COMPUTATIONAL STATISTICS & DATA ANALYSIS

Computational Statistics & Data Analysis 34 (2000) 387–413

www.elsevier.com/locate/csda

Importance sampling in Bayesian networks using probability trees ☆

Antonio Salmerón[a,*], Andrés Cano[b], Serafín Moral[b]

[a]*Department of Statistics and Applied Mathematics, University of Almería, Spain*
[b]*Department of Computer Science and Artificial Intelligence, University of Granada, Spain*

Received 1 April 1999; received in revised form 1 October 1999; accepted 26 November 1999

Abstract

In this paper a new Monte-Carlo algorithm for the propagation of probabilities in Bayesian networks is proposed. This algorithm has two stages: in the first one an approximate propagation is carried out by means of a deletion sequence of the variables. In the second stage a sample is obtained using as sampling distribution the calculations of the first step. The different configurations of the sample are weighted according to the importance sampling technique. We show how the use of probability trees to store and to approximate probability potentials, and a careful selection of the deletion sequence, make this algorithm able to propagate over large networks with extreme probabilities. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Bayesian networks; Simulation; Importance sampling; Probability trees; Approximate pre-computation

1. Introduction

It is known that exact probabilistic inference in Bayesian networks may be infeasible in large networks (Cooper, 1990). This motivates the development of approximate algorithms, most of them based on Monte Carlo simulation.

☆ This work has been supported by CICYT under projects TIC97-1135-C04-01 and TIC97-1135-C04-02.
* Corresponding author.
E-mail addresses: asc@stat.ualm.es (A. Salmerón), acu@decsai.ugr.es (A. Cano), smc@decsai.ugr.es (S. Moral).

We can distinguish two groups of Monte Carlo algorithms: those based on Gibbs sampling (Jensen et al., 1995; Pearl, 1987) and those based on importance sampling (Dagum and Luby, 1997; Fung and Chang, 1990; Henrion, 1988; Hernández et al., 1996,1998; Shachter and Peot, 1990). However, when dealing with very large networks with extreme probabilities, only the most sophisticated of them are able to provide accurate results, namely blocking Gibbs sampling (Jensen et al., 1995) and importance sampling based on approximate pre-computation (Hernández et al., 1996,1998). In both cases, the goal is to draw samples from a probability distribution that is difficult to manage, in the sense that its size is too big.

Some importance sampling algorithms such as likelihood weighting (Fung and Chang, 1990; Shachter and Peot, 1990) or backward simulation (Fung and Favero, 1994) start simulating values for some variables of the graph without taking into account the information stored in other parts of the graph. For example, likelihood weighting starts simulating the root nodes, without considering how these values are affected by the observations in other parts of the graph. Backward simulation tries to overcome this problem starting to simulate in the nodes observed, but again this problem appears when we continue obtaining values for the parents of the nodes observed: the information in other parts of the graph is not taken into account. In the extreme case, when the values obtained are incompatible with the observations or the 'a priori' probabilities then the configuration obtains a weight of value 0 and has to be discarded. An extension of likelihood weighting is the bounded-variance algorithm, described in Dagum and Luby (1997). This method provides very good approximations in polynomial time for a wide class of networks: those without extreme probabilities. However, if extreme probabilities are present, the problem described above can arise. In fact, Dagum and Luby (1993) proved that the problem of approximating probabilities in Bayesian networks is NP-hard in the worst case. This motivates the study of heuristic procedures to make inferences in large networks with extreme probabilities.

A class of these heuristic procedures is composed by the importance sampling algorithms based on approximate pre-computation. These methods perform first a fast but non-exact propagation, following a node removal process (Zhang and Poole, 1996). In this way, an approximate 'a posteriori' distribution is obtained. In a second stage, a sample is drawn using the approximate distribution and the probabilities are estimated according to the importance sampling methodology. A similar idea can be found in consistency algorithms for propositional logic (Dechter and Rish, 1994), when a bounded directional resolution procedure is followed by a Davis–Putnam backtracking algorithm. The bounded directional resolution is as the approximate probabilistic computation, and the Davis–Putnam backtracking uses the previous approximate computations in the same way as the sample is obtained from the sampling distribution.

