_k)})}$$

by $P(D_{\sigma(i_1)})$. Applying recursively such modifications, there remains a graph $\mathcal{G}'_J$ the nodes of which are those in $\mathcal{C}$ (by property 2 of lemma 6.2) and having the running intersection property, a junction tree. Hence the lemma holds. ♦

Proof of Proof of corollary 6.1.: It is well known that expression Π_J in lemma 6.1 factorizes the joint probability on $\mathcal{V}$. By lemma 6.1 the factorization given in corollary 6.1 thus obtains. Since the expressions of Π_J and Π_E are identical, except for expressions in Π_E that appear both at the numerator and the denominator, conditional independencies between any pair of random variables in $\mathcal{V}$ are similarly captured by $\mathcal{G}_J$ and $\mathcal{G}_E$. ♦

Proof of Proof of corollary 6.2.: By corollary 6.1, the joint probability $P(\mathcal{V})$ factorizes as $P(\mathcal{V}) = \left[\prod_{D_{\sigma(i)}\in\mathcal{D}} P(D_{\sigma(i)})\right] / \left[\prod_{(D_{\sigma(i)},D_{\sigma(j)})\in\mathcal{A}_E} P(D_{\sigma(i)} \cap D_{\sigma(j)})\right]$. Note that, in the elimination tree, a clique $D_{\sigma(i)}$ has an outgoing arc if and only if it contains at least two nodes. Hence, if $\mathcal{D}^1 = \{D_{\sigma(i)} : D_{\sigma(i)} = \{X_{\sigma(i)}\}\}$ and $\mathcal{D}^2 = \mathcal{D}\backslash\mathcal{D}^1$, then

$$P(\mathcal{V}) = \prod_{D_{\sigma(i)}\in\mathcal{D}^1} P(X_{\sigma(i)}) \times \prod_{D_{\sigma(i)}\in\mathcal{D}^2} \frac{P(D_{\sigma(i)})}{P(D_{\sigma(i)} \cap D_{\sigma(j)})},$$

where, in the denominator of the right product, $P(D_{\sigma(i)} \cap D_{\sigma(j)})$ is the probability of the label of edge $(D_{\sigma(i)}, D_{\sigma(j)}) \in \mathcal{A}_E$. But, by property 1 of lemma 6.2, $D_{\sigma(i)} \cap D_{\sigma(j)} = D_{\sigma(i)}\backslash\{X_{\sigma(i)}\}$. Hence:

$$P(\mathcal{V}) = \prod_{D_{\sigma(i)}\in\mathcal{D}^1} P(X_{\sigma(i)}) \times \prod_{D_{\sigma(i)}\in\mathcal{D}^2} P(X_{\sigma(i)}|D_{\sigma(i)}\backslash\{X_{\sigma(i)}\}),$$

which corresponds precisely to the factorization of the joint probability represented by DAG $\mathcal{G}'$. ♦

Proof of Proof of proposition 6.1: Assume that $\mathcal{V}_{\sigma(i_0)} \neq \emptyset$ and that there exists a node $X_{\sigma(j)} \in \mathcal{V}_{\sigma(i_0)}$ such that arc $(X_{\sigma(i_0)}, X_{\sigma(j)})$ cannot be reversed. By theorem 6.1, there exists a path in the graph, say $\{Z_1 = X_{\sigma(i_0)}, Z_2, \ldots, Z_p = X_{\sigma(j)}\}$, different from arc $(X_{\sigma(i_0)}, X_{\sigma(j)})$, going from $X_{\sigma(i_0)}$ to $X_{\sigma(j)}$. Note that $j > i$, else by construction of

$\mathcal{G}_T$ (see corollary 6.2) $(X_{\sigma(i_0)}, X_{\sigma(j)})$ would belong to $\mathcal{A}_T$ since $X_{\sigma(j)}$ would be eliminated before $X_{\sigma(i_0)}$. Similarly, the existence of arc (Z_{p-1}, Z_p) implies that Z_{p-1} is eliminated after $X_{\sigma(i_0)}$ else it would belong to $\{X_{\sigma(k)} : k < i_0\}$ but arc (Z_{p-1}, Z_p) would not belong to $\mathcal{A}_T$, a contradiction. By induction, all the Z_i's, $i > 1$, are eliminated after $X_{\sigma(i_0)}$.

Since $\mathcal{G}$ is a DAG, there exists a node in $\mathcal{V}_{\sigma(i_0)}$, say Z, without ancestor in $\mathcal{V}_{\sigma(i_0)}$. If there existed a path from $X_{\sigma(i_0)}$ to Z different from arc $(X_{\sigma(i_0)}, Z)$, the second node of this path, say Z_2, would also belong to $\mathcal{V}_{\sigma(i_0)}$ since, by the preceding paragraph, it would be eliminated after $X_{\sigma(i_0)}$ and arc $(X_{\sigma(i_0)}, Z_2)$ would belong to $\mathcal{A}\backslash\mathcal{A}_T$. This leads to a contradiction since Z_2 would be one of Z's ancestors. Hence theorem 6.1 can be applied and arc $(X_{\sigma(i_0)}, Z)$ be reversed.

Now let us show that arcs added by arc $(X_{\sigma(i_0)}, Z)$'s reversal correspond to edges in $\mathcal{E}$. These are illustrated in Figure 6.7: $X_{\sigma(i_0)}$'s (resp. Z's) parents must become Z's (resp. $X_{\sigma(i_0)}$'s) parents as well. Since before reversal $X_{\sigma(i_0)}$ and W_k's are parents

(a) before reversal (b) after reversal

Figure 6.7: The arcs added to the graph by arc (X, Y)'s reversal.

of Z, the moralization phase leading to $\mathcal{G}_E$ shall add edges in $\mathcal{E}$ between all these nodes. Hence arcs $(W_k, X_{\sigma(i_0)})$ added by arc reversal correspond to edges in $\mathcal{E}$. Z, U_i's, V_j's and W_k's are known to be eliminated after $X_{\sigma(i_0)}$ (see the construction of $\mathcal{A}'$ in corollary 6.2). But during the triangulation process, when $X_{\sigma(i_0)}$ is eliminated, edges are added between all of its neighbors so as to form a clique. Hence $\mathcal{E}$ contains edges between the U_i's and Z. Hence all the arcs added by $(X_{\sigma(i_0)}, Z)$'s reversal correspond to edges in $\mathcal{E}$. After arc reversal, $X_{\sigma(i_0)}$ and Z have new parents and we must check that triangulating again will result in $\mathcal{G}_E$. To do this, it is sufficient to show that edges obtained by marrying $X_{\sigma(i_0)}$'s parents and Z's parents already belong to $\mathcal{E}$. As the U_i's and V_j's (resp. V_j's and W_k's) were already $X_{\sigma(i_0)}$'s (resp. Z's) parents, by moralization these parents were already married before reversal. We had already seen that $X_{\sigma(i_0)}$ and the W_k's were among Z's parents before reversal, hence by moralization, they were married in $\mathcal{E}$. During triangulation, when node $X_{\sigma(i_0)}$ is eliminated, all of its neighbors are linked by edges. As the U_i's are $X_{\sigma(i_0)}$'s parents, they are also neighbors. Since edges were added by Z's moralization between $X_{\sigma(i_0)}$ and the W_k's, the latter are also $X_{\sigma(i_0)}$'s neighbors. Hence $X_{\sigma(i_0)}$'s elimination adds edges in $\mathcal{E}$ between all of the U_i's and W_k's. Consequently, triangulating again the graph obtained from $\mathcal{G}$ after arc reversal results again in $\mathcal{G}_E$ et $\mathcal{G}_T$. ♦

Proof of Proof of proposition 6.2: When function `elimination(·)` is called on node $X_{\sigma(1)}$, if $\mathcal{V}' = \emptyset$, all the arcs adjacent to $X_{\sigma(1)}$ are properly directed and no arc reversal need be performed. If, on the contrary, $\mathcal{V}' = \{Y_1, \ldots, Y_p\} \neq \emptyset$, by step 2 of

the function it is possible to select a Y_i and to apply proposition 6.1, hence resulting in a graph the triangulation of which is still $\mathcal{G}_T$. Note that the arcs adjacent to $X_{\sigma(1)}$ added to $\mathcal{A}$ by this process are directed toward $X_{\sigma(1)}$ and thus their head cannot belong to $\mathcal{V}'$. Hence steps 1–5 remove by induction all the improper arcs and they result in $\mathcal{V}' = \emptyset$. Moreover, proposition 6.1 guarantees that triangulating the modified graph leads to $\mathcal{G}_T$.

When steps 6–9 are performed, only arcs of type $(Y, X_{\sigma(1)})$ are added. Since these arcs already belonged to $\mathcal{A}_T$, they corresponded to edges adjacent to $X_{\sigma(1)}$ in the undirected triangulated graph, say $\mathcal{G}_E$. Therefore, when $X_{\sigma(1)}$ has been eliminated, edges between all these Y's and the other $X_{\sigma(1)}$'s neighbors have been added. But the arc additions by steps 6–9 has precisely this effect by moralization. So triangulating again after their addition results again in $\mathcal{G}_E$, hence in $\mathcal{G}_T$.

The above two paragraphs can be applied to $X_{\sigma(2)}$, $X_{\sigma(3)}$, etc. By induction, it can be concluded that applying steps 1–9 and triangulating results in $\mathcal{G}_T$. But steps 1–5 ensure that arcs adjacent to each of the nodes in the modified version of graph $\mathcal{G}$ are directed as those in $\mathcal{G}_T$, and steps 6–9 guarantee that all the arcs belonging to $\mathcal{G}_T$ also belong to the modified version of graph $\mathcal{G}$. Hence, the algorithm maps $\mathcal{G}$ into $\mathcal{G}_T$. ♦

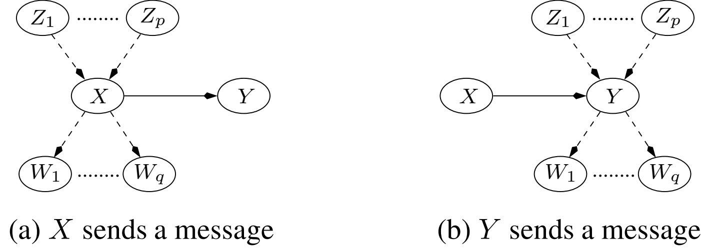
Proof of Proof of proposition 6.3: By proposition 6.2, $\mathcal{G}'$ represents a decomposition of the joint probability compatible with that defined by P. Hence if the cutset algorithm removes all cycles, *Collect-Evidence* and *Distribute-Evidence*, or equivalently Faÿ and Jaffray's algorithm, will provide exact computations.

By corollary 6.2 all the arcs in $\mathcal{A}'$ adjacent to $X_{\sigma(1)}$, say $(X_{\sigma(i_1)}, X_{\sigma(1)}), \dots, (X_{\sigma(i_k)}, X_{\sigma(1)})$, are directed toward $X_{\sigma(1)}$. Since k is a finite number, there exists in $\{i_1, \dots, i_k\}$ a smallest index. Without loss of generality, assume it is i_1. By triangulation, all of $X_{\sigma(1)}$'s neighbors are linked together to form a clique. Hence between every pair of variables $(X_{\sigma(i_j)}, X_{\sigma(i_l)})$, $j, l \in \{2, \dots, k\}$, there exists an arc in $\mathcal{A}'$. So there exist $k-1$ triangles $X_{\sigma(i_1)}X_{\sigma(i_j)}X_{\sigma(1)}$, and cutting arcs $(X_{\sigma(i_j)}, X_{\sigma(1)})$ removes all cycles passing through $X_{\sigma(1)}$ (since there remains only one arc adjacent to $X_{\sigma(1)}$: $(X_{\sigma(i_1)}, X_{\sigma(1)})$). Let us now eliminate cycles passing through $X_{\sigma(2)}$. There exists no arc $(X_{\sigma(1)}, X_{\sigma(2)})$ in $\mathcal{A}'$ because $X_{\sigma(1)}$ has no child. Hence the process applied on $X_{\sigma(1)}$ can also be applied to $X_{\sigma(2)}$. By induction on the nodes ordered according to elimination sequence σ, all cycles are removed from the graph. Hence Faÿ and Jaffray's algorithm will provide exact computations.

Let us show that if every arc (X, Y) is labeled by X before any cutset, then for any node $X_{\sigma(k)}$, the cutset algorithm of the proposition changes only the label of the remaining arc directed toward $X_{\sigma(k)}$ and, moreover, this label becomes equal to the set of all $X_{\sigma(k)}$'s parents. Triangle $X_{\sigma(i)}X_{\sigma(j)}X_{\sigma(k)}$ is cut on arc $(X_{\sigma(i)}, X_{\sigma(k)})$ only if $j < i$. Hence $X_{\sigma(j)}$ is eliminated before $X_{\sigma(i)}$ and accordingly there exists an arc $(X_{\sigma(i)}, X_{\sigma(j)})$. In other words, the triangle is constituted by arcs $(X_{\sigma(i)}, X_{\sigma(j)})$, $(X_{\sigma(i)}, X_{\sigma(k)})$ and $(X_{\sigma(j)}, X_{\sigma(k)})$. Consequently, cutting arc $(X_{\sigma(i)}, X_{\sigma(k)})$ adds only label $X_{\sigma(i)}$ to arc $(X_{\sigma(j)}, X_{\sigma(k)})$, i.e., to the only remaining arc adjacent to $X_{\sigma(k)}$ after all cutsets. Now when all triangles adjacent to a given node have been cut, there remains no cycle passing through it and *a fortiori* through its only remaining adjacent

arc. Hence future cuts cannot modify the label of this arc.

Let us show by induction that π–λ messages are equal to those described at the end of the proposition. Induction basis: assume that, after cutsetting, node X has only one neighbor left, say Y (see Figure 6.8.a). X cannot have other parents $Z_1, \ldots, Z_p$ in $\mathcal{G}'$



(a) X sends a message (b) Y sends a message

Figure 6.8: Induction bases.

since the cutset algorithm always leaves an adjacent arc. If X received an evidence e_X, then $e^+_{XY} = e_X$ else $e^+_{XY} = e_X = \emptyset$. By the labeled formulas on page 41, message $\pi_Y(X)$ sent by X is equal to $P(X, e_X)$ (no product is computed since all the arcs adjacent to X except (X, Y) have been removed). But $C_{XY} = X$, else there would exist $Z \neq X$ belonging to C_{XY} and, by the cutset algorithm, there would also exist a triangle ZXY where Z is parent of both X and Y, a contradiction since X has no parent in $\mathcal{G}'$. Consequently, $\pi_Y(X) = P(C_{XY}, e^+_{XY})$.

Consider now a node, say Y, having only one parent left after cutset (see Figure 6.8.b). By the cutset algorithm, Y cannot have children $W_1, \ldots, W_q$ in $\mathcal{G}'$ because this algorithm always leaves an adjacent arc. Again, the existence of arc (X, Y) and the removal of arcs (Z_i, Y) imply that node X is, among Y's parents, the first one to be eliminated during the triangulation process. Hence, by the third paragraph of this proof, arc (X, Y)'s label is equal to the set of Y's parents, i.e., $C_{XY} = \{X, Z_1, \ldots, Z_p\}$. The labeled formulas on page 41 show that the message sent by Y is equal to

$$\sum_{Z \in \mathcal{V} \setminus C_{XY}} P(Y, e^-_{XY} | \mathrm{Pa}(Y))$$

(no product is performed since only one arc remains). Consequently,

$$\lambda_Y(X) = \sum_Y P(Y, e^-_{XY} | X, Z_1, \ldots, Z_p) = P(e^-_{XY} | X, Z_1, \ldots, Z_p),$$

and $\lambda_Y(X) = P(e^-_{XY} | C_{XY})$.

Let us study the general case of the induction (Figure 6.9). Arcs (Z_i, X)'s have been removed hence, during the triangulation process, nodes Z_i's have been eliminated after X. For the same reason, since X is one of V_i's parents, all the other arcs adjacent to V_i and outgoing from V_i's parents in $\mathcal{G}'$ have been removed using triangles passing through X. But this is possible only if the V_i's are eliminated before X.

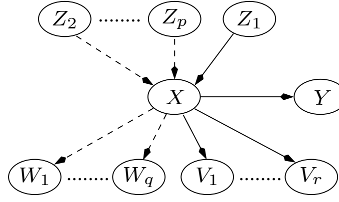


Figure 6.9: General case for the induction.

By induction hypothesis, $\lambda_X(V_i) = P(e^-_{XV_i}|C_{XV_i})$, $\forall\ i$. X is one of V_i's parents, hence all the other arcs ingoing V_i in graph $\mathcal{G}'$ have been cut and, by the cutset algorithm, this is possible only if these arcs are outgoing from parents of both X and V_i. Consequently, labels C_{XV_i} are subsets of $\{X, Z_1, \ldots, Z_p\}$. If there exists $Z_j \notin C_{XV_i}$, then Z_j is independent of V_i conditionally to C_{XV_i}. Indeed, in the contrary, there would exist a trail $\{U_1, \ldots, U_k\}$ between Z_j and V_i not passing through any variable in C_{XV_i}. But by d-connection[1], this trail cannot have converging arcs. Hence, since Z_j cannot be V_i's descendant (else there would exist a directed cycle), U_{k-1} must be one of V_i's parent. Thus it is either X, but X belongs to C_{XV_i}, or a parent of V_i such that arc (U_{k-1}, V_i) has been cut, which implies that $U_{k-1} \in C_{XV_i}$. In both cases, Z_j would be independent of V_i conditionally to C_{XV_i}, a contradiction. Now if Z_j is independent of V_i conditionally to C_{XV_i}, it is also independent of $e^-_{XV_i}$ conditionally to C_{XV_i}, else there would exist a d-connecting trail between Z_j and a node X_k having received evidence in $e^-_{XV_i}$, and a d-connecting trail between X_k and V_i, the union of these two trails being a d-connecting trail between Z_j and V_i, a contradiction. In conclusion, $P(e^-_{XV_i}|C_{XV_i}) = P(e^-_{XV_i}|X, Z_1, \ldots, Z_p)$.

By construction of the triangulated DAG in corollary 6.2, $X_{\sigma(1)}$ has only parents, and after cutsetting only one remains, the label of which is equal to $X_{\sigma(1)}$'s parents set, or equivalently to the separator content of the outgoing arc of $D_{\sigma(1)}$. Once $X_{\sigma(1)}$, has been removed, $X_{\sigma(2)}$ is in a similar case. Therefore, we can show by induction that each label on the arc ingoing of every node $X_{\sigma(i)}$ of $\mathcal{G}'$ (after cutsetting) corresponds to the content of the outgoing arc of $D_{\sigma(i)}$ in the elimination tree. Therefore the remaining arcs in the cutsetted version of $\mathcal{G}'$ are one-to-one with arcs in the elimination tree as well as their adjacency properties (two arcs are adjacent in the former if and only if they are adjacent as well in the latter).

A well known property of elimination trees (in fact of joint trees) is that the variables of two cliques, say C_1 and C_2, are independent if all the variables of some separator(s) or clique(s) on trail between C_1 and C_2 are instantiated. By preceding paragraph, D_{V_i} and D_{V_j}, the cliques created when V_i and V_j are eliminated respectively, are linked in the elimination tree by trail $\{D_{V_i}, D_X, D_{V_j}\}$. So the running intersection property ensures that, for all X_k, X_l such that X_k (resp. X_l) is on V_i's side of arc (X, V_i) (resp.

[1]Two variables, X and Y, are probabilistically dependent if and only if they are d-connected, i.e., there exists a trail $\{A_1, \ldots, A_k\}$ such that $A_1 = X$, $A_k = Y$, and such that along the trail, nodes A_l having converging arcs, e.g., arcs (A_{l-1}, A_l) and (A_{l+1}, A_l), either received evidence or have descendants with evidence, and the other nodes along the trail are not instantiated.

$(X, V_j))$ in the cutsetted version of $\mathcal{G}'$, the paths between cliques containing X_k and those contain X_l must pass through D_{V_i} and D_{V_j}. Consequently, X_k is independent of X_l conditionally to D_X and, by the preceding paragraphs, $D_X = \{X, Z_1, \ldots, Z_p\}$. But by hypothesis, all evidence e_{X_k}'s are independent of the rest of the network conditionally to X_k. So $e^-_{XV_i}$ is independent of $e^-_{XV_j}$ conditionally to $X, Z_1, \ldots, Z_p$ and:

$$\prod_{i=1}^{r} \lambda_X(V_i) = P\left(\bigcup_{i=1}^{r} e^-_{XV_i} | X, Z_1, \ldots, Z_p\right).$$

(Z_1, X) is the only remaining arc after cutsetting the arcs outgoing from X's parents. This implies that all the other (Z_i, X)'s have been cut within triangles $Z_i Z_1 X$. Consequently, $\{Z_1, \ldots, Z_p\} \subseteq C_{Z_1 X}$. But $C_{Z_1 X}$ cannot contain any other variables since those would be among X's parents (by the cutset algorithm). So $\pi_{Z_1}(X) = P(e^+_{Z_1 X}, C_{Z_1 X}) = P(e^+_{Z_1 X}, Z_1, \ldots, Z_p)$. For reasons similar to those of the preceding paragraph, $e^+_{Z_1 X}$ is independent of X and of e_X conditionally to $\{Z_1, \ldots, Z_p\}$. Consequently, $P(X, e_X | \mathrm{Pa}(X)) \pi_{Z_1}(X) = P(X, e_X, e^+_{Z_1 X} | Z_1, \ldots, Z_p)$. Similarly, the $e^-_{XV_i}$'s are independent from e_X and $e^+_{Z_1 X}$ conditionally to $\{X, Z_1, \ldots, Z_p\}$. Consequently,

$$P(X, e_X | \mathrm{Pa}(X)) \times \pi_X(Z_1) \times \prod_{j=1}^{r} \lambda_{V_j}(X) = P(e^+_{XY}, X, Z_1, \ldots, Z_p).$$

But by symmetry between Y and the V_i's, $C_{XY} \subseteq \{X, Z_1, \ldots, Z_p\}$. So, if we sum $P(e^+_{XY}, X, Z_1, \ldots, Z_p)$ over all the variables not belonging to C_{XY}, we get $P(e^+_{XY}, C_{XY})$, which is precisely the message described in the proposition. The proof is similar for upward messages.

To complete this proof, there remains to show that the computational complexity of the propagation algorithm of the proposition is at most equal to that of HUGIN. The latter is equal to the sum of the complexities of the all the absorptions performed. The complexity of the absorption of a clique C_j from C_i (see Figure 4.8) is equal to $|C_i| + |C_j|$, where $|C|$ refers to the product of the number of possible values of all the random variables in C.

The complexity of HUGIN is greater than or equal to that of Jensen's algorithm applied on the elimination tree. Indeed, in the second paragraph of the proof of lemma 6.1, the transformation applied to the elimination tree consists in removing a sequence of nodes $N_1 = \{D_{\sigma(i_2)}, \ldots, D_{\sigma(i_{p-1})}\}$ as well as their outgoing arcs and replacing arc $(D_{\sigma(i_1)}, D_{\sigma(i_2)})$ by $(D_{\sigma(i_1)}, D_{\sigma(j_1)})$. The corresponding impact in the absorptions complexity is a change from $|D_{\sigma(i_1)}| + |D_{\sigma(i_p)}| + 2\sum_{j=2}^{p-1} |D_{\sigma(i_j)}|$ to $|D_{\sigma(i_1)}| + |D_{\sigma(j_1)}|$. Now, as for all j's, $2 \leq j \leq p-1$, $D_{\sigma(i_j)} \in \mathcal{D}$ and has only one parent, by property 3 of lemma 6.2, $D_{\sigma(i_j)} = D_{\sigma(i_{j-1})} \backslash \{X_{\sigma(i_{j-1})}\}$. Since all nodes X_k's have at least 2 possible values, $|D_{\sigma(i_j)}| \leq |D_{\sigma(i_{j-1})}|/2$, so that $2\sum_{j=2}^{p-1} |D_{\sigma(i_j)}| \leq 2|D_{\sigma(i_1)}|$. Moreover $|D_{\sigma(j_1)}| > |D_{\sigma(i_p)}|$, so the complexity before transformation is less than or equal to $3|D_{\sigma(i_1)}| + |D_{\sigma(j_1)}|$, or equivalently to $|D_{\sigma(i_1)}| + |D_{\sigma(j_1)}|$. Now, each time the transformation is applied, $D_{\sigma(i_1)}$ is different from preceding $D_{\sigma(i_1)}$'s because after each transformation, $D_{\sigma(i_1)}$ is linked to a node in $\mathcal{C}$. Hence the set of transformations applied in

the second paragraph of the proof of lemma 6.1 increase the propagation complexity. For the same reason, the transformations applied in the third and fourth paragraphs of the proof of lemma 6.1 keep the overall complexity unchanged, so that the complexity of Jensen's algorithm applied in the elimination tree is at most equal to that applied in the junction tree.

In the propagation algorithm of proposition 6.3 (and on Figure 6.5), π–λ messages are equal to:

$$\pi_Y(X) = \sum_{Z \in \mathcal{V} \backslash C_{XY}} \left[P(X, e_X | \mathrm{Pa}(X)) P(e^+_{U_1 X}, C_{U_1 X}) \prod_{j=1}^{m} P(e^-_{XY_j} | C_{XY_j}) \right],$$

$$\lambda_Y(X) = \sum_{Z \in \mathcal{V} \backslash C_{XY}} \left[P(Y, e_Y | \mathrm{Pa}(Y)) \prod_{j=1}^{p} P(e^-_{YS_j} | C_{YS_j}) \right],$$

since nodes of $\mathcal{G}'$ have at most one parent. By the preceding paragraphs, labels C_{XY} and C_{XY_j} are subsets of $\{X, \mathrm{Pa}(X)\}$. Hence message $\pi_Y(X)$ is equal to

$$\sum_{Z \in \mathcal{V} \backslash C_{XY}} [P(X, e_X | \mathrm{Pa}(X)) \prod_{j=1}^{m} P(e^-_{XY_j} | C_{XY_j}) \sum_{Z \in \mathcal{V} \backslash \{X, \mathrm{Pa}(X)\}} P(e^+_{U_1 X}, C_{U_1 X})].$$

Again by the preceding paragraphs, labels C_{AB} of every arc (A, B) in $\mathcal{G}'$ are equal to the set of B's parents, hence to cliques of the elimination tree. Now note that the complexity of computing $\pi_Y(X)$ as defined above would be equal to that of the absorption by a clique $\{X, \mathrm{Pa}(X)\}$ from cliques $C_{U_1 X}$ and C_{XY_j}'s. Note also that clique $\{X, \mathrm{Pa}(X)\}$ is equal in the elimination tree to D_X. Similarly, message $\lambda_Y(X)$ corresponds to the absorption by clique $\{Y, \mathrm{Pa}(Y)\}$ from cliques C_{YS_j}'s. Finally by the preceding paragraphs, if all these cliques are adjacent in the triangulated DAG, they are also adjacent in the elimination tree. Hence the overall complexity of the collect-distribute phases in the triangulated DAG is equal to that in the elimination tree. By transitivity, it is also at most equal to the complexity in HUGIN. ♦

Chapter 7

Binary join trees within Pearl's algorithm

It is well known that Shafer-Shenoy's algorithm is competitive with Jensen's method only when it is applied in binary join trees, that is, although these algorithms have the same computational complexity, in terms of the number of arithmetic operations they perform, Jensen's algorithm usually outperforms that of Shafer-Shenoy. Only when applied within binary join trees is the latter as fast as Jensen's algorithm. As the variation of Pearl's algorithm we introduced in the preceding chapter is strongly related to Shafer-Shenoy's algorithm, it suffers from the same property : unless it can perform computations as if it were in a binary join tree, it will require more arithmetic operations than Jensen. In this chapter we will show how propagation in binary join trees can be effectively reproduced using a stack to store temporary computations.

For this purpose, we first present an example which enables us to highlight the differences between propagating in a general join tree and in a binary one. Then we show how these can be used to reduce the number of computations performed by Pearl. Quite naturally, it is shown that this can be achieved using some stacks to store temporary computations.

7.1 Example of a propagation in a BJT

7.1.1 Computations in a nonbinary join tree

Consider the Bayesian network of Figure 7.1. The graph resulting from moralization and triangulation using elimination sequence A, F, G, I, J, K, H, D, B, C, E, is shown in Figure 7.2. This, in turn, leads to the join tree of Figure 7.3. To understand precisely the difference between propagating in a classical join tree and in a binary join tree, we shall first apply Shafer-Shenoy's algorithm in the join tree of Figure 7.3, and then in a binary join tree. Using clique $HCDE$ as the root, the former computations are described in Tables 7.1 and 7.2. These are summarized in Figure 7.4.

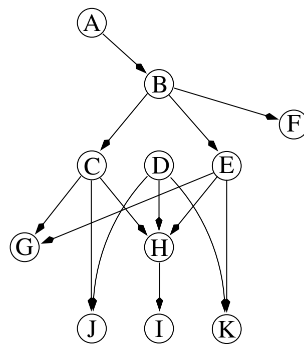


Figure 7.1: A Bayesian network.

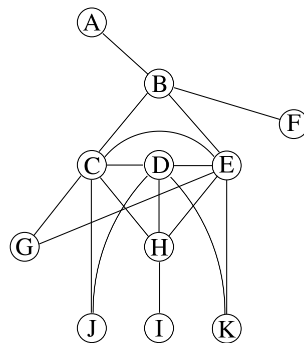


Figure 7.2: The Markov network obtained after triangulation.

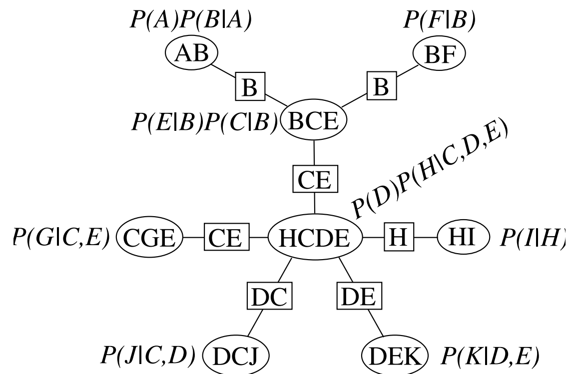


Figure 7.3: A compatible join tree.

Remark that the computations of messages ❶ to ❹ is very inefficient because the same products are performed several times. To avoid these redundancies, we shall use a binary join tree instead of the tree of Figure 7.3.

Sending clique	computation	message
AB	$\sum_A P(A)P(B\|A) = P(B)$	①
BF	$\sum_F P(F\|B) = \mathbb{1}_B$	②
BCE	$\sum_B P(E\|B)P(C\|B)P(B)\mathbb{1}_B = P(C,E)$	③
CGE	$\sum_G P(G\|C,E) = \mathbb{1}_{CE}$	④
DCJ	$\sum_J P(J\|C,D) = \mathbb{1}_{CD}$	⑤
DEK	$\sum_K P(K\|D,E) = \mathbb{1}_{DE}$	⑥
HI	$\sum_I P(I\|H) = \mathbb{1}_H$	⑦

Table 7.1: The inward pass performed by Shafer-Shenoy in the nonbinary join tree.

Receiving clique	computation	message
HI	$\sum_{C,D,E} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{CE}\mathbb{1}_{DC}\mathbb{1}_{DE} = P(H)$	❶
DEK	$\sum_{C,H} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{CE}\mathbb{1}_{DC}\mathbb{1}_H = P(D,E)$	❷
DCJ	$\sum_{H,E} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{CE}\mathbb{1}_{DE}\mathbb{1}_H = P(C,D)$	❸
CGE	$\sum_{H,D} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{DC}\mathbb{1}_{DE}\mathbb{1}_H = P(C,E)$	❹
BCE	$\sum_{H,D} P(D)P(H\|C,D,E)\mathbb{1}_{CE}\mathbb{1}_{DC}\mathbb{1}_{DE}\mathbb{1}_H = \mathbb{1}_{CE}$	❺
BF	$\sum_{C,E} P(E\|B)P(C\|B)P(B)\mathbb{1}_{CE} = P(B)$	❻
AB	$\sum_{C,E} P(E\|B)P(C\|B)\mathbb{1}_B\mathbb{1}_{CE} = \mathbb{1}_B$	❼

Table 7.2: The outward pass performed by Shafer-Shenoy in the nonbinary join tree.

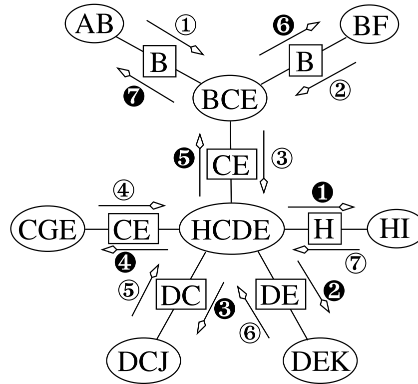


Figure 7.4: Messages in the nonbinary join tree.

7.1.2 Computations in a binary join tree

A binary join tree corresponding to the join tree of Figure 7.3 may be that of Figure 7.5. Observing that i) node $CEBG$ has been added only to factorize computations involving cliques BCE, $HCDE$ and CGE; and ii) that only one of these three cliques contains B (i.e., clique BCE) and only one contains G (i.e., clique CGE), node $CEBG$ can be advantageously substituted by the smaller node CE without altering the correctness of the propagation algorithm. Similarly, node $CDEKJ$ can also be replaced by CDE. Such substitutions lead to the more efficient binary join tree of Figure 7.6 that we will

use for our computations. Now, if clique $HCDE$ is chosen as the root for Shafer-Shenoy's algorithm, then the computations performed will be those of Tables 7.3 and 7.4. Note that, in this new join tree, the redundancies in the computations mentioned above have been avoided.

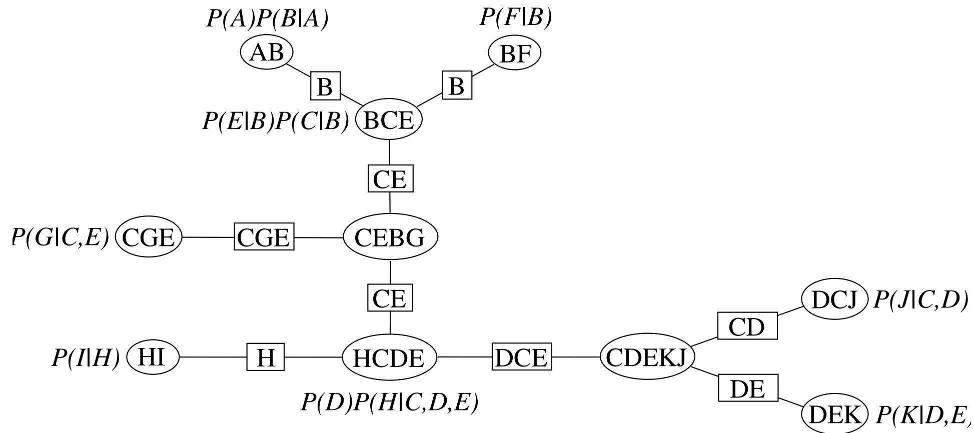


Figure 7.5: A compatible binary join tree.