Known approximate pre-computation algorithms, proposed in Hernández et al. (1996,1998), use probability tables (see Jensen, 1996) to represent the sampling distributions. The main problem of the probability tables representation is that the size of a table is proportional to the product of the number of possible values of each of the variables for which the potential is defined, with independence of the possible existence of regularities or repetitions.

Under limited resources, it turns out necessary to find a method able to concentrate more information in less space. To this end, two kinds of sparse representations have been used in the context of Bayesian networks: rule bases (Poole, 1997a,b) and *probability trees* (Boutilier et al., 1996; Cano and Moral, 1997; Kozlov and Koller, 1997). They both try to use the regularities of the conditional distributions, i.e., *context-specific independence* (Boutilier et al., 1996) to simplify the network in order to facilitate the inference procedures, as in Zhang and Poole (1996).

The advantage of these methods over probability tables is more important when we cannot afford to compute exact values and we have to approximate the potentials. Probability trees (rule bases too) have the possibility of approximating in an asymmetrical way, concentrating more resources (finer discrimination) where they are more necessary: higher values with more variability.

In this paper we present a Monte Carlo algorithm based on pre-computation, in which the approximation is based on the probability tree representation. Computations are carried out directly over probability trees. The performance of the resulting algorithm is compared with previous importance sampling based on pre-computation with the probability tables representation, showing how the new methods reduce both the error and computing time.

We start off establishing some notation and the concept of probability propagation in Section 2. Approximate propagation by means of importance sampling technique is analyzed in Section 3, including a very simple motivating example. Probability trees are studied in Section 4, including tree construction and operations necessary to perform probability propagation using them. In Section 5 an importance sampling algorithm using probability trees is proposed, pointing out the advantages of this new representation. Section 6 is devoted to describe the experimental tests carried out for testing the algorithm proposed, and to discuss their results. The paper ends with conclusions in Section 7.

2. Notation and problem formulation

A *Bayesian network* is a directed acyclic graph where each node represents a random variable, and the topology of the graph shows the independence relations among the variables, according to the d-separation criterion (Pearl, 1988). Given the independences attached to the graph, the joint distribution is determined giving a probability distribution for each node conditioned on its parents.

Let $A=\{X_1,\dots,X_n\}$ be the set of variables in the network. Assume each variable X_i takes values on a finite set U_i. For any set U, $|U|$ stands for the number of elements it contains. If I is a set of indices, we will write X_I for the set $\{X_i \mid i\in I\}$. $N=\{1,\dots,n\}$ will denote the set of indices of all the variables in the network; thus, $X_N=A$. We will denote by U_I the Cartesian product $\prod_{i\in I} U_i$. Given $x\in U_I$ and $J\subseteq I$, x_J will denote the element of U_J obtained from x dropping the coordinates not in J.

A potential f defined on U_I will be a mapping $f: U_I \longrightarrow \mathbb{R}_0^+$, where $\mathbb{R}_0^+$ is the set of non-negative real numbers. Probabilistic information (including 'a priori', conditional and 'a posteriori' distributions) will always be represented by means of potentials, as in Lauritzen and Spiegelhalter (1988).

If f is a potential defined on U_I, $s(f)$ will denote the set of indices of the variables for which f is defined (i.e. $s(f) = I$).

The *marginal* of a potential f over a set of variables X_J with $J \subseteq I$ is denoted by $f^{\downarrow J}$. The conditional distribution of each variable X_i, $i = 1, \ldots, n$, given its parents in the network, $X_{F(i)}$, is denoted by a potential $p_i(x_i|x_{F(i)})$ where p_i is defined over $U_{i \cup F(i)}$.

Then, the joint probability distribution for the n-dimensional random variable X_N can be expressed as

$$p(x) = \prod_{i \in N} p_i(x_i|x_{F(i)}) \quad \forall x \in U_N. \tag{1}$$

An *observation* is the knowledge about the exact value $X_i = e_i$ of a variable. The set of observations will be denoted by e, and called the *evidence set*. E will be the set of indices of the variables observed.