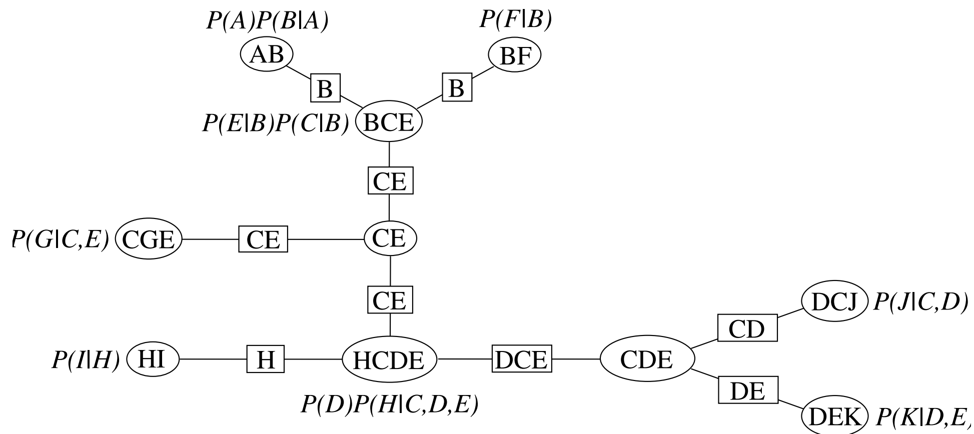


Figure 7.6: An optimized binary join tree.

Note that, when comparing the computations performed in the nonbinary join tree and in the binary one, no major difference can be found during the collect (or inward) pass, all the differences lie in the distribute (outward) pass. Indeed, the only differences between the inward inner passes performed in the nonbinary and the binary join trees are the following two messages that are computed in the latter and not in the former:

- $P(C,E)\mathbb{1}_{CE} = P(C,E)$
- $\mathbb{1}_{CD}\mathbb{1}_{DE} = \mathbb{1}_{CDE}$

Sending clique	computation	message
AB	$\sum_A P(A)P(B\|A) = P(B)$	①
BF	$\sum_F P(F\|B) = \mathbb{1}_B$	②
BCE	$\sum_B P(E\|B)P(C\|B)P(B)\mathbb{1}_B = P(C,E)$	③
CGE	$\sum_G P(G\|C,E) = \mathbb{1}_{CE}$	④
CE	$P(C,E)\mathbb{1}_{CE} = P(C,E)$	⑤
DCJ	$\sum_J P(J\|C,D) = \mathbb{1}_{CD}$	⑥
DEK	$\sum_K P(K\|D,E) = \mathbb{1}_{DE}$	⑦
CDE	$\mathbb{1}_{CD}\mathbb{1}_{DE} = \mathbb{1}_{CDE}$	⑧
HI	$\sum_I P(I\|H) = \mathbb{1}_H$	⑨

Table 7.3: The inward pass performed by Shafer-Shenoy in the binary join tree.

Receiving clique	computation	message
HI	$\sum_{C,D,E} P(H\|C,D,E)P(D)P(C,E)\mathbb{1}_{CDE} = P(H)$	❶
CDE	$\sum_H P(H\|C,D,E)P(D)P(C,E)\mathbb{1}_H = P(C,D,E)$	❷
DEK	$\sum_C P(C,D,E)\mathbb{1}_{CD} = P(D,E)$	❸
DCJ	$\sum_E P(C,D,E)\mathbb{1}_{DE} = P(C,D)$	❹
CE	$\sum_{H,D} P(H\|C,D,E)P(D)\mathbb{1}_{CDE}\mathbb{1}_H = \mathbb{1}_{CE}$	❺
CGE	$P(C,E)\mathbb{1}_{CE} = P(C,E)$	❻
BCE	$\mathbb{1}_{CE}\mathbb{1}_{CE} = \mathbb{1}_{CE}$	❼
BF	$\sum_{B,E} P(E\|B)P(C\|B)P(B)\mathbb{1}_{CE} = P(B)$	❽
AB	$\sum_{C,E} P(E\|B)P(C\|B)\mathbb{1}_B\mathbb{1}_{CE} = \mathbb{1}_B$	❾

Table 7.4: The outward pass performed by Shafer-Shenoy in the binary join tree.

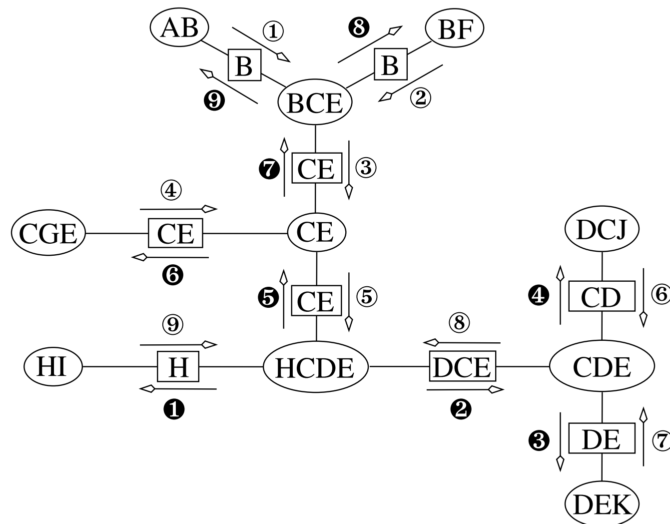


Figure 7.7: The messages sent throughout the binary join tree.

and the differences between the outer passes are given in Table 7.5. The efficiency of the binary join tree lies in the computations of the messages sent to cliques :

- DEK and DCJ where $\sum_H P(D)P(H|C,D,E)P(C,E) = P(C,D,E)$ is computed only once;
- CGE and BCE where the expression $\sum_{H,D} P(D)P(H|C,D,E)P(C,E)$ is computed as $\sum_D P(C,D,E)$.

join tree	Receiving clique	computation
nonbinary	HI	$\sum_{C,D,E} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{CE}\mathbb{1}_{DC}\mathbb{1}_{DE} = P(H)$
binary	HI	$\sum_{C,D,E} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{CDE} = P(H)$
nonbinary	DEK	$\sum_{C,H} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{CE}\mathbb{1}_{DC}\mathbb{1}_{H} = P(D,E)$
binary	DEK	$\sum_{C} P(C,D,E)\mathbb{1}_{CD} = P(D,E)$
nonbinary	DCJ	$\sum_{H,E} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{CE}\mathbb{1}_{DE}\mathbb{1}_{H} = P(C,D)$
binary	DCJ	$\sum_{E} P(C,D,E)\mathbb{1}_{DE} = P(C,D)$
nonbinary	CGE	$\sum_{H,D} P(D)P(H\|C,D,E)P(C,E)\mathbb{1}_{DC}\mathbb{1}_{DE}\mathbb{1}_{H} = P(C,E)$
binary	CGE	$P(C,E)\mathbb{1}_{CE} = P(C,E)$
nonbinary	BCE	$\sum_{H,D} P(D)P(H\|C,D,E)\mathbb{1}_{CE}\mathbb{1}_{DC}\mathbb{1}_{DE}\mathbb{1}_{H} = \mathbb{1}_{CE}$
binary	BCE	$\mathbb{1}_{CE}\mathbb{1}_{CE} = \mathbb{1}_{CE}$
binary	CDE	$\sum_{H} P(H\|C,D,E)P(D)P(C,E)\mathbb{1}_{H} = P(C,D,E)$
binary	CE	$\sum_{H,D} P(H\|C,D,E)P(D)\mathbb{1}_{CDE}\mathbb{1}_{H} = \mathbb{1}_{CE}$

Table 7.5: Comparison of Shafer-Shenoy's outer passes.

7.2 Pearl's algorithm in a triangulated Bayes net

Let us now perform our variant of Pearl's algorithm on the example of the preceding section. The first step consists in creating the triangulated Bayesian network corresponding to the elimination sequence A, F, G, I, J, K, H, D, B, C, E. The resulting graph is shown in Figure 7.8. The second step consists in retrieving the new conditional probabilities to be stored in each node of the triangulated Bayesian network. Those correspond to the following decomposition of the joint probability distribution :

$$\begin{aligned} P(\mathcal{V}) &= P(A|B)P(B|C,E)P(C|E)P(D|C,E)P(E)P(F|B) \\ &\quad P(G|C,E)P(H|C,D,E)P(I|H)P(J|C,D)P(K|D,E). \end{aligned}$$

Finally, local cutsets must be established. Using dashed arcs to represent the arcs removed by cutsetting, the graph in which Pearl's propagation will be performed is that of Figure 7.9.

Using node H as the root of Pearl's algorithm, the computations performed during the collect and the distribute phases are given in Tables 7.6 and 7.7 respectively.

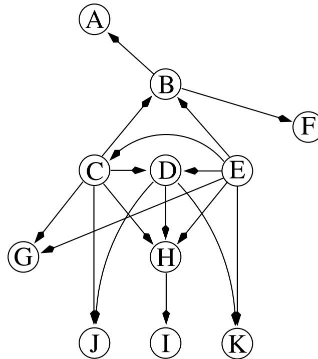


Figure 7.8: The triangulated Bayesian network.

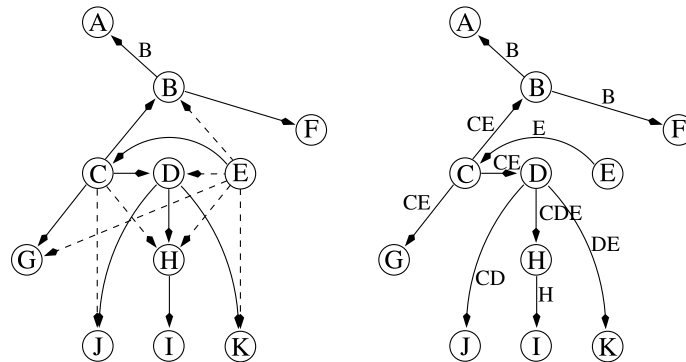


Figure 7.9: Local cutsets in the triangulated Bayesian network.

Sending node	computation	message
A	$\sum_A P(A\|B) = \mathbb{1}_B$	①
F	$\sum_F P(F\|B) = \mathbb{1}_B$	②
B	$\sum_B P(B\|C,E)\mathbb{1}_B\mathbb{1}_B = \mathbb{1}_{CE}$	③
E	$P(E)$	④
G	$\sum_G P(G\|C,E) = \mathbb{1}_{CE}$	⑤
C	$P(C\|E)P(E)\mathbb{1}_{CE}\mathbb{1}_{CE} = P(C,E)$	⑥
J	$\sum_J P(J\|C,D) = \mathbb{1}_{CD}$	⑦
K	$\sum_K P(K\|D,E) = \mathbb{1}_{DE}$	⑧
D	$P(D\|C,E)P(C,E)\mathbb{1}_{CD}\mathbb{1}_{DE} = P(C,D,E)$	⑨
I	$\sum_I P(I\|H) = \mathbb{1}_H$	⑩

Table 7.6: The collect pass performed by Pearl.

The messages are shown in Figure 7.10. The messages we sent here are roughly equivalent in terms of the number of arithmetic operations performed to those sent by Shafer-Shenoy in the nonbinary join tree. We noticed in subsection 7.1.2 that, in the nonbinary join tree, the inward pass could not be improved but that the outward pass could be speeded up by computing cleverly $\sum_H P(D)P(H|C,D,E)P(C,E)$ and

Receiving node	computation	message
I	$\sum_{C,D,E} P(H\|C,D,E)P(C,D,E) = P(H)$	❶
D	$\sum_{H} P(H\|C,D,E)\mathbb{1}_H = \mathbb{1}_{CDE}$	❷
K	$\sum_{C} P(D\|C,E)P(C,E)\mathbb{1}_{CDE}\mathbb{1}_{CD} = P(D,E)$	❸
J	$\sum_{E} P(D\|C,E)P(C,E)\mathbb{1}_{CDE}\mathbb{1}_{DE} = P(C,D)$	❹
C	$\sum_{D} P(D\|C,E)\mathbb{1}_{CDE}\mathbb{1}_{DE}\mathbb{1}_{CD} = \mathbb{1}_{CE}$	❺
G	$P(C\|E)P(E)\mathbb{1}_{CE}\mathbb{1}_{CE} = P(C,E)$	❻
E	$\sum_{C} P(C\|E)\mathbb{1}_{CE}\mathbb{1}_{CE}\mathbb{1}_{CE} = \mathbb{1}_E$	❼
B	$P(C\|E)P(E)\mathbb{1}_{CE}\mathbb{1}_{CE} = P(C,E)$	❽
F	$\sum_{C,E} P(B\|C,E)P(C,E)\mathbb{1}_B = P(B)$	❾
A	$\sum_{C,E} P(B\|C,E)P(C,E)\mathbb{1}_B = P(B)$	❿

Table 7.7: The distribute pass performed by Pearl.

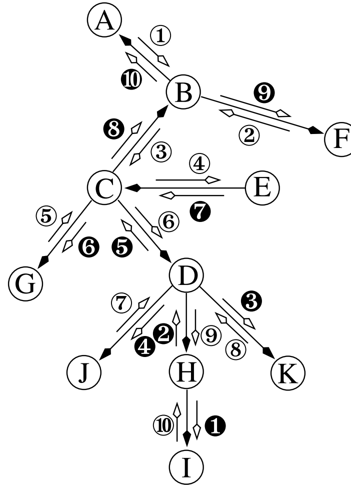


Figure 7.10: Messages sent in the triangulated Bayesian network.

$\sum_{H,D} P(D)P(H|C,D,E)P(C,E)$. Such computations were used for sending messages to cliques DEK, DCJ, CGE and BCE. In our Pearl's propagation the same observation can be made. Indeed, notice that messages sent to cliques DEK, DCJ, CGE and BCE correspond to messages sent to nodes K, J, G and B, and that the latter are:

- $\sum_{C} P(D|C,E)P(C,E)\mathbb{1}_{CDE}\mathbb{1}_{CD} = P(D,E)$,
- $\sum_{E} P(D|C,E)P(C,E)\mathbb{1}_{CDE}\mathbb{1}_{DE} = P(C,D)$,
- $P(C|E)P(E)\mathbb{1}_{CE}\mathbb{1}_{CE} = P(C,E)$,
- $P(C|E)P(E)\mathbb{1}_{CE}\mathbb{1}_{CE} = P(C,E)$.

Using the same argument as in binary join trees, we can see that $P(D|C,E)P(C,E) = P(D,C,E)$ and $P(C|E)P(E) = P(C,E)$ need be computed only once. Doing so,

we will obtain the same performance as Shafer-Shenoy with binary join trees. Now the problem that we shall solve is: "how can we easily identify such redundancies in the computations?". We could of course transform the Bayesian network as nonbinary join trees were transformed into binary join trees but, here, a simpler adaptation can be applied: as we shall see in the next section, it is sufficient to use some stacks storing temporary computations to avoid all redundancies.

7.3 Simulating BJT's with Pearl's algorithm

As we just saw, to be as efficient as Shafer-Shenoy in binary join trees, Pearl's algorithm just needs to avoid redundancies in messages computations. Each message from a node X to a node Y is computed as a summation over the product of the conditional probability stored into X by the messages sent to X by its neighbors except Y. Hence if there exist redundancies, they can only arise when a node X sends messages to several of its neighbors (as the messages have some products in common). Hence we shall study in the next subsection how messages are generated by a given node during both collect and distribute phases. This will suggest a new variation of Pearl's algorithm and we will see that the latter is not only competitive with Jensen or Shafer-Shenoy in terms of computational complexity but also in terms of the number of arithmetic operations it will perform. In a second subsection we will apply this algorithm to the example of the preceding section and show that our version of Pearl's algorithm is actually competitive with both Jensen or Shafer-Shenoy.

7.3.1 Avoiding redundancies in Pearl's algorithm

Consider an arbitrary node X in a triangulated Bayesian network (see Figure 7.11). Assume that we apply Pearl's algorithm with local cutset in this graph and that the root is on Y's side.

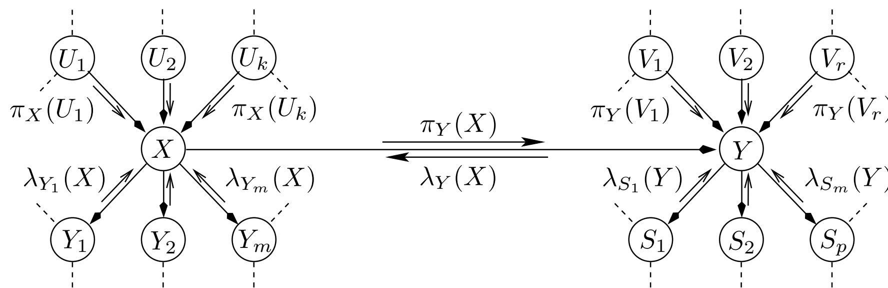


Figure 7.11: Local messages.

During the collect pass, X must send to Y message:

$$\pi_Y(X) = \sum_{Z \in \mathcal{V} \setminus C_{XY}} P(X|U_1, \ldots, U_k) \prod_{i=1}^{k} \pi_X(U_i) \prod_{j=1}^{m} \lambda_{Y_j}(X),$$

where $\mathcal{V}$ represents the set of all the random variables of the Bayesian network, and C_{XY} represents the label of arc (X, Y) (see page 87). Similarly, during the distribute pass, X must send the following messages:

$$\begin{aligned}\forall\, k,\ \lambda_X(U_k) &= \sum_{Z \in \mathcal{V} \setminus C_{U_k X}} P(X|U_1, \ldots, U_k)\lambda_Y(X) \prod_{i \neq k} \pi_X(U_i) \prod_{j=1}^{m} \lambda_{Y_j}(X), \\ \forall\, k,\ \pi_{Y_k}(X) &= \sum_{Z \in \mathcal{V} \setminus C_{XY_k}} P(X|U_1, \ldots, U_k)\lambda_Y(X) \prod_{i=1}^{k} \pi_X(U_i) \prod_{j \neq k} \lambda_{Y_j}(X).\end{aligned}$$

As we can see, the differences between the terms involved in all these products are rather small, so that it should be possible to perform many factorizations.

Remark that, on a monoprocessor computer, it is not possible to compute several multiplications at a time, so the products must be performed one by one. So, consider that during the collect pass, the products are performed using the following algorithm:

let $M_1 = P(X|U_1, \ldots, U_k)$
for i = 1 to k **do**
  let $M_{i+1} = M_i \times \pi_X(U_i)$
done
for j = 1 to m **do**
  let $M_{k+j+1} = M_{k+j} \times \lambda_{Y_j}(X)$
done

Then $\sum_{Z \in \mathcal{V} \setminus C_{XY}} M_{k+m+1}$ is obviously equal to $\pi_Y(X)$.

$$\begin{aligned}M_{k+m} &= P(X|U_1, \ldots, U_k) \prod_{i=1}^{k} \pi_X(U_i) \prod_{j<m} \lambda_{Y_j}(X), \\ \pi_{Y_m}(X) &= \sum_{Z \in \mathcal{V} \setminus C_{XY_m}} P(X|U_1, \ldots, U_k)\lambda_Y(X) \prod_{i=1}^{k} \pi_X(U_i) \prod_{j \neq m} \lambda_{Y_j}(X) \\ &= \sum_{Z \in \mathcal{V} \setminus C_{XY_m}} M_{k+m}\lambda_Y(X).\end{aligned}$$

Similarly,

$$\begin{aligned}M_{k+m-1} &= P(X|U_1, \ldots, U_k) \prod_{i=1}^{k} \pi_X(U_i) \prod_{j<m-1} \lambda_{Y_j}(X), \\ \pi_{Y_{m-1}}(X) &= \sum_{Z \in \mathcal{V} \setminus C_{XY_{m-1}}} P(X|U_1, \ldots, U_k)\lambda_Y(X) \prod_{i=1}^{k} \pi_X(U_i) \prod_{j \neq m-1} \lambda_{Y_j}(X) \\ &= \sum_{Z \in \mathcal{V} \setminus C_{XY_{m-1}}} M_{k+m-1}\lambda_Y(X)\lambda_{Y_m}(X)\end{aligned}$$

and

$$\begin{aligned} M_{k+m-2} &= P(X|U_1,\ldots,U_k)\prod_{i=1}^{k}\pi_X(U_i)\prod_{j<m-2}\lambda_{Y_j}(X), \\ \pi_{Y_{m-2}}(X) &= \sum_{Z\in\mathcal{V}\setminus C_{XY_{m-2}}} P(X|U_1,\ldots,U_k)\lambda_Y(X)\prod_{i=1}^{k}\pi_X(U_i)\prod_{j\neq m-2}\lambda_{Y_j}(X) \\ &= \sum_{Z\in\mathcal{V}\setminus C_{XY_{m-2}}} M_{k+m-2}\lambda_Y(X)\lambda_{Y_m}(X)\lambda_{Y_{m-1}}(X). \end{aligned}$$

So, had we stored all the temporary M_{k+j}'s of the above algorithm, the computations of the $\pi_{Y_r}(X)$'s, for all $r\in\{1,\ldots,m\}$ would be given by the following formula:

$$\pi_{Y_r}(X) = \sum_{Z\in\mathcal{V}\setminus C_{XY_r}} M_{k+r}\lambda_Y(X)\prod_{j=r+1}^{m}\lambda_{Y_j}(X).$$

Similarly,

$$\begin{aligned} M_k &= P(X|U_1,\ldots,U_k)\prod_{i<k}\pi_X(U_i), \\ \lambda_X(U_k) &= \sum_{Z\in\mathcal{V}\setminus C_{U_kX}} P(X|U_1,\ldots,U_k)\lambda_Y(X)\prod_{i\neq k}\pi_X(U_i)\prod_{j=1}^{m}\lambda_{Y_j}(X) \\ &= \sum_{Z\in\mathcal{V}\setminus C_{U_kX}} M_k\lambda_Y(X)\prod_{j=1}^{m}\lambda_{Y_m}(X) \end{aligned}$$

and

$$\begin{aligned} M_{k-1} &= P(X|U_1,\ldots,U_k)\prod_{i<k-1}\pi_X(U_i), \\ \lambda_X(U_{k-1}) &= \sum_{Z\in\mathcal{V}\setminus C_{U_{k-1}X}} P(X|U_1,\ldots,U_k)\lambda_Y(X)\prod_{i\neq k-1}\pi_X(U_i)\prod_{j=1}^{m}\lambda_{Y_j}(X) \\ &= \sum_{Z\in\mathcal{V}\setminus C_{U_{k-1}X}} M_k\lambda_Y(X)\prod_{j=1}^{m}\lambda_{Y_m}(X)\pi_X(U_k). \end{aligned}$$

Hence, by induction, it is easily seen that the computation of the $\lambda_X(U_r)$ messages can be computed for all $r\in\{1,\ldots,k\}$ as follows:

$$\lambda_X(U_r) = \sum_{Z\in\mathcal{V}\setminus C_{U_rX}} M_r\lambda_Y(X)\prod_{j=1}^{m}\lambda_{Y_j}(X)\prod_{i=r+1}^{k}\pi_X(U_i).$$

To summarize, the π-λ messages sent by node X to its neighbors can be computed as:

$$\begin{aligned}
\pi_Y(X) &= \sum_{Z \in \mathcal{V} \setminus C_{XY}} M_{k+m+1}, \\
\pi_{Y_r}(X) &= \sum_{Z \in \mathcal{V} \setminus C_{XY_r}} M_{k+r} \lambda_Y(X) \prod_{j=r+1}^{m} \lambda_{Y_j}(X), \\
\lambda_X(U_r) &= \sum_{Z \in \mathcal{V} \setminus C_{U_r X}} M_r \lambda_Y(X) \prod_{j=1}^{m} \lambda_{Y_j}(X) \prod_{i=r+1}^{k} \pi_X(U_i).
\end{aligned}$$

Remark now that the $\pi_{Y_r}(X)$-$\lambda_X(U_r)$ messages can be computed in any order. But if we first compute the $\pi_{Y_r}(X)$'s in r's decreasing order, and then the $\lambda_X(U_r)$ also in r's decreasing order, then the computation of most products can be avoided. Indeed, in the $\pi_{Y_r}(X)$ formula, if we denote by N_{k+r} the product $\lambda_Y(X) \times \prod_{j=r+1}^{m} \lambda_{Y_j}(X)$ for all $r \in \{1, \ldots, m\}$, then $\lambda_Y(X) \times \prod_{j=r}^{m} \lambda_{Y_j}(X) = N_{k+r-1}$ is equal to $N_{k+r} \times \lambda_{Y_r}(X)$. Similarly, if we denote by N_r, for all $r \in \{1, \ldots, k\}$, the value of the expression $\lambda_Y(X) \prod_{j=1}^{m} \lambda_{Y_j}(X) \prod_{i=r+1}^{k} \pi_X(U_i)$, then $N_k = N_{k+1} \times \lambda_{Y_1}(X)$ and, for all $r < k$, $N_r = N_{r+1} \times \pi_X(U_{r+1})$. Hence, the collect-distribute messages can be computed as:

$$\begin{aligned}
\pi_Y(X) &= \sum_{Z \in \mathcal{V} \setminus C_{XY}} M_{k+m+1}, \\
\pi_{Y_r}(X) &= \sum_{Z \in \mathcal{V} \setminus C_{XY_r}} M_{k+r} N_{k+r}, \\
\lambda_X(U_r) &= \sum_{Z \in \mathcal{V} \setminus C_{U_r X}} M_r N_r.
\end{aligned}$$

Of course, a similar treatment may be applied if Y is a parent of X. Note also that the root of the propagation algorithm may be treated slightly differently in the sense that there is no need to compute M_{k+m+1} since this matrix is just needed for the message sent toward the root during the collect phase.

This justifies the following propagation algorithm:

Algorithm 7.1 (The polytree algorithm in a BJT) *Let $(\mathcal{V}, \mathcal{A}, P)$ be a BN. To compute a posteriori marginal probabilities of all the random variables in $\mathcal{V}$, label the arcs using Algorithm 3.2 (see page 41) and apply Algorithm 6.1 (see page 87) computing the $\pi - \lambda$ messages as follows: consider a subgraph such as that of Figure 7.11. Then, during the collect phase, X stores in a stack matrices M_i obtained by the following algorithm :*

let $M_1 = P(X|U_1, \ldots, U_k)$
for i = 1 *to* k **do**
  let $M_{i+1} = M_i \times \pi_X(U_i)$
done
let $h = m$ *is* X *is not the root else* $h = m - 1$
for j = 1 *to* h **do**
  let $M_{k+j+1} = M_{k+j} \times \lambda_{Y_j}(X)$
done
X *then sends message* $\pi_Y(X) = \sum_{Z \in \mathcal{V} \setminus C_{XY}} M_{k+m+1}$. *For the distribute phase,* X *sends messages first to the* Y_r*'s, in* r*'s decreasing order, then to the* U_r*'s, also in* r*'s decreasing order, and these messages are computed as follows:*
let $N_{k+m} = \lambda_Y(X)$ *if* X *is not the root else* $N_{k+m} = \mathbb{1}_X$
for j = m *to* 1 **do**
  let $\pi_{Y_j}(X) = \sum_{Z \in \mathcal{V} \setminus C_{XY_j}} M_{k+j} N_{k+j}$
  let $N_{k+j-1} = N_{k+j} \times \lambda_{Y_j}(X)$
done
for i = k *to* 1 **do**
  let $\lambda_X(U_i) = \sum_{Z \in \mathcal{V} \setminus C_{U_i X}} M_i N_i$
  let $N_{i-1} = N_i \times \pi_X(U_i)$
done

7.3.2 Example of Pearl's propagation in a BJT

In this subsection, we will apply the above algorithm to the example of Section 7.2 and compare the number of arithmetic operations performed with those required by Jensen or Shafer-Shenoy. So, consider again the Bayesian network of Figure 7.12 and assume that we apply Algorithm 7.1 using node H as the root of the algorithm.

According to this algorithm, during the collect phase, node H asks its neighbors D and I for messages. In turn, they also ask their other neighbors for messages, and so on. We assume without loss of generality that messages are sent in the following order: first A sends its message, then F, B, E, G, C, J, K, D and finally I. In the sequel we will note $M_i(X)$ the ith element stored into the stack of node X. Now the collect phase of Algorithm 7.1 is achieved by the following operations:

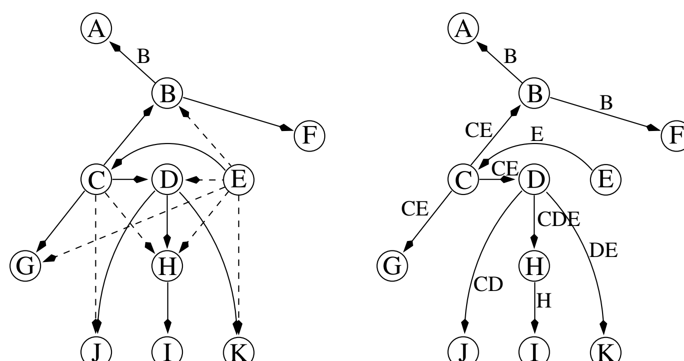


Figure 7.12: Local cutsets in the triangulated Bayesian network.

Node A: create a stack containing $M_1(A) = P(A|B)$
send message $\lambda_A(B) = \sum_A M_1(A) = \mathbb{1}_B$

Node F: create a stack containing $M_1(F) = P(F|B)$
send message $\lambda_F(B) = \sum_F M_1(F) = \mathbb{1}_B$

Node B: create a stack containing $M_1(B) = P(B|C, E)$
add to the stack $M_2(B) = M_1(B) \times \lambda_A(B) = P(B|C, E)$
add to the stack $M_3(B) = M_2(B) \times \lambda_F(B) = P(B|C, E)$
send message $\lambda_B(C) = \sum_B M_3(B) = \mathbb{1}_{CE}$

Node E: create a stack containing $M_1(E) = P(E)$
send message $\pi_C(E) = M_1(E) = P(E)$

Node G: create a stack containing $M_1(G) = P(G|C, E)$
send message $\lambda_G(C) = \sum_G M_1(G) = \mathbb{1}_{CE}$

Node C: create a stack containing $M_1(C) = P(C|E)$
add to the stack $M_2(C) = M_1(C) \times \lambda_B(C) = P(C|E)$
add to the stack $M_3(C) = M_2(C) \times \pi_C(E) = P(C, E)$
add to the stack $M_4(C) = M_3(C) \times \lambda_G(C) = P(C, E)$
send message $\pi_D(C) = M_4(C) = P(C, E)$

Node J: create a stack containing $M_1(J) = P(J|C, D)$
send message $\lambda_J(D) = \sum_J M_1(J) = \mathbb{1}_{CD}$

Node K: create a stack containing $M_1(K) = P(K|D, E)$
send message $\lambda_K(D) = \sum_K M_1(K) = \mathbb{1}_{DE}$

Node D: create a stack containing $M_1(D) = P(D|C, E)$
add to the stack $M_2(D) = M_1(D) \times \pi_D(C) = P(C, D, E)$
add to the stack $M_3(D) = M_2(D) \times \lambda_J(D) = P(C, D, E)$
add to the stack $M_4(D) = M_3(D) \times \lambda_K(D) = P(C, D, E)$
send message $\pi_H(D) = M_4(D) = P(C, D, E)$

Node I: create a stack containing $M_1(I) = P(I|H)$
send message $\lambda_I(H) = \sum_I M_1(I) = \mathbb{1}_H$

Node H: create a stack containing $M_1(H) = P(H|C, D, E)$
add to the stack $M_2(H) = M_1(H) \times \pi_H(D) = P(H, C, D, E)$

The distribute phase is achieved by the following operations:

Node H: let $N_2(H) = \mathbb{1}_H$
send message $\pi_I(H) = \sum_{C,D,E} M_2(H) \times N_2(H) = P(H)$
let $N_1(H) = N_2(H) \times \lambda_I(H) = \mathbb{1}_H$
send message $\lambda_D(H) = \sum_H M_1(H) \times N_1(H) = \mathbb{1}_{CDE}$

Node I: let $N_1(I) = \pi_I(H) = P(H)$

Node D: let $N_3(D) = \lambda_H(D) = \mathbb{1}_{CDE}$
send message $\pi_K(D) = \sum_C M_3(D) \times N_3(D) = P(D, E)$
let $N_2(D) = N_3(D) \times \lambda_K(D) = \mathbb{1}_{CDE}$
send message $\pi_J(D) = \sum_E M_2(D) \times N_2(D) = P(C, D)$
let $N_1(D) = N_2(D) \times \lambda_J(D) = \mathbb{1}_{CDE}$
send message $\lambda_D(C) = \sum_D M_1(D) \times N_1(D) = \mathbb{1}_{CE}$

Node K: let $N_1(K) = \pi_K(D) = P(D, E)$

Node J: let $N_1(K) = \pi_J(D) = P(C, D)$

Node C: let $N_3(C) = \lambda_D(C) = \mathbb{1}_{CE}$
send message $\pi_G(C) = M_3(C) \times N_3(C) = P(C, E)$
let $N_2(C) = N_3(C) \times \lambda_G(C) = \mathbb{1}_{CE}$
send message $\lambda_C(E) = \sum_C M_2(C) \times N_2(C) = \mathbb{1}_E$
let $N_1(C) = N_2(C) \times \pi_C(E) = P_C(E)$
send message $\pi_B(C) = M_1(C) \times N_1(C) = P(C, E)$

Node G: let $N_1(G) = \pi_G(C) = P(C, E)$

Node E: let $N_1(E) = \lambda_C(E) = \mathbb{1}_E$

Node B: let $N_2(B) = \pi_B(C) = P(C, E)$
send message $\pi_F(B) = \sum_{C,E} M_2(B) \times N_2(B) = P(B)$
let $N_1(B) = N_2(B) \times \lambda_F(B) = P_B(C, E)$
send message $\pi_A(B) = \sum_{C,E} M_1(B) \times N_1(B) = P(B)$