Every observation, $X_i = e_i$, is represented by means of a potential which is a Dirac function defined on U_i as $\delta_i(x_i; e_i) = 1$ if $e_i = x_i$, $x_i \in U_i$, and $\delta_i(x_i; e_i) = 0$ if $e_i \neq x_i$.

The goal of probability propagation is to calculate the 'a posteriori' probability function $p(x'_k|e)$, for every $x'_k \in U_k$, where $k \in \{1, \ldots, n\}$. This probability could be obtained from the joint distribution (1), but we assume that it is difficult to manage due to its size, since we are interested in large networks and the number of values necessary to specify the joint distribution grows exponentially in the number of variables in the network. Notice that $p(x'_k|e)$ is equal to $p(x'_k, e)/p(e)$, and, since $p(e) = \sum_{x'_k \in U_k} p(x'_k, e)$, we can calculate the *posterior* probability if we compute the value $p(x'_k, e)$ for every $x'_k \in U_k$, normalizing afterwards. $p(x'_k, e)$ can be expressed in the following way:

$$\begin{aligned} p(x'_k, e) &= \sum_{\substack{x \in U_N \\ x_E = e \\ x_k = x'_k}} \prod_{i \in N} p_i(x_i|x_{F(i)}) \\ &= \sum_{x \in U_N} \left(\prod_{i \in N} p_i(x_i|x_{F(i)}) \right) \left(\prod_{j \in E} \delta_j(x_j; e_j) \right) \delta_k(x_k; x'_k) \\ &= \sum_{x \in U_N} g(x), \end{aligned} \tag{2}$$

where $g(x) = (\prod_{i \in N} p_i(x_i|x_{F(i)}))(\prod_{j \in E} \delta_j(x_j; e_j))\delta_k(x_k; x'_k)$ for all $x \in U_N$ and $\delta_k(x_k; x'_k)$ is equal to 1 if $x_k = x'_k$ and 0 otherwise. One way of estimating the addition in (2) is the *importance sampling* technique.

3. Importance sampling in Bayesian networks

Importance sampling is well known as a variance reduction technique for estimating integrals by means of Monte Carlo methods (see, for instance, Rubinstein, 1981). Here we study how to use it to estimate additions instead of integrals.

The technique is based on a transformation of formula (2). We consider a probability function $p^* : U_N \to [0,1]$, verifying that $p^*(x) > 0$ for every point $x \in U_N$ such that $g(x) > 0$. Then, we can write formula (2) as follows:

$$p(x'_k, e) = \sum_{\substack{x \in U_N, \\ g(x)>0}} \frac{g(x)}{p^*(x)} p^*(x) = E\left[\frac{g(X^*)}{p^*(X^*)}\right],$$

where X^* is a random variable with distribution p^*. Then,

$$\xi = \frac{g(X^*)}{p^*(X^*)} \tag{3}$$

is an unbiased estimator of $p(x'_k, e)$ with variance $var(\xi)=E[\xi^2]-[E\xi]^2=(\sum_{x \in U_N} g^2(x) / p^*(x)) - p^2(x'_k, e)$.

Since the sample mean is an unbiased estimator of the expectation, we can estimate $p(x'_k, e)$ from a sample $\{x^{(j)}\}_{j=1}^m$ for variable X^* as

$$\frac{1}{m}\sum_{j=1}^{m} \frac{g(x^{(j)})}{p^*(x^{(j)})}. \tag{4}$$

Minimizing the variance of this sample mean is the same as minimizing the variance of ξ. It can be shown that this minimum is equal to zero, and it can only be achieved if we sample with a distribution $p^*(x)=g(x)/(\sum_{y \in U_N} g(y))=g(x)/p(x'_k, e)$ for all $x \in U_N$.

However, in general, we will not be able to use a sampling distribution proportional to $g(x)$, but at most a distribution close to it, in some sense.

The drawback of this method, as formulated above, is that it is necessary to apply it separately to every value of variable X_k. And if we want to estimate the 'a posteriori' probability for a different variable X_l, then we have to repeat the calculations obtaining a sample for each one of the possible values of variable X_l. If we want to calculate the 'a posteriori' probability for all the variables in the network, then for each case of each variable, a sampling distribution has to be computed and a sample drawn from it. This is the solution adopted in Cano et al. (1996). Dagum and Luby (1997) use a slightly different method; they estimate $p(x'_k, e)$ and $p(e)$ with different samples in order to obtain and approximation for $p(x'_k|e)$. However, this process may be too costly if the size of the network is large enough.

Hernández et al. (1998) propose a variation over the former scheme that allows to estimate the probabilities for all of the variables more quickly. The idea is to perform a simulation as if we wanted to estimate $p(e)$. In this case,

$$p(e) = \sum_{\substack{x \in U_N \\ x_E = e}} \prod_{i \in N} p_i(x_i|x_{F(i)})$$

$$= \sum_{x \in U_N} \left(\prod_{i \in N} p_i(x_i|x_{F(i)}) \right) \left(\prod_{j \in E} \delta_j(x_j; e_j) \right) = \sum_{x \in U_N} f(x), \tag{5}$$

with $f(x) = (\prod_{i \in N} p_i(x_i|x_{F(i)}))(\prod_{j \in E} \delta_j(x_j; e_j))$ for all $x \in U_N$.

We can construct an estimator ξ for $p(e)$ in the same way as we did before:

$$\xi = \frac{f(X^*)}{p^*(X^*)}, \tag{6}$$

where f is as in Eq. (5).

Using distribution p^*, we generate a sample $x^{(i)} \in U_N$, $i = 1, \ldots, m$. This single sample is used to estimate all the probability values in the following way: for each value x'_k of variable X_k, we take all the configurations $x^{(j)}$, $j \in J^{(x'_k)} \subseteq \{1, \ldots, m\}$ such that $x_k^{(j)} = x'_k$ and estimate $p(x'_k, e)$ as

$$\hat{p}(x'_k, e) = \frac{1}{m} \sum_{j \in J^{(x'_k)}} \frac{f(x^{(j)})}{p^*(x^{(j)})} = \frac{1}{m} \sum_{j=1}^{m} w_j. \tag{7}$$

Note that, in fact, what we are doing is to use an unbiased estimator, $\xi_{x'_k}$, of $p(x'_k, e)$, and each w_j is a value for that estimator, which is defined as

$$\xi_{x'_k} = \frac{f(X^*)\delta_k(X^*_k; x'_k)}{p^*(X^*)}, \tag{8}$$

with X^* a random variable with distribution p^*, but in this case the sampling distribution, p^*, is constructed for estimating $p(e)$ instead of $p(x'_k, e)$. Therefore, we must not expect the minimum variance to be equal to zero, even in the case of choosing p^* proportional to f. In fact, what we have when p^* is proportional to f, is that $var(\xi_{x'_k}) = p(x'_k, e)(p(e) - p(x'_k, e))$, and therefore, taking into account Eq. (7) we have $var(\hat{p}(x'_k, e)) = (1/m)p(x'_k, e)(p(e) - p(x'_k, e))$.

However, sampling with a distribution proportional to f will not always be possible (this is equivalent to know the conditional probabilities $p(x_k|e)$ which are the values we are estimating), and the best we will be able to do is to select p^* close to $p(\cdot|e)$.

Once p^* is selected, we can estimate $p(x'_k, e)$, for each value x'_k of each variable X_k, $k \in N - E$, with the following algorithm:

Importance Sampling

```
1. For $j := 1$ to $m$ (sample size)
   (a) Generate a configuration $x^{(j)} \in U_N$ using $p^*$.
   (b) Calculate
       $$w_j := \frac{(\prod_{i \in N} p_i(x_i^{(j)} | x_{F(i)}^{(j)})) \cdot (\prod_{l \in E} \delta_l(x_l^{(j)}; e_l))}{p^*(x^{(j)})}. \qquad (9)$$
2. For each $x'_k \in U_k$, $k \in N - E$, estimate $p(x'_k, e)$ using formula (7).
3. Normalize values $p(x'_k, e)$ in order to obtain $p(x'_k | e)$.
```

It can be shown that using the same sample obtained to estimate $p(e)$, for estimating every $p(x'_k, e)$, does not imply an important increment on the variances of the estimators, even in the case in which p^* is not proportional to f. This is demonstrated in the following results.