Node F: let $N_1(F) = \pi_F(B) = P(B)$

Node A: let $N_1(A) = \pi_A(B) = P(B)$

To summarize, our algorithm has performed the following arithmetic operations:

Collect phase		
Sending node	additions	multiplications
A	$\|AB\|$	
F	$\|FB\|$	
B	$\|BCE\|$	$2\|BCE\|$
G	$\|GCE\|$	
C		$3\|CE\|$
J	$\|JCD\|$	
K	$\|KDE\|$	
D		$3\|DCE\|$
I	$\|IH\|$	
H		$\|HCDE\|$

Distribute phase		
Sending node	additions	multiplications
H	$2\|HCDE\|$	$\|H\| + 2\|HCDE\|$
D	$3\|CDE\|$	$5\|CDE\|$
C	$2\|CE\|$	$5\|CE\|$
B	$3\|BCE\|$	$3\|BCE\|$

whereas Shafer-Shenoy used the following arithmetic operations:

Collect phase		
Sending clique	additions	multiplications
AB	$\|AB\|$	$\|AB\|$
BF	$\|FB\|$	
BCE	$\|BCE\|$	$\|B\| + \|BC\| + \|BCE\|$
CGE	$\|GCE\|$	
CE		$\|CE\|$
DCJ	$\|JCD\|$	
DEK	$\|KDE\|$	
CDE		$\|CDE\|$
HI	$\|IH\|$	

Distribute phase		
HI	$\|HCDE\|$	$2\|CDE\| + \|HCDE\|$
CDE	$\|HCDE\|$	$\|CDE\| + 2\|HCDE\|$
DEK	$\|CDE\|$	$\|CDE\|$
DCJ	$\|CDE\|$	$\|CDE\|$
CE	$\|HCDE\|$	$\|CDE\| + 2\|HCDE\|$
CGE		$\|CE\|$
BCE		$\|CE\|$
BF	$\|BCE\|$	$\|BC\| + 2\|BCE\|$
AB	$\|BCE\|$	$\|BC\| + 2\|BCE\|$

The difference |number of operations in Shafer-Shenoy| − |number of operations in our Pearl variant| is thus equal to:

$$\begin{array}{rl} |HCDE| - |CDE| - 2|CE| - |BCE| & \text{additions,} \\ 2|HCDE| + 3|BC| + |AB| + |B| - |CDE| - 5|CE| - |H| & \text{multiplications.} \end{array}$$

If all random variables have 10 possible values, then our algorithm outperforms Shafer-Shenoy by 7800 additions and 20000 multiplications. Of course, the computations of Shafer-Shenoy's algorithm may be improved by observing that, during the distribute phase, some computations still may be factorized. But then, additional savings may also be achieved in our algorithm by observing that there is no need to postpone the summations in the M_i's and the N_j's until we send messages: we could also perform some summations in the induction formula used for computing the M_i's or the N_j's. Thus, our algorithm would still be competitive with that of Shafer-Shenoy or Jensen.

Chapter 8

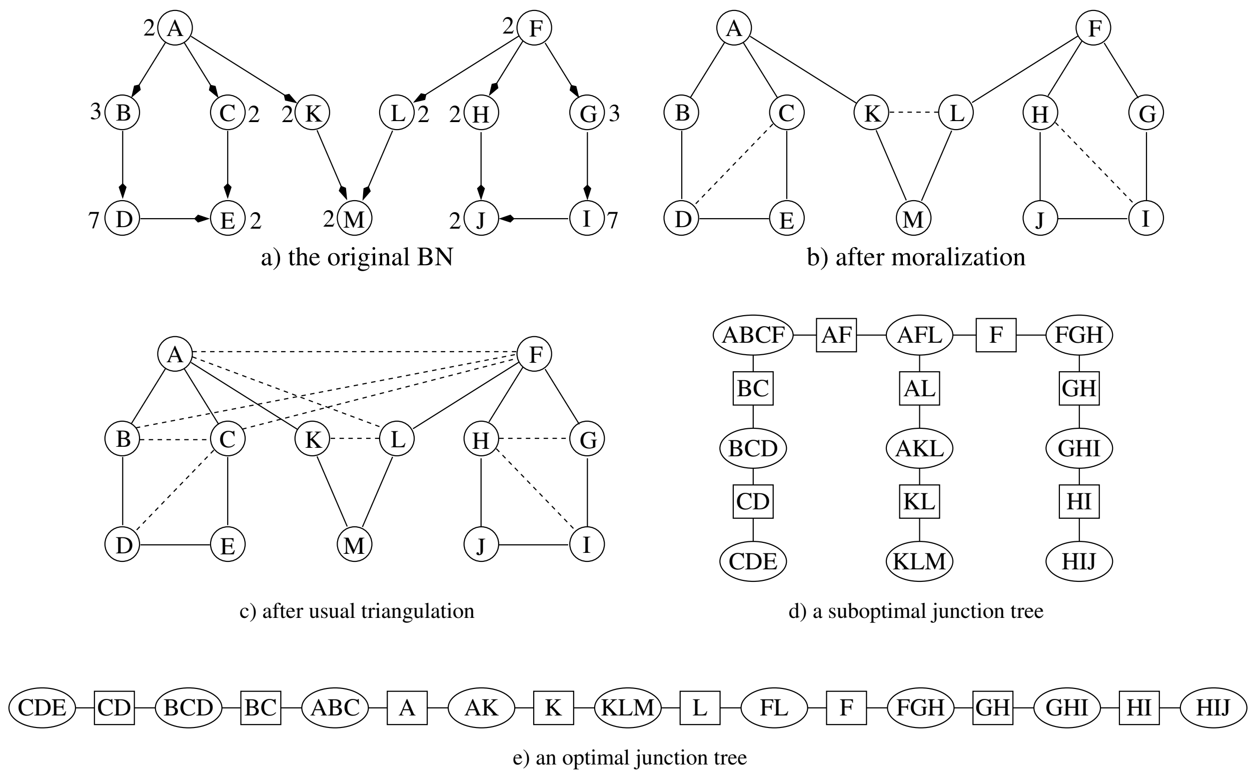
Updating undirected methods with local triangulation

There exist multiple ways to convert a BN into a junction tree [BG96b, JJ94, Kjæ90, SG97] and unfortunately the efficiency of inference techniques varies widely from one junction tree to another. Moreover finding optimal junction trees, i.e., minimizing the computational effort of probabilistic inference, being NP-hard [ACP87], only heuristics are used in practice. Hence algorithms producing efficient junction trees are crucial for probabilistic computations and this chapter is devoted to improve existing ones. These results should improve all inference mechanisms, including our Triangulated Bayes Net method.

Almost all algorithms for creating junction trees share the same idea: they first moralize the BN, i.e., they add undirected edges between every pair of parents. Then they remove the arcs directions (see Figure 8.1(b)) and they triangulate the resulting graph (Figure 8.1(c)). The cliques obtained (maximal complete subgraphs) form the nodes of the junction tree (Figure 8.1(d)). Finally edges are added to the latter to respect the *running intersection property*. Usually, different algorithms only differ in the way they perform the triangulation.

As was shown in section 4.2, triangulating a graph consists in adding new edges called *fill-ins* so that each cycle of length greater than 3 contains a chord, that is, an edge connecting two non adjacent nodes of the cycle. As such, only fill-ins connecting nodes belonging to a same cycle should be added, others can be dispensed with. However, in practice it often turns out that algorithms add fill-ins between nodes that do not belong to the same cycle, thus creating suboptimal junction trees. For instance, moralizing the BN of Figure 8.1(a), where the numbers beside nodes represent the number of values that the random variable can take, results in the graph of Figure 8.1(b). This graph is almost triangulated, that is, only edges (B, C) and (G, H) are needed to make it triangulated. However classical software such as HUGIN are unable to see it and they add unnecessary edges such as those of Figure 8.1(c). As the computational complexity of probabilistic inference deeply depends on the size of the cliques and on the topology of the junction tree, it is undesirable to obtain the junction tree of Figure 8.1(d) (which

directly follows from the triangulated graph of Figure 8.1(c)) and indeed computations performed in this tree are slower than those performed on the optimal tree. For large BN, finding as best as possible junction trees is appreciable. The purpose of this chapter is to present a preprocess for triangulation algorithms that avoids adding such unnecessary edges. This preprocess is very flexible in that it is fast and can benefit to any triangulation algorithm.



a) the original BN

b) after moralization

c) after usual triangulation

d) a suboptimal junction tree

e) an optimal junction tree

Figure 8.1: From a BN to Junction Trees.

More precisely, we will describe an efficient algorithm for producing junction trees avoiding adding any fill-in where there are no cycle. The key idea is to extract from the BN subgraphs that only contain intersecting cycles and to triangulate them separately. A second pass then aggregates all the resulting subgraphs to produce a triangulated graph of the original BN. We also show how junction subtrees resulting from each separate triangulation can be aggregated to result in a junction tree of the whole BN.

Section 8.1 shows the key idea for triangulating BN without creating fill-ins connecting nodes not belonging to the same cycle. Section 8.2 presents an algorithm based on the same idea that finds junction trees without proceeding to the triangulation of the whole BN. Empirical results obtained both on randomly generated BN and on classical benchmarks are given in Section 8.3. Finally proofs of all the propositions and lemma are given at the end of the chapter.

8.1 Triangulation avoiding unnecessary fill-ins

The aim of this section is to propose an algorithm for triangulating BN that avoids adding fill-ins connecting nodes that do not belong to the same cycle. The idea is simple: if no moralization were needed, extracting subgraphs of $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ constituted only by intersecting cycles and triangulating separately each of them would be sufficient to triangulate $\mathcal{G}$. We know that moralization is needed to ensure that the sets of variables in each conditional probability of the BN can be included in some clique of the triangulated graph. To take into account moralization, it should be sufficient to extract subgraphs of $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ constituted only by intersecting cycles, then to moralize and triangulate separately each subgraph, and finally to add the edges that should have been added during the moralization of the whole of $\mathcal{G}$ but that were missed by moralizing only subgraphs.

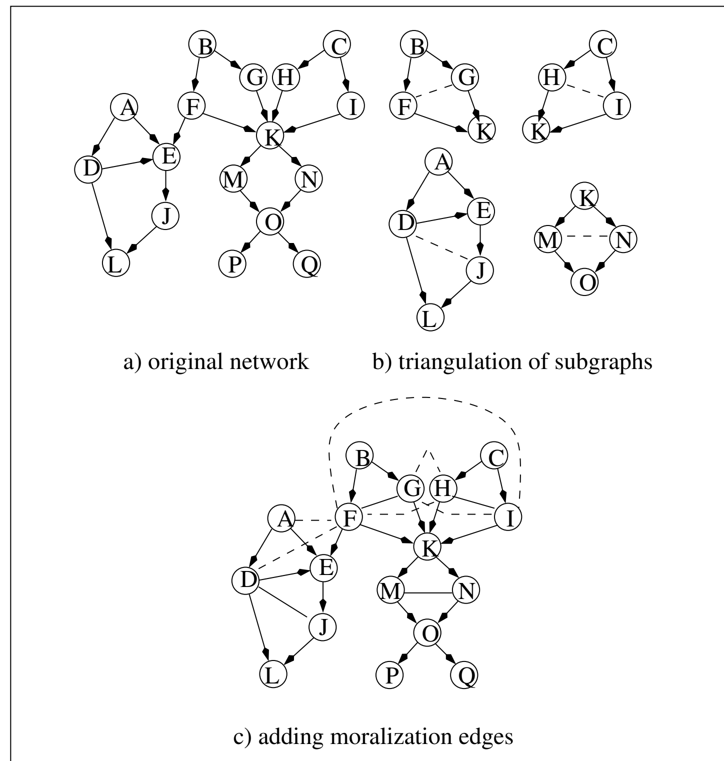


Figure 8.2: An Example of Triangulation.

Consider indeed the network of Figure 8.2(a). Subgraphs containing only intersecting cycles are pictured on Figure 8.2(b), where fill-ins added by moralization and triangulation are displayed with dashed lines. Note the lower left subgraph contains two cycles since those are intersecting, i.e., they have at least one arc in common. Adding only the fill-ins of each subgraph to the BN would be sufficient to triangulate it, but some moralization edges would prevent conditional probabilities to belong to some clique. For instance, as no clique would contain both A, D, E and F, it would not

be possible to store $P(E|A,D,F)$ in any clique. Hence additional "moralizing" edges should be added (the dashed lines on Figure 8.2(c)). Note that these edges do not alter triangulatedness, i.e., the graph of Figure 8.2(c) is triangulated. This applies for any graph as is shown in proposition 8.1 below.

Before stating this proposition, we shall briefly explain how the subgraphs can be retrieved from the original BN. For this purpose, let us recall some notions of graph theory [Ata99]. An *articulation node* is a vertex whose deletion along with its adjacent edges breaks up the remaining graph into two or more disconnected pieces. A graph is said to be *biconnected* if it has no articulation node. A *biconnected component* of a connected graph is a maximal subset of edges such that the corresponding induced subgraph is biconnected. It is easily seen that the set of subgraphs we triangulate corresponds precisely to the set of biconnected components of size strictly greater than 1 (those of size 1 correspond to arcs not belonging to any cycle). Then, the following proposition ensures that the operations we performed on the graph of Figure 8.2(a) apply for any BN.

Proposition 8.1 *Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a connected DAG, let $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_k\}$ be the set of biconnected components of $\mathcal{G}$ such that $|\mathcal{B}_i| > 1$. For any $i \in \{1, \ldots, k\}$, let $\mathcal{G}_{\mathcal{B}_i}$ be the subgraph induced by $\mathcal{B}_i$, i.e., the graph the nodes of which are those of $\mathcal{V}$ adjacent to the edges of $\mathcal{B}_i$, and the set of edges of which is $\mathcal{B}_i$. Let $\mathcal{G}_*$ be the graph resulting from the application of the following three steps :*

1. *moralize and triangulate separately (using any algorithm) every $\mathcal{G}_{\mathcal{B}_i}$. Let $\mathcal{E}_T$ be the union of the sets of fill-ins added to every $\mathcal{G}_{\mathcal{B}_i}$'s;*
2. *for each $X \in \mathcal{V}$, moralize X's parents in $\mathcal{G}$;*
3. *change all directed edges into undirected edges in the resulting graph and add all the edges of $\mathcal{E}_T$.*

Then $\mathcal{G}_*$ is a triangulated undirected graph.

As shown in the next proposition, proposition 8.1 is particularly attractive as the application of its 3 steps can infer an optimal triangulation for the whole BN from optimal triangulations of subgraphs (the $\mathcal{G}_{\mathcal{B}_i}$'s).

Proposition 8.2 *If the triangulations performed on step 1 of proposition 8.1 are optimal for every $\mathcal{G}_{\mathcal{B}_i}$ (in the sense that they produce small-size cliques [Mel87]), then $\mathcal{G}_*$ is optimal as well.*

Extracting biconnected components from a graph is an easy task and can be achieved by a depth first search procedure with a complexity of $O(|\mathcal{V}|+|\mathcal{A}|)$ [HT73, Tar72]. The application of the three steps described in proposition 8.1 creates a triangulated graph $\mathcal{G}_*$ from which a junction tree can be derived [JJ94]. However, this may not be the most efficient way (in terms of computational complexity) to get a junction tree. Indeed triangulation methods traditionally use a node elimination technique and the elimination ordering determines both the cliques and the edges of a corresponding junction tree. As the triangulations of the subgraphs induce some orders on subsets of $\mathcal{V}$, these may certainly be used to speed up the process of creating a junction tree compatible with $\mathcal{G}_*$.

The aim of the next section is to show how this can be achieved efficiently in practice.

8.2 A new algorithm for finding junction trees

The idea of our algorithm is to create the junction trees for each separate subgraph containing intersecting cycles (the $\mathcal{G}_{\mathcal{B}_i}$'s). Then, to take into account step 2 of proposition 8.1, these "local" trees are aggregated and modified to form a "global" junction tree of the BN. We first illustrate this principle on the BN of Figure 8.2(a) and, then, we derive from this example a generic procedure.

Consider again the BN of Figure 8.2(a), that we reproduced on Figure 8.3(a). Subgraphs containing only cycles are pictured on Figure 8.3(b) and their corresponding junction trees on Figure 8.3(c). The latter suggests the basic operations needed for converting these local junction (sub)trees into a global one:

- *Operation 1:* some cliques belonging to different subtrees share some random variables and, thus, should become connected in the global junction tree. This is for instance the case of cliques FGK and KMN that share variable K;
- *Operation 2:* step 2 of proposition 8.1 may require merging some cliques of different subtrees into one superset. For instance FGK and HIK should be substituted by a clique $FGHIK$ because F, G, H and I are K's parents;
- *Operation 3:* some random variables such as P do not belong to any local tree's clique and, consequently, some new cliques may have to be created.