Proposition 1. *Let $\hat{p}(e)$ and $\hat{p}(x'_k, e)$, $x'_k \in U_k$ be the sample mean estimators of $p(e)$ and $p(x'_k, e)$, $x'_k \in U_k$ respectively. Then, using the same sample of total size m for computing both estimators, it holds that*

$$\sum_{x'_k \in U_k} var(\hat{p}(x'_k, e)) = var(\hat{p}(e)) + \frac{p^2(e) - \sum_{x'_k \in U_k} p^2(x'_k, e)}{m}. \qquad (10)$$

Proof. Let ξ and $\xi_{x'_k}$ be unbiased estimators of $p(e)$ and $p(x'_k, e)$ as defined in Eqs. (6) and (8), respectively. Let $U_k = \{x'_{k_1}, \dots, x'_{k_n}\}$ be the set of all possible values of variable X_k. It is clear that $\xi = \sum_{x'_k \in U_k} \xi_{x'_k}$ and $\xi_{x'_{k_i}} \cdot \xi_{x'_{k_j}} = 0$ if $x'_{k_i} \neq x'_{k_j}$. In these conditions,

$$E[\xi^2] = E[(\xi_{x'_{k_1}} + \dots + \xi_{x'_{k_n}})^2]$$

$$= \sum_{x'_k \in U_k} E[\xi^2_{x'_k}] + 2 \sum_{\substack{x'_{k_i}, x'_{k_j} \in U_k \\ x'_{k_i} \neq x'_{k_j}}} E[\xi_{x'_{k_i}} \cdot \xi_{x'_{k_j}}] = \sum_{x'_k \in U_k} E[\xi^2_{x'_k}].$$

Thus,

$$var(\xi) = E[\xi^2] - [E\xi]^2 = \left(\sum_{x'_k \in U_k} E[\xi^2_{x'_k}] \right) - p^2(e). \qquad (11)$$

Since $E[\xi_{x'_k}] = p(x'_k, e)$, then $var(\xi_{x'_k}) = E[\xi^2_{x'_k}] - p^2(x'_k, e)$. Substituting in formula (11),

$$var(\xi) = \sum_{x'_k \in U_k} (var(\xi_{x'_k}) + p^2(x'_k, e)) - p^2(e). \qquad (12)$$

Now, if we consider the sample mean estimators $\hat{p}(e)$ and $\hat{p}(x'_k, e)$, we have that $var(\hat{p}(e)) = var(\xi)/m$ and $var(\hat{p}(x'_k, e)) = var(\xi_{x'_k})/m$. This, together with formula (12) implies that

$$\sum_{x'_k \in U_k} var(\hat{p}(x'_k, e)) = var(\hat{p}(e)) + \frac{p^2(e) - \sum_{x'_k \in U_k} p^2(x'_k, e)}{m}. \qquad \square$$

So, according to this proposition, for every $\varepsilon > 0$, if we select m such that $1/m < \varepsilon$, then we can assure that $var(\hat{p}(x'_k, e)) < var(\hat{p}(e)) + \varepsilon$. As a consequence, if we choose a sample producing a small variance in the estimation of $p(e)$, then the same sample will produce small variances for the estimations of every $p(x'_k, e)$, $x'_k \in U_k$.

In essence, we do not want to estimate $p(x'_k, e)$ but the normalized value $p(x'_k|e) = p(x'_k, e)/p(e)$. This is done by using $\hat{p}(x'_k|e) = \hat{p}(x'_k, e)/\hat{p}(e)$. We do not know exact results about the distribution or variance of this estimation. Geweke (1989) provides asymptotic results showing that in the limit it has a normal distribution with a variance that in our case can be expressed as

$$\sigma^2 = \frac{E[(\delta_k(X_k; x'_k) - p(x'_k|e))^2 \cdot \xi]}{m},$$

where X_k follows distribution $p(\cdot|e)$ and $\delta_k(X_k; x'_k)$ is equal to 1 if $X_k = x'_k$ and 0 otherwise.

When the weights are constant ($p^* = p(.|e)$) then this variance is exact and equal to $p(x'_k|e)(1 - p(x'_k|e))/m$. This is a desirable variance, but problems occur when we have a large variance in the weights: most of them are small and a few of them very high. Geweke (1989) defines the relative efficiency of an estimator (RNE) as the ratio of $p(x'_k|e)(1 - p(x'_k|e))/m$ (the variance obtained when $p^* = p(.|e)$) and the variance of our estimator. Low values of the RNE indicate a poor behaviour of our sampling distribution.

3.1. Computing a sampling distribution

The performance of the simulation procedure described above depends on the sampling distribution. Known importance sampling algorithms, as those developed by Cano et al. (1996); Fung and Chang (1990); Shachter and Peot (1990) and Dagum and Luby (1997) generate configurations simulating values for each non-observed variable using its conditional distribution, and instantiating each observed variable in X_E to the evidence e. This may lead to bad situations. This happens when most of the weights are low and a few of them are high. The following example illustrates this situation.

Example 1. Assume a very simple Bayesian network with 2 variables X_1 and X_2. X_1 is the father of variable X_2. Each variable X_i can take two values $\{x_{i_0}, x_{i_1}\}$. We assume the following initial probabilities:

$$p_1(x_{1_0}) = 1 - \varepsilon_1, \quad p_1(x_{1_1}) = \varepsilon_1,$$
$$p_2(x_{2_0}|x_{1_0}) = 1 - \varepsilon_2, \quad p_2(x_{2_1}|x_{1_0}) = \varepsilon_2,$$
$$p_2(x_{2_0}|x_{1_1}) = 0, \quad p_2(x_{2_1}|x_{1_1}) = 1,$$

where ε_0 and ε_1 are positive real numbers very close to zero.

The evidence set is $e = \{X_2 = x_{2_1}\}$. Then, the joint probabilities of configurations and evidence are

$$\begin{aligned} &p(x_{1_0}, x_{2_0}, e) = 0, \quad p(x_{1_0}, x_{2_1}, e) = (1 - \varepsilon_1)\varepsilon_2, \\ &p(x_{1_1}, x_{2_0}, e) = 0, \quad p(x_{1_1}, x_{2_1}, e) = \varepsilon_1, \end{aligned}$$

and the probability of the evidence is $(1 - \varepsilon_1)\varepsilon_2 + \varepsilon_1 = \varepsilon_1 + \varepsilon_2 - \varepsilon_1\varepsilon_2$.

Classical likelihood weighting simulation, works in the following way:

- It obtains a value for X_1 according to its 'a priori' distribution p_1. That is, we get x_{1_0} with probability $1 - \varepsilon_1$ and x_{1_1} with probability ε_1.
- The value of X_2 is fixed to the observation $X_2 = x_{2_1}$.
- Configuration (x_{1_i}, x_{2_1}) is weighted according to importance sampling (formula (9)), with the following values:

$$\begin{aligned} w(x_{1_0}, x_{2_1}) &= \frac{(1 - \varepsilon_1)\varepsilon_2}{(1 - \varepsilon_1)} = \varepsilon_2, \\ w(x_{1_1}, x_{2_1}) &= \frac{\varepsilon_1}{\varepsilon_1} = 1. \end{aligned}$$

Observe that the weights are quite different: one of them is 1 and the other one is very close to 0. Furthermore, in most of the cases (with probability $1 - \varepsilon_1$) we will obtain a configuration with low weight and in some very few cases we will obtain a configuration of weight 1. This is an important problem in the estimation of conditional probabilities. From an intuitive point of view, the situation is that most of the configurations have very little importance in the estimation, and only few of them have a real importance.