In order to obtain an efficient algorithm, *Operation* 1 requires that we can retrieve quickly the sets of cliques sharing some variables. This can be done during the subgraphs triangulation stage: it is sufficient to create a data structure $\mathcal{C}_X$ associated to each random variable X and keeping track of the cliques X belongs to. As inside each subtree, cliques sharing some variables are connected, *Operation* 1 requires that $\mathcal{C}_X$ keeps track of only one clique in each subtree. *Operation* 2 suggests that when there are several possible choices, the clique that contains both X and all of its parents in the subgraph should be chosen to belong to $\mathcal{C}_X$. Then when *Operation* 2 applies, the cliques that should be replaced by supersets are quickly identified. For instance, $\mathcal{C}_E$ should contain ADE rather than DEJ because A and D are E's parent set in the subgraph induced by $ADEJL$. The construction of the $\mathcal{C}_X$'s can be achieved during the subgraphs triangulation phase: before each triangulation, mark all the variables of the subgraph as having no clique stored. When a node X is eliminated, if it has the "no clique" mark, store the newly created clique within $\mathcal{C}_X$ and mark the node as "having a clique"; for each child of X having the "no clique" mark, do the same thing. This process is based on the fact that, before triangulation, the moralization within the subgraph ensures that all parents of a node are connected together. Thus, for the BN of Figure 8.3(a), after the triangulation of the subgraphs, we shall obtain the following $\mathcal{C}_X$'s:

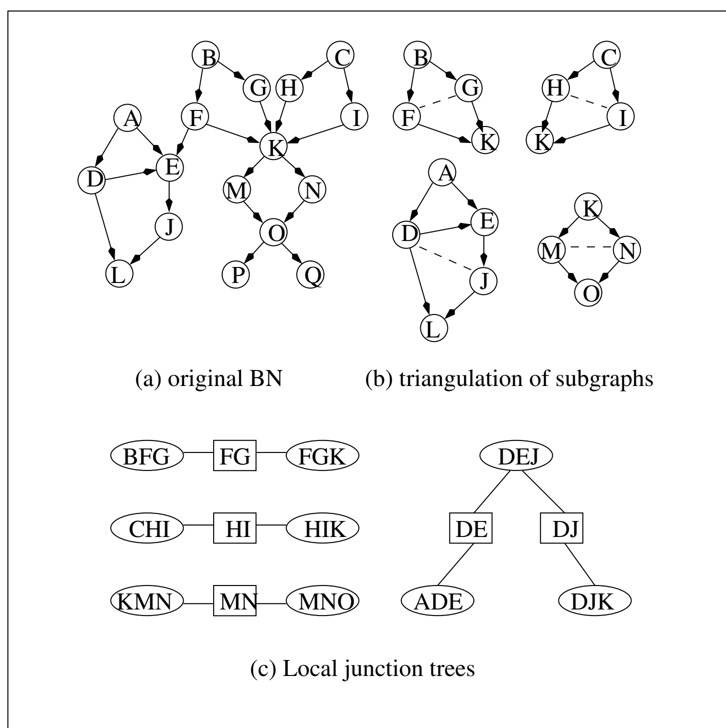


Figure 8.3: Local Junction Trees.

$$\begin{array}{lll}
\mathcal{C}_A = \{ADE\} & \mathcal{C}_B = \{BFG\} & \mathcal{C}_C = \{CHI\} \\
\mathcal{C}_D = \{ADE\} & \mathcal{C}_E = \{ADE\} & \mathcal{C}_F = \{BFG\} \\
\mathcal{C}_G = \{BFG\} & \mathcal{C}_H = \{CHI\} & \mathcal{C}_I = \{CHI\} \\
\mathcal{C}_J = \{DEJ\} & \mathcal{C}_L = \{DJL\} & \mathcal{C}_M = \{KMN\} \\
\mathcal{C}_N = \{KMN\} & \mathcal{C}_O = \{MNO\} & \mathcal{C}_P = \mathcal{C}_Q = \emptyset \\
\mathcal{C}_K = \{FGK, HIK, KMN\} & &
\end{array}$$

Now, assume the collection of junction subtrees of Figure 8.3(c) have been created and let us finalize the global junction tree construction. We will consider successively each node/random variable of the original BN in the inverse topological order, i.e. a node will be examined only when all of its children have already been examined, and we will find how the collection of subtrees constructed so far need be updated in order to take into account this variable as well as its parent set. Thus, consider node P. This node and its parent set constitute clique OP. Problem: does there already exist a clique containing OP in the collection of subtrees? If such a clique existed, it would belong to $\mathcal{C}_P$. As $\mathcal{C}_P = \emptyset$, set OP is a new clique and should be added to the collection of junction subtrees. Moreover, since it is not connected to the other cliques yet, it represents a new subtree and since O belongs to it, OP should be added to $\mathcal{C}_O$, which now becomes $\mathcal{C}_O = \{MNO, OP\}$. Similarly, examining Q results in adding clique OQ to the collection of junction subtrees and to add OQ to $\mathcal{C}_O$, thus becoming $\mathcal{C}_O = \{MNO, OP, OQ\}$.

Let us examine node O: This node and its parents constitute set MNO, which already exists in $\mathcal{C}_O$, hence there is no need to create a new clique MNO. However, $\mathcal{C}_O$ contains 3 cliques, namely MNO, OP, OQ, that are not yet connected and that should be since they all contain O. Hence we should add some edges, for instance (OP, MNO) and (OQ, MNO) (see Figure 8.4(a)). Note that this ensures that the Running Intersection Property holds in each subtree. Examine node M: $\mathcal{C}_M = \{KMN\}$ hence there is no need to create a new clique MK containing M and its parent K. Moreover, since $\mathcal{C}_M$ contains only one clique, no new edge need be added. Similarly for N. Consider now node K. $\mathcal{C}_K = \{FGK, HIK, KMN\}$. Here, K and its parents create a new clique $KFGHI$ and the latter should replace both FGK and HIK. In other words, in the collection of junction subtrees, cliques that were adjacent to FGK or HIK should now be adjacent to $KFGHI$ and, similarly, in all the $\mathcal{C}_X$'s, FGK and HIK should be substituted by $KFGHI$ (i.e. $\mathcal{C}_K = \{KFGHI, KMN\}$). Moreover, as $\mathcal{C}_K$ contains two elements, $KFGHI$ and KMN, these should be connected (see Figure 8.4(b)).

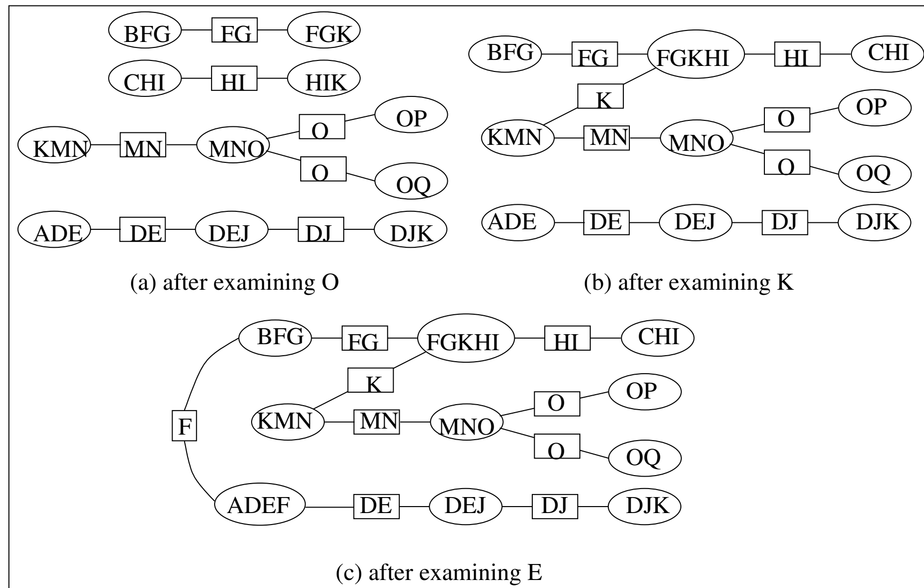


Figure 8.4: The Modifications Brought to the Collection of Junction Subtrees.

It can be easily seen that examining nodes L and J do neither change the $\mathcal{C}_X$'s nor the collection of junction subtrees. Let us examine node E: it should form a new clique $ADEF$ that should replace ADE. Moreover, as F belongs to $ADEF$, the latter should be added to $\mathcal{C}_F$. Now, we can state a general rule: when a new clique $T = \{X\} \cup Pa(X)$ is created, for each random variable Y in T either there exists a clique S in $\mathcal{C}_Y$ that is included in T and S is substituted by T both in $\mathcal{C}_Y$ (as well as in all $\mathcal{C}_Z$'s containing S) and in the collection of junction subtrees, or T should be added to $\mathcal{C}_Y$. Moreover, when we examine a node X, if $\mathcal{C}_X$ contains several cliques, we always connect them. Note that, to ensure that the Running Intersection Property

holds, it is sufficient to connect each such clique to that which contains both X and all of its parents in the BN. Here as $\mathcal{C}_E = \{ADEF\}$, no edge needs be added. Similarly, examining nodes D and A will neither change the graph nor the $\mathcal{C}_X$'s. Examine node F: according to its parent set, a clique BF should be created. However, it is contained in clique BFG that already exists. Hence no new clique will be added to the graph. As $\mathcal{C}_F = \{BFG, ADEF\}$ contains 2 cliques, these should be connected, hence resulting in the graph of Figure 8.4(c). Finally, the examination of the remaining nodes of the BN do neither modify the tree obtained nor any of the $\mathcal{C}_X$'s.

Assuming that $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ denotes the original BN, that $\mathcal{G}_T = (\mathcal{V}_T, \mathcal{E}_T)$ denotes the collection of junction subtrees resulting from the subgraphs triangulations, and that $\mathcal{B}$ is the set of biconnected components of the BN of size strictly greater than 1, the above procedure can be stated more generally as function **finalize_JT** below:

Function finalize_JT $(\mathcal{G}, \mathcal{G}_T, \mathcal{B})$
01 $\mathcal{V}'' = \emptyset$; $\mathcal{V}' \leftarrow \{X \in \mathcal{V}$: there exists no $(X, Y) \in \mathcal{A}\}$
02 **while** $\mathcal{V}' \neq \emptyset$ **do**
03 $X \leftarrow$ any element of $\mathcal{V}'$
04 $\mathcal{V}' \leftarrow \mathcal{V}' \backslash \{X\}$; $NC \leftarrow \{X\} \cup pa(X)$
05 **if** there exists $C_X \in \mathcal{C}_X$ s.t. $NC \subseteq C_X$
06 **then** $NC \leftarrow C_X$; $\mathcal{C}_X \leftarrow \mathcal{C}_X \backslash \{C_X\}$
07 **else** $\mathcal{V}_T \leftarrow \mathcal{V}_T \cup NC$
08 **endif**
09 **for each** $C_X \in \mathcal{C}_X$ **do**
10 **if** $C_X \subset NC$ **then**
11 **for** $C_Y \in \mathcal{V}_T$ s.t. $(C_X, C_Y) \in \mathcal{E}_T$ **do**
12 $\mathcal{E}_T \leftarrow \{(NC, C_Y)\} \cup \mathcal{E}_T \backslash \{(C_X, C_Y)\}$
13 **done**
14 **for each** $Y \in \mathcal{V} \backslash \mathcal{V}''$ **do**
15 **if** $C_X \in \mathcal{C}_Y$ **then** $\mathcal{C}_Y \leftarrow \mathcal{C}_Y \backslash \{C_X\}$
16 **done**
17 $\mathcal{V}_T \leftarrow \mathcal{V}_T \backslash \{C_X\}$
18 **endif**
19 **done**
20 **while** $\mathcal{C}_X \neq \emptyset$ **do**
21 $C_X \leftarrow$ any element of $\mathcal{C}_X$
22 $\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{(NC, C_X)\}$; $\mathcal{C}_X \leftarrow \mathcal{C}_X \backslash \{C_X\}$
23 **done**
24 $\mathcal{V}'' \leftarrow \mathcal{V}'' \cup \{X\}$
25 **for each** $Y \in pa(X)$ **do**
26 **if** there exists no i s.t. $(Y, X) \in \mathcal{B}_i$ **then**
27 $\mathcal{C}_Y \leftarrow \mathcal{C}_Y \cup NC$
28 **endif**
29 **if** $\{(Y, Z) \in \mathcal{A} : Z \in \mathcal{V} \backslash \mathcal{V}''\} = \emptyset$ **then**
30 $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{Y\}$
31 **endif**
32 **done**
33 **done**

In this function, lines 03 to 08 test whether for a given node X of the BN a new clique NC containing precisely $X \cup Pa(X)$ should be created. Lines 09 to 19 substitute each clique C_Y included in NC by NC (both in the $\mathcal{C}_Z$'s and in the collection of junction subtrees). Lines 20 to 23 add edges connecting NC to any clique in $\mathcal{C}_X$, thus ensuring that the running intersection property holds within all subtrees. Lines 26–28 add NC to the data structures of nodes that were connected to X by edges not belonging to any cycle. The following proposition shows that it computes properly junction trees.

Proposition 8.3 *After performing* **finalize_JT**, $\mathcal{G}_T$ *is a junction tree.* **finalize_JT** *can be implemented in* $O(|\mathcal{V}| + |\mathcal{A}|)$. *Moreover, if the junction subtrees obtained for the* $\mathcal{G}_{\mathcal{B}_i}$*'s are optimal, the cliques of* $\mathcal{G}_T$ *are those of an optimal junction tree.*

Proposition 8.3 does not preclude the possibility that $\mathcal{G}_T$ may be suboptimal even when the junction subtrees of the $\mathcal{G}_{\mathcal{B}_i}$'s are optimal. The reason is that there may be several ways to link cliques so as to verify the running intersection property. An optimal junction tree may however be inferred using the algorithm advocated by [JJ94].

8.3 Empirical results

As there is no guaranty as to whether the result produced by function **finalize_JT** is optimal, the efficiency of the trees produced by our algorithm may be assessed through empirical results. To obtain meaningful comparisons, for every network of our benchmark, function **finalize_JT** was applied using the triangulation technique advocated by [Kjæ90] on each $\mathcal{G}_{\mathcal{B}_i}$, then the same technique was applied on the whole BN. The efficiency of our algorithm for a particular network is computed as the ratio of the number of operations required by Shafer-Shenoy's algorithm to send messages on both directions of every edge of the JT obtained by our algorithm divided by the number of operations required in the JT obtained by moralizing and triangulating the whole BN. Thus, when this ratio is less than 1, computations performed in the junction tree produced by our algorithm are faster than those performed in the junction tree resulting from the classical triangulation method. Our tests are twofold: first they are run on randomly generated BN, then on the networks of the Bayes net repository (http://www.cs.huji.ac.il/labs/compbio/Repository/).

The results of the first collection of tests is represented on Figure 8.5. In this figure, the Y-axis represents the ratios of the numbers of operations required by inference and the X-axis represents the ratio of the number of edges in the original BN by the number of nodes in this BN (thus giving some insight about the density of the network). Furthermore, each curve corresponds to the average ratio of number of operations obtained on a set of networks having the same number of nodes. The upper curve thus corresponds to network with 20 nodes. Experimentally, it turns out that the lower the curve the higher the number of nodes. The lowest curve corresponds to BN with 200 nodes. During the tests, BN were generated randomly with the constraint that they are connected. The number of values taken by random variables/nodes were also drawn randomly between 2 and 5.

Figure 8.5 shows that the bigger the network, the higher the expectation of improving inference using our triangulation algorithm. Of course, this is only an expectation as the gain using our triangulation depends highly on the topology of the network, which is only partially taken into account in the density ratio we used in our tests. The latter are restricted on the Figure between 1 and 3 as a ratio below 1 would correspond to disconnected BN and a ratio above 3 most often implies that all cycles of the BN intersect each other. In such a case, both triangulations perform the same operations and the

ratios of the operations required by inference equal 1.

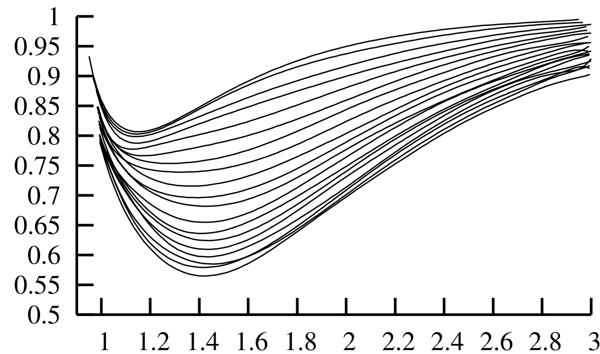


Figure 8.5: Efficiency of our Triangulation on Randomly Generated Graphs.

network	ratio
alarm	0.970963
diabetes	0.999787
link	1.000000
munin2	0.696683
munin4	0.778010
water	1.000000

network	ratio
carpo	0.659218
insurance	0.810709
munin1	0.715494
munin3	0.673786
pigs	0.717863
win95pts	0.922649

Figure 8.6: Efficiency of our Triangulation in Practical Situations.

As a conclusion, we can say that the construction of a junction tree optimizing probabilistic inference is known to be NP-hard, hence heuristics are applied in practice and the resulting junction trees may be far from optimal. These heuristics first moralize, then triangulate the original BN. Due to the moralization stage, it often happens that triangulation adds edges connecting nodes not belonging to a common cycle, thus creating suboptimal junction trees. [Kjæ90] proposes two algorithms for detecting unnecessary edges both based on a result by [RTL76] having a complexity of $O(c^2|T|^2)$ and $O(c^2(|T|/|\mathcal{V}|)^2)$ respectively, where T is the set of fill-ins and c is a constant representing the density of the graph.

In this chapter we proposed an efficient algorithm that never adds any edge connecting nodes that do not belong to the same cycle. Our algorithm is efficient in the sense that the complexity of the operations it performs beside the triangulation of each subgraph (which cannot be dispensed with) is linear in the number of edges and nodes in the whole BN. For instance, instructions such as the test of line 05 can be performed in $O(1)$: if there existed $C_X \subseteq NC$, then all of X's parents would have belonged to intersecting cycles. Consequently, all arcs from each of X's parents to X in the original BN would have belonged to the same biconnected component $\mathcal{B}_i$ of size strictly greater

than 1. The biconnected components retrieval algorithm [Ata99] can be easily modified while keeping the same overall complexity so as to mark every node such that all of its incoming arcs belong to the same biconnected component. As the complexity of triangulation is higher than linear and as we only perform it only on subgraphs, the overall number of operations we need to construct a junction tree is very often less than that required by classical algorithms. Moreover, most often, the junction trees we create are "better" than those derived from classical algorithms (although sometimes it may turn out that the latter find better trees when applied on some graphs with more edges than fewer). Thus our algorithm is fast and most often creates junction trees that are "better" than those created by classical algorithms.

8.4 Proofs

Lemma 8.1 *Function* **finalize_JT** *terminates in a finite time. Moreover, every node of* $\mathcal{V}$ *is assigned to* X *on line 03 exactly once.*

Proof of lemma 8.1: At the beginning of the first pass of the outer `while` loop, i.e. the loop of lines 02–33, $\mathcal{V}'$ corresponds to the set of nodes of $\mathcal{V}\backslash\mathcal{V}''$ that have no children except those in $\mathcal{V}''$. Assume this holds until the ith pass in the `while` loop. In the next pass, $\mathcal{V}''$ is modified by the addition of node X (line 24), hence if $\mathcal{V}'$ is to correspond to the set of nodes of $\mathcal{V}\backslash\mathcal{V}''$ that have no children except those in $\mathcal{V}''$, X shall be removed from $\mathcal{V}'$ and the parents of X that do not have any children except those in $\mathcal{V}''$ shall be added to $\mathcal{V}'$ (they did not already belonged to $\mathcal{V}'$ since X did not belong to $\mathcal{V}''$ before the pass). But this is precisely what is performed on lines 03 and 29–31. Hence, each time the algorithm performs a `while` loop, $\mathcal{V}'$ is equal to the set of nodes of $\mathcal{V}\backslash\mathcal{V}''$ that have no children except those in $\mathcal{V}''$. At each of these passes some node X is added to $\mathcal{V}''$, hence X cannot be added later to $\mathcal{V}'$. So each node is assigned at most once to line 03.

Assume now some node Y of $\mathcal{G}$ is never assigned to X on line 03, or equivalently that Y never belongs to $\mathcal{V}''$, then Y is never added to $\mathcal{V}'$. So the condition on line 29 either is never true for Y or is never reached for Y. In the first case, each time a child of Y has been assigned to X on line 03, Y still had a child in $\mathcal{V}\backslash\mathcal{V}''$ and so Y never belonging to $\mathcal{V}''$ implies that one of its children also never belongs to $\mathcal{V}''$; In the second case, no child of Y is never assigned on line 03, hence no child of Y belongs to $\mathcal{V}''$. So, in both cases, Y never belonging to $\mathcal{V}''$ implies there exists a child of Y not belonging either to $\mathcal{V}''$, hence resulting by induction in Y having an infinite set of descendants not belonging to $\mathcal{V}''$, which is impossible in a finite DAG. Consequently, each node of $\mathcal{G}$ is assigned exactly once to X on line 03.

At the beginning of the algorithm, for every Y of $\mathcal{G}$, $\mathcal{C}_Y$ is a finite set. Only line 27 can add elements to $\mathcal{C}_Y$ and, moreover, at most one element is added by each pass of the outer `while` loop. Consequently, as this outer loop is performed a finite number

of times, all the inner loops are finite and the whole function terminates in a finite time. ♦

Lemma 8.2 *After performing* **finalize_JT**, $\mathcal{G}_T$ *is a singly connected graph.*

Proof of lemma 8.2: The proof consists in showing that after performing each `while` loop of lines 02–33, the four following properties hold:

1. $\mathcal{G}_T$ is singly connected;
2. for any pair of distinct trees of $\mathcal{G}_T$, the cliques of which are $\mathcal{V}_T^1$ and $\mathcal{V}_T^2$ respectively in $\mathcal{V}_T$, let $\mathcal{V}^1$ and $\mathcal{V}^2$ be the sets of nodes in $\mathcal{V}$ that belong to cliques in $\mathcal{V}_T^1$ and $\mathcal{V}_T^2$ respectively, then $|\mathcal{V}^1 \cap \mathcal{V}^2| \leq 1$;
3. let $\mathcal{T}$ be any tree of $\mathcal{G}_T$ and let $\mathcal{V}^{\mathcal{T}}$ be the set of nodes of $\mathcal{V}$ contained in the cliques of $\mathcal{T}$, then all pairs of nodes of $\mathcal{V}^{\mathcal{T}}$ are connected in $\mathcal{G}$ by trails passing only through nodes of $\mathcal{V}^{\mathcal{T}}$;
4. for every $Z \in \mathcal{V} \backslash \mathcal{V}''$ and every $C_Z^i, C_Z^j \in \mathcal{C}_Z$, C_Z^i and C_Z^j are different and belong to different trees of $\mathcal{G}_T$.

Clearly these four properties hold before entering the `while` loop of lines 02–33 for the first time. Assume now that these properties hold until a given node $X \in \mathcal{V}$ is selected on line 03 and let us show that they still hold the end of the corresponding `while` loop.

First note that the properties 1) and 4) still hold when reaching line 09. Indeed, NC is defined either on line 04 or on line 06. If it is on line 06, then as C_X is removed from $\mathcal{C}_X$ and as all the elements of $\mathcal{C}_X$ belong to different trees, NC and all the C_X^i's of $\mathcal{C}_X$ are different and belong to different trees. Moreover, as $\mathcal{G}_T$ and the $\mathcal{C}_Z$'s, for every $Z \neq X$, are kept unchanged, properties 1) and 4) still hold on line 09. If NC is defined on line 04, then no $C_X^i \in \mathcal{C}_X$ can be equal to NC (else it would have been defined on line 06). Moreover, there does not exist any clique NC in $\mathcal{G}_T$, else as function **finalize_JT** never selects a given node more than once on line 03, clique NC would have been created during the classical triangulation phase, which is impossible because NC would have then belonged to $\mathcal{C}_X$ (by the triangulation algorithm) and so NC would have been defined on line 06 instead of line 04. Consequently, the operation performed on line 07 adds a new clique NC to $\mathcal{G}_T$ with no adjacent edge, so NC belongs to a tree different from those of the C_X^i's, and properties 1) and 4) still hold.

Similarly, when reaching line 09, property 3) holds since the only modification that can be issued on $\mathcal{G}_T$ is the addition of clique NC on line 07, NC being constituted by X and its parents. As NC is not linked to any other node in $\mathcal{G}_T$, property 3) holds since X and its parents in $\mathcal{G}$ are obviously connected in $\mathcal{G}$ by trails passing only through nodes of NC.

The operations performed on lines 09–19 consist in substituting some nodes C_X^1, $C_X^2, \ldots, C_X^k$ of $\mathcal{G}_T$ by node NC (and in updating edges adjacent to the C_X^i's accordingly). For convenience of notation consider that nodes C_X^i's are replaced in i-increasing order. No cycle can be created after C_X^1 has been replaced by NC simply because, by the preceding paragraph, NC and C_X^1 belong to different trees; moreover as the C_X^i's, $i > 1$, belonged to trees of $\mathcal{G}_T$ different from those containing NC and C_X^1, after C_X^1's substitution all the C_X^i's, $i > 1$, still belong to trees of $\mathcal{G}_T$ different from that containing NC. Assume there exists a node $Z \in \mathcal{V} \backslash \mathcal{V}''$ such that $C_Z^1, C_Z^2 \in \mathcal{C}_Z$ belonged to different trees before C_X^1's substitution and to only one tree after that substitution. Then C_Z^1 and C_Z^2 were in the same tree as NC and C_X^1 respectively before substitution. For NC to be in the same tree as another node before substitution, NC must have been created on line 06, but then Z and X both belonged to 2 different trees before performing the `while` loop, which is impossible since property 2) held at that time. Assume that until after C_X^{i-1} has been replaced there exists no cycle and that NC and all the C_X^j's, $j \geq i$, belong to different trees. If the substitution of C_X^i by NC induces a cycle in $\mathcal{G}_T$, then a node C_Y adjacent to C_X^i would be connected to NC before substitution, and consequently C_X^i would also have been connected to NC, which is impossible by induction hypothesis. Moreover, NC and the C_X^j's, $j > i$, belong to different trees else there already existed either a trail in $\mathcal{G}_T$ between a C_X^j and NC or a trail between a C_X^j and C_X^i, which are both impossible according to the induction hypothesis. If there existed $Z \in \mathcal{V} \backslash \mathcal{V}''$ such that $C_Z^1, C_Z^2 \in \mathcal{C}_Z$ belonged to different trees before C_X^i's substitution and to only one tree after that substitution, then C_Z^1 and C_Z^2 were in the same tree as NC and some C_X^j.

But as loop 09–19 only substitutes some C_X^k's by NC, this means that at the beginning of the `while` loop C_Z^1 and C_Z^2 belonged to the same trees as some C_X^j and some C_X^k, $j \neq k$, respectively which is impossible since Z and X would both have belonged to 2 different trees, hence contradicting property 2). Hence properties 1) and 4) still hold when reaching line 20. Property 3) also holds. Indeed, all the trees that are merged into one single tree by lines 09–19 contain node X of $\mathcal{G}$, hence by property 3), all the nodes of each tree were connected to X before substitutions, so the nodes of the union of these trees were connected to each other, as well as with X's parents. Consequently, property 3) still holds on line 20.

Let us now show that lines 20–23 preserve properties 1) and 4). First remark that as the C_X's in $\mathcal{C}_X$ are all different from NC (this results from lines 6 and 11-13), no loop (NC, NC) can be added on lines 20–23. Second, as NC and all the C_X's of $\mathcal{C}_X$ belong to different trees when reaching line 20, adding edges (NC, C_X) cannot create any cycle. If property 4) was not preserved, there would exist $Z \in \mathcal{V} \backslash \mathcal{V}''$ such that $C_Z^1, C_Z^2 \in \mathcal{C}_Z$ belonged to different trees before the loop and to only one tree after. But then at the beginning of the `while` loop C_Z^1 and C_Z^2 would belong to the same trees as some C_X^j and some C_X^k, $j \neq k$, respectively which is impossible since Z and X would both have belonged to 2 different trees, hence contradicting property 2). Consequently, properties 1) and 4) still hold when reaching line 25. Obviously for the same reason as in the preceding paragraph, property 3) is preserved by lines 20–25.

Property 2) is also preserved when reaching line 25. Assume it were not true, then there would exist two trees, both containing at least two different nodes of $\mathcal{V}$, say A and B. Clearly, that containing NC is one of them. Call $\mathcal{T}$ the other tree. If A or B is equal to X then either X and some other nodes contained in $\mathcal{T}$ form a cycle, or no cycle of nodes of $\mathcal{T}$ passes through X. The former case is impossible since the classical triangulation algorithm would have added a clique C_X of $\mathcal{T}$ in $\mathcal{C}_X$ and $\mathcal{T}$ would have been merged with the tree containing NC by lines 09–23. In the second case, as function **finalize_JT** never selects more than once node X on line 03, a clique containing X can only be added to $\mathcal{T}$ during the execution of the `while` loop of lines 02–33 corresponding to a child Y of X. But then, as X is not in a loop within $\mathcal{T}$, arc (X, Y) does not belong to any $\mathcal{B}_i$ and so line 27 adds the NC corresponding to Y in $\mathcal{C}_X$. Consequently, $\mathcal{C}_X$ contains a clique belonging to $\mathcal{T}$ which is impossible since, then, $\mathcal{T}$ should have been merged with the tree containing the NC of Y during the `while` loop of lines 02–33 related to X. Hence neither A nor B can be equal to X. If, before the `while` loop, A belonged to one tree and B belonged to another, both trees being merged with NC, then there was a trail between A and X in the first tree and another trail between B and X in the second one. These trails have no node in common except X by property 2) since both trees already contained X. As A and B are also contained in $\mathcal{T}$, there is a trail between A and B not passing through X since $\mathcal{T}$ is not merged with NC. Consequently, there was a cycle passing through A, B and X, which is impossible since by the classical triangulation, there should exist a tree in $\mathcal{G}_T$ containing these 3 nodes, hence contradicting the fact that property 2) was preserved before entering the `while` loop of lines 02–33. The last possible case is that at least one of the nodes, say A, was not yet contained in any tree of $\mathcal{G}_T$ before the merging. Then it can only be a parent of X that does not belong to any cycle. As we know that A and B are connected within $\mathcal{T}$, that B and X are connected within the tree containing NC and that there exists an arc (A, X) in $\mathcal{G}$, there exists a cycle in $\mathcal{G}$ passing through A, hence a contradiction. Consequently, in all cases, the violation of property 2) leads to a contradiction, hence it still holds on line 23.

To complete this proof, there remains to show that the four properties hold when reaching line 33. As from line 25 on, $\mathcal{G}_T$ is kept unchanged, properties 1), 2) and 3) trivially hold. As property 4) is known to be preserved until line 25, its violation can only be the consequence of the execution of line 26. But this line is performed only when arc (Y, X) does not belong to any $\mathcal{B}_i$, i.e., when arc (Y, X) belongs to no cycle. This implies that no cycle containing Y also contains X and, moreover, no cycle containing Y has any intersection Z with the tree containing NC, as defined when arrived on line 23, else there would exist a cycle passing through Z, Y and X, implying that arc (Y, X) belongs to a cycle. Consequently, the cliques in $\mathcal{C}_Y$ do not belong to the tree containing NC. Hence adding NC to $\mathcal{C}_Y$ preserves property 4). ♦

Proposition 8.4 *After performing* **finalize_JT***, $\mathcal{G}_T$ is a junction tree.*

Proof of proposition 8.4: We shall prove that at each pass of the outer `while` loop, the following two properties hold: 1) for any $Z \in \mathcal{V}''$, there exists exactly one tree in $\mathcal{G}_T$ the cliques of which contain Z; and 2) the running intersection property (RIP) holds in any tree of $\mathcal{G}_T$. Of course both properties hold the first time we reach the outer `while` loop since $\mathcal{V}'' = \emptyset$ and all the trees of $\mathcal{G}_T$ are constructed by a classical triangulation algorithm. Assume that these properties hold until a pass of the `while` loop in which some node X is assigned on line 03. Clearly, by lines 12 and 22, all the C_X's of $\mathcal{C}_X$ are linked to NC, which implies that all the trees containing some C_X form only one singly-connected graph (a tree by lemma 8.2). Assume at the end of the `while` pass there exists another tree $\mathcal{T}$ that contains a clique containing X. Then this clique, as well as any clique in $\mathcal{T}$, did not belong to $\mathcal{C}_X$. By the classical triangulation algorithm, this implies that X does not belong to any cycle of nodes of $\mathcal{T}$. But then the clique(s) of $\mathcal{T}$ containing X must have been added to $\mathcal{T}$ by function **finalize_JT**. This cannot be during the pass related to X else $\mathcal{T}$ would be connected to NC. Hence it has to be during a pass related to a child, say Y, of X. Thus clique $NC(Y)$, the NC clique created during the `while` pass related to Y, belongs to $\mathcal{T}$. By property 2) of the proof of lemma 8.2, there does not exist any other tree of $\mathcal{G}_T$ containing both X and Y. Hence since X does not belong to any cycle in $\mathcal{T}$, there exists no i such that arc $(X, Y) \in \mathcal{B}_i$. Consequently, line 27 shall add $NC(Y)$ to C_X and during the `while` pass related to X, $\mathcal{T}$ should thus be linked with $NC(X)$, a contradiction. Hence property 1) still holds after the `while` loop related to X.

By property 2) of the proof of lemma 8.2, at each step of the algorithm, two different trees of $\mathcal{G}_T$ have at most one node of $\mathcal{G}$ in common. At each pass of the outer `while` loop, some trees that contain X are merged. As they all contain X, they have no other node in common. Hence RIP cannot fail because a node, say A, of $\mathcal{G}$ belongs to two to-be-merged trees, say $\mathcal{T}^1$ and $\mathcal{T}^2$, and A is not on the trail between one clique of $\mathcal{T}^1$ and one of $\mathcal{T}^2$ containing A. Consequently, RIP can fail only because there exists a clique C in a tree $\mathcal{T}$ (merged with that of NC) containing a node B of $\mathcal{G}$ also belonging to NC, and B does not belong to the trail between C and NC. Now, let C_X be the clique in $\mathcal{C}_X$ belonging to $\mathcal{T}$. By construction, C_X contains X and all of X's parents in $\mathcal{V}^{\mathcal{T}}$, where $\mathcal{V}^{\mathcal{T}}$ is the set of nodes of $\mathcal{G}$ contained in cliques of $\mathcal{T}$. So B not only belongs to NC but also to C_X. But this is impossible since by induction hypothesis, property 2) holding before X is selected on line 03, RIP holds in $\mathcal{T}$ and in particular between C and C_X. Hence property 2) also holds after each pass of the outer `while` loop on every tree.

By property 1), for any node $Z \in \mathcal{V}''$, there exists only one tree in $\mathcal{G}_T$ containing Z. Since, by lemma 8.1, after performing function **finalize_JT**, $\mathcal{V}''$ contains all the nodes in $\mathcal{G}$, and since by lemma 8.2 $\mathcal{G}_T$ has no cycle, $\mathcal{G}_T$ is a single tree. By property 2), RIP holds in every tree of $\mathcal{G}_T$ during the execution of the function. In particular, it holds when all the nodes of $\mathcal{G}$ have been added to $\mathcal{V}''$, i.e., when $\mathcal{G}_T$ is a tree. Hence after executing function **finalize_JT** $\mathcal{G}_T$ satisfies the running intersection property. By lines 09–19 and the fact that at each step of the algorithm trees have no more than one node of $\mathcal{G}$ in common, no node C_i of $\mathcal{G}_T$ is a subset of another node C_j of $\mathcal{G}_T$. Consequently,

$\mathcal{G}_T$ is a junction tree. ♦

Proof of proposition 8.1: First note that all the nodes and edges in $\mathcal{G}_{\mathcal{B}_i}$'s belong to cycles. Indeed, assume some node X were not in a cycle, then if it is connected in $\mathcal{G}_{\mathcal{B}_i}$ to at least two nodes, say Y and Z, removing X would disconnect $\mathcal{G}_{\mathcal{B}_i}$ since Y and Z would not be connected anymore, which is impossible since $\mathcal{G}_{\mathcal{B}_i}$ is a biconnected graph. X cannot have no adjacent node since $\mathcal{G}_{\mathcal{B}_i}$ is constructed from set of edges $\mathcal{B}_i$. If X has only one adjacent node Y, then as $|\mathcal{B}_i| > 1$, there exists at least another node Z in $\mathcal{G}_{\mathcal{B}_i}$ and removing Y would disconnect Z and X, impossible. Hence all nodes belong to cycles. For the same reason, all edges belong to cycles in $\mathcal{G}_{\mathcal{B}_i}$'s.

Let $\mathcal{G}'_{\mathcal{B}_i} = (\mathcal{V}_{\mathcal{B}_i}, \mathcal{E}_{\mathcal{B}_i})$ be the graph resulting for the application of step 1) on $\mathcal{G}_{\mathcal{B}_i} = (\mathcal{V}_{\mathcal{B}_i}, \mathcal{A}_{\mathcal{B}_i})$. By definition, $\mathcal{G}'_{\mathcal{B}_i}$ is triangulated. Let $\mathcal{G}'$ be the union of all the $\mathcal{G}'_{\mathcal{B}_i}$'s, i.e., $\mathcal{G}' = (\cup_i \mathcal{V}_{\mathcal{B}_i}, \cup_i \mathcal{E}_i)$. Then $\mathcal{G}'$ is triangulated. Indeed, for any couple of distinct biconnected graphs $(\mathcal{G}_{\mathcal{B}_i}, \mathcal{G}_{\mathcal{B}_j})$, $|\mathcal{V}_{\mathcal{B}_i} \cap \mathcal{V}_{\mathcal{B}_j}| \leq 1$ else there would exist two nodes, say A and B, both belonging to $\mathcal{V}_{\mathcal{B}_i}$ and $\mathcal{V}_{\mathcal{B}_j}$. But then graph $(\mathcal{V}_{\mathcal{B}_i} \cup \mathcal{V}_{\mathcal{B}_j}, \mathcal{B}_i \cup \mathcal{B}_j)$ would be a biconnected graph: indeed, by definition removing node A (resp. B) can disconnect neither the other nodes in $\mathcal{V}_{\mathcal{B}_i}$ nor the other nodes in $\mathcal{V}_{\mathcal{B}_j}$ since they are still connected to B (resp. A). And removing a node in $\mathcal{V}_{\mathcal{B}_i} \backslash \mathcal{V}_{\mathcal{B}_j}$ can disconnect neither the nodes in $\mathcal{V}_{\mathcal{B}_i}$ (since $\mathcal{B}_i$ is a biconnected component) nor the nodes in $\mathcal{V}_{\mathcal{B}_j}$ since $\mathcal{B}_j$ is unaffected by the removal. By symmetry the proof is similar when removing a node in $\mathcal{V}_{\mathcal{B}_j} \backslash \mathcal{V}_{\mathcal{B}_i}$. Hence as the difference between $\mathcal{G}_{\mathcal{B}_i}$ and $\mathcal{G}'_{\mathcal{B}_i}$ is that $\mathcal{G}'_{\mathcal{B}_i}$ has more edges than $\mathcal{G}_{\mathcal{B}_i}$, for any couple $(\mathcal{G}'_{\mathcal{B}_i}, \mathcal{G}'_{\mathcal{B}_j})$, $|\mathcal{V}_{\mathcal{B}_i} \cap \mathcal{V}_{\mathcal{B}_j}| \leq 1$. Consequently, there exists no cycle containing some nodes of $\mathcal{G}'_{\mathcal{B}_i}$ and some nodes of $\mathcal{G}'_{\mathcal{B}_j}$. Thus for any $i \in \{2, \ldots, k\}$, the union of $\mathcal{G}_{\mathcal{B}_i}$ and of the union of the $\mathcal{G}_{\mathcal{B}_j}$'s, $j < i$, is triangulated, and so $\mathcal{G}'$ is triangulated.

Let $\mathcal{V}' = \mathcal{V} \backslash (\cup_i \mathcal{V}_{\mathcal{B}_i})$ and let $\mathcal{A}' = \mathcal{A} \backslash (\cup_i \mathcal{B}_i)$. Of course after adding nodes of $\mathcal{V}'$ to $\mathcal{G}'$ the resulting graph is still triangulated, and similarly as $\mathcal{A}'$ is the set of edges not belonging to any cycle (edges belonging to cycles belong to biconnected components of sizes strictly greater than 1, hence to some $\mathcal{B}_i$) after adding $\mathcal{A}'$ to $\mathcal{G}'$ the resulting graph, say $\mathcal{G}''$, is still triangulated.

Assume that, after performing the moralizations of step 2) on some nodes of $\mathcal{G}$, $\mathcal{G}''$ is triangulated and consider adding the moralization edges of some node X to $\mathcal{G}''$ on step 2). If the resulting graph is no more triangulated, then there exist two parents of X, say Y and Z, and a chordless cycle of length 4 or more, say $\mathcal{C}$, passing through Y and Z. By moralization, Y and Z are adjacent, so the cycle passes through edge (Y, Z) (else (Y, Z) is a chord). If the cycle passes through X, then either (Y, X) and/or (Z, X) is a chord or $\mathcal{C}$ is of length 3. If the cycle does not pass through X, then it cannot pass through any other X's parents else some moralization edge would be a chord. Before X's parents moralization, there already existed an arc from Y to X and another from Z to X. Replace edge (Y, Z) in the cycle by edges (Y, X) and (Z, X), then by hypothesis, this new cycle $\mathcal{C}' = \{A_1 = Y, A_2 = X, A_3 = Z, \ldots, A_{r-1}, A_r = Y\}$ has a chord. It cannot be (Y, Z) else (Y, Z) already existed before moralization and $\mathcal{C}$ would have a chord. If the chord was not adjacent to X, then this is also a chord for $\mathcal{C}$, a contradiction. So all the chords of $\mathcal{C}'$ are adjacent to X. Let us now prove that in this case Y and Z

already belonged to some $\mathcal{G}_{\mathcal{B}_i}$ and thus have been moralized on step 1), hence edge (Y, Z) already existed before step 2) and $\mathcal{C}$ has a chord. If edge (A_3, A_4) did not belong to $\mathcal{A}$, then either it has been added by moralization (either on step 1) or on step 2)), but then both A_3 and A_4 have a common child, say B (different from X else A_4 would be one of X's parent), in $\mathcal{G}$ and after adding B between A_3 and A_4 all edges from A_1 to A_4 belong to $\mathcal{G}$, or edge (A_3, A_4) has been added as a chord by triangulation of step 1). In this case, there exists a trail not passing though X between A_3 and A_4 in $\mathcal{G}$. Add all nodes on this trail to $\mathcal{C}'$ between A_3 and A_4. Note that this trail does not pass through Y and Z since edge (Y, Z) is not in any $\mathcal{B}_i$. Perform the same process to all pairs (A_i, A_{i+1}) in $\mathcal{C}'$. Then $\mathcal{C}'$ is a sequence of nodes going from Y to Y along edges of $\mathcal{A}$. Hence there exists a cycle passing though Y and Z in $\mathcal{G}$, so edge (Y, Z) is added to $\mathcal{G}''$ by moralization on step 1), a contradiction. Consequently, moralizing on step 2) cannot fail to preserve triangulatedness.

Hence the graph resulting of the application of steps 1) and 2) is triangulated. As step 3) removes the edges directions, graph $\mathcal{G}_*$ is a triangulated undirected graph. ♦

Chapter 9

Conclusion

Probabilities have become a cornerstone of many domains, including Artificial Intelligence, Economics, Operations Research and even Psychology. However, until the end of the 80's, computing probabilities involving many different interdependent random variables was impractical because: i) the amount of storage required for such probabilities was too high; and ii) the computations themselves were too big to be performed in a reasonable amount of time. In the 80's, Pearl [Pea88] introduced a directed graphical structure called a Bayesian network that made both the storage and the computations manageable. At about the same time, Lauritzen and Spiegelhalter and, then, Jensen developed another graphical structure —this one undirected— that was shown to outperform Pearl's model.

It is now common knowledge that Jensen's algorithm, and more generally join tree (Shafer-Shenoy) or junction tree-based algorithms (Jensen), propagate informations faster than Pearl's algorithm because the former deal much more efficiently with cycles in the network than the latter and because such cycles are quite usual in practice. Unfortunately the arc directions of a Bayesian network bring some useful informations that are, at least to some extent, lost in join trees. As a consequence, Jensen or Shafer-Shenoy may have to perform some computations that can be shown to be unnecessary using arc directions. Thus these algorithms may be improved by just making the informations about arc directions available to them. Conversely, if Pearl's algorithm could be modified so as to deal with cycles more or less as Jensen's algorithm, then it would be greatly improved. These remarks suggest that, studying the relationships between directed and undirected graphs, and between directed and undirected propagation algorithms, some kind of unification of these methods may emerge that would improve all of them. This unification has been the main topic of this thesis.

To improve these methods, we first had to identify the weaknesses and the strengths of both undirected and directed algorithms. This was summarized in Chapter 5. Briefly, the main advantage of Pearl lies in the *d*-separation analysis, that is, a probabilistic independence property encoded in directed graphs that enables to identify qualitatively which computations are required to calculate a posteriori probabilities. The main advantage of Jensen's or Shafer-Shenoy's method lies in the secondary structure used by

these algorithms and that copes efficiently with cycles in the Bayes net. This efficiency derives from the fact that this structure is the result of a triangulation process.

Hence, were we to improve propagation algorithms, we should try to find a way to create a directed secondary structure using some kind of triangulation algorithm. The result being directed, *d*-separation may thus be used to avoid unnecessary computations, and the triangulatedness would keep the remaining computations very efficient. This led us in Chapter 6 to propose a new directed secondary structure we called a *triangulated Bayesian network* or TBN, and a new variant of Pearl's algorithm. We showed that this variant has the same computational complexity as Jensen's or Shafer-Shenoy's algorithms. Of course even if some algorithms have the same computational complexity, one of them may be twice or thrice as fast as another (since multiplicative constants are not taken into account in computational complexity) hence our algorithm may prove to be slower than Jensen's in practical situations. In Chapter 7, we modified it using the same idea as Shenoy's binary join tree. This resulted in an algorithm that is as fast as Jensen or Shafer-Shenoy. The only difference with the latter is that our algorithm relies on a directed graph and that, in such graphs, *d*-separation can be used to reduce the computation burden of propagation.

Finally, as our method now also relies on a graphical structure resulting from a triangulation technique, it may be wondered whether the latter cannot benefit from our unification between directed and undirected propagation methods. The answer is positive and is detailed in Chapter 8. In join tree-based algorithms, it is usual to moralize the Bayesian network before triangulating it. This step is not actually compulsory but it ensures that the conditional probabilities initially stored in the Bayesian network can also be stored in the join tree. So all undirected propagation methods first moralize the BN and, then triangulate it and compile the result into a join tree. As we perform our triangulation in a directed graph, the problem that would arise without moralization cannot arise and we can see that it is possible to postpone moralization after several steps of triangulation. Doing so, we can show that the join trees obtained are smaller and, as a consequence, the propagation algorithms are faster. The gain is shown to be important in some practical situations.

To conclude, we have shown in this thesis how undirected and directed propagation methods in Bayesian networks could be unified. Doing so, we have been able to use the advantage of both kinds of methods to improve the efficiency of the propagations.

Bibliography

[ACP87] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[Ata99] M. Atallah, editor. *Algorithms and theory of computation handbook*. CRC Press, 1999.

[BG96a] A. Becker and D. Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithm for the vertex feedback set problem. *Artificial Intelligence*, 83:167–188, 1996.

[BG96b] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, pages 81–89. Morgan Kaufmann, 1996.

[BYGNR94] R. Bar-Yehuda, D. Geiger, J. Naor, and R.M. Roth. Approximation algorithms for the vertex feedback set problem with application to constraints satisfaction and Bayesian inference. In *Proceedings of the 5th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 344–354, 1994.

[CDLS99] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

[dF74] B. de Finetti. *Theory of Probability*. Wiley, 1974.

[Die96] F.J. Diez. Local conditioning in Bayesian networks. *Artificial Intelligence*, 87:1–20, 1996.

[FHKR95] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The batmobile: Towards a Bayesian automated taxi. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1878–1885, 1995.

[FJ00] A. Faÿ and J.-Y. Jaffray. A justification of local conditioning in Bayesian networks. *International Journal of Approximate Reasoning*, 24(1):59–81, 2000.

[FSMB91] R.C.G. Franklin, D.J. Spiegelhalter, F. Macartney, and K. Bull. Evaluation of an algorithm for neonates. *British Medical Journal*, 302:935–939, 1991.

[HT73] J.E. Hopcroft and R.E. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372–378, 1973.

[Ish81] V. Isham. An introduction to spatial point processes and Markov random fields. *International Statistical Review*, 49:21–43, 1981.

[Jam96] A.J. Jameson. Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interaction*, 5:193–251, 1996.

[Jen96] F.V. Jensen. *An Introduction to Bayesian Networks*. Springer-Verlag, North America, 1996.

[JJ94] F.V. Jensen and F. Jensen. Optimal junction trees. In R.L Mantaras and D. Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Articial Intelligence*, pages 360–366. Morgan Kaufmann, 1994.

[JLO90] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[Kjæ90] U. Kjærulff. Triangulation of graphs — algorithms giving small total state space. Technical Report R-90-09, Dept. of Mathematics and Computer Science, Aalborg University, 1990.

[KP83] J.H. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in inference engines. In *Proceedings of IJCAI'83*, Karlsruhe, Germany, 1983.

[Lau82] S.L. Lauritzen. *Lectures on Contingency Tables*. University of Aalborg Press, 2nd edition, 1982.

[LS88] S.L. Lauritzen and D.J. Spiegelhalter. 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):157–224, 1988.

[Mel87] K. Mellouli. *On the propagation of beliefs in networks using the Dempster-Shafer theory of evidence*. PhD thesis, University of Kansas, 1987.

[MG96] R.J. Mislevy and D.H. Gitomer. The role of probability-based inference in an intelligent tutoring system. *User Modeling and User-Adapted Interaction*, 5:253–282, 1996.

[MJ99] A.L. Madsen and F.V. Jensen. Lazy propagation: A junction tree inference algorithm based on lazy inference. *Artificial Intelligence*, 113:203–245, 1999.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman Publishers, inc, 1988.

[PP85] J. Pearl and A. Paz. Graphoids: A graph-based logic for reasoning about relevance relations. In B. Du Boulay, D. Hogg, and L. Steels, editors, *Advances in Artificial Intelligence 2*. North Holland, 1985.

[PS91] M.A. Peot and R.D. Shachter. Fusion and propagation with multiple observations in belief networks. *Artificial Intelligence*, 48:299–318, 1991.

[PW95] D.V. Pynadath and M.P. Wellman. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 472–481, 1995.

[Ros70] D.J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

[RTL76] D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM journal on Computing*, 5:266–283, 1976.

[SAS94] 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*, 1994.

[SC90] 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.

[SG97] K. Shoikhet and D. Geiger. Finding optimal triangulations via minimal vertex separators. In *Proceedings of AAAI-97*, 1997.

[Sha76] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey, 1976.

[Sha86] R.D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, 1986.

[Sha96] G. Shafer. *Probabilistic expert systems*. Society for Industrial and Applied Mathematics, 1996.

[She97] P.P. Shenoy. Binary join trees for computing marginals in the shenoy-shafer architecture. *International Journal of Approximate Reasoning*, 17(1):1–25, 1997.

[Sho76] E.H. Shortliffe. *Computer-Based Medical Consultation: MYCIN*. Elsevier, 1976.

[SS90a] G. Shafer and P.P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[SS90b] P.P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. In R.D. Shachter, T.S. Levitt, J.F. Lemmer, and L.N. Kanal, editors, *Proceedings of Uncertainty in Artificial Intelligence*, pages 169–198, 1990.

[Tar72] R.E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[Yan81] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM journal on Algebraic and Discrete Methods*, 2:77–79, 1981.