From a more formal point of view, when computing the conditional probabilities for variable X_1, the variance of the estimation of $p(x_{1_1}, e)$ for a sample of size m is equal to $var_1 = \varepsilon_1(1 - \varepsilon_1)/m$. This variance is much higher than when we generate the sample with probability proportional to $p(.|e)$. In that case, the variance is: $var_2 = \varepsilon_1\varepsilon_2(1 - \varepsilon_1)/m$. The relative numerical efficiency (RNE) of this sampling distribution (Geweke, 1989) is $var_2/var_1 = \varepsilon_2$, which is very close to 0, indicating a very poor behaviour.[1]

The situation is that when we start to simulate, the values obtained for variable X_1 according to its 'a priori' distribution are quite incompatible with the observation on X_2. To solve this problem, Fung and Favero (1994) proposed the so called *backward simulation*, in which we start to simulate in the observations and then going backward to the root nodes. In this case, the procedure is:

- Fix the value of X_2 to the observed value $X_2 = x_{2_1}$.
- From X_2 obtain a value for its parent X_1 by using the conditional probability of X_2 given X_1 and the observed value. This implies that the sampling probabilities for X_1 are $p^*(x_{1_0}) = \varepsilon_2/(1 + \varepsilon_2)$ and $p^*(x_{1_1}) = 1/(1 + \varepsilon_2)$.
- The configuration is weighted according to importance sampling: $w(x_{1_0}, x_{2_1}) = (1 - \varepsilon_1)(1 + \varepsilon_2)$ and $w(x_{1_1}, x_{2_1}) = \varepsilon_1(1 + \varepsilon_2)$.

[1] Note that we have used joint probabilities instead of conditional ones to compute the RNE. It can be checked that the result is the same.

Again, we find the same situation: in most of the configurations (with probability $1/(1+\varepsilon_2)$) we obtain the value x_{1_1} for variable X_1, but in that case the weight is really small if ε_1 is very close to 0.

The problem now is that we start to simulate in the observations and they have a high degree of inconsistency with the 'a priori' information about X_1.

In the example above, there are two parts of the graph which are contradictory, which motivates that simulation will not work fine when it starts in one part of the graph without taking into account the information contained in other parts of the graph. The result would be the obtainment of very small weights with a very high probability. The solution to this problem could be to calculate the probability $p(.|e)$ and then to simulate with this distribution. This is indeed feasible in the case of the above very simple network involving only two variables, but in most of the cases it will be infeasible. Then, what we can do is to try to get an estimation of it in a reasonable time, so that when we simulate a value for a variable we use most of the information in the graph that we can afford.

In this direction, the *approximate pre-computation* technique, introduced by Hernández et al. (1996, 1998), consists of computing a sampling distribution gathering as much information as possible from all the information available. An exact sampling distribution (obtaining configuration x with a probability equal to $p(x|e)$) can be calculated locally by means of a probabilistic propagation algorithm, more precisely, we consider a variable elimination algorithm as in Zhang and Poole (1996). In the following, we briefly describe such procedure. More details can be found in Hernández et al. (1998).

Assume that H is the set of all the potentials involved in the calculation of f, i.e. $H=\{p_i \mid i=1,\ldots,n\}\cup\{\delta_j(\cdot;e_j)\mid j\in E\}$. An elimination order σ is considered and variables are deleted according to such order: $X_{\sigma(1)},\ldots,X_{\sigma(n)}$.

The deletion of a variable $X_{\sigma(i)}$ consists of combining all the functions in H which are defined for that variable, marginalizing afterwards in order to remove $X_{\sigma(i)}$ from the set of variables for which the combination is defined. The potential obtained is inserted in H. More precisely, the steps are as follows:

- Let $H_{\sigma(i)}=\{h_j\in H \mid \sigma(i)\in s(h_j)\}$.
- Calculate $h=\prod_{h_j\in H_{\sigma(i)}} h_j$ and $h'=h^{\downarrow s(h)-\sigma(i)}$.
- Transform H into $H-H_{\sigma(i)}\cup\{h'\}$.

Hernández et al. (1998) have shown that if h is the potential calculated when deleting variable $X_{\sigma(i)}$ and $J=\{\sigma(i+1),\ldots,\sigma(n)\}$ we have that, for every $x\in U_N$,

$$p(x_{\sigma(i)}|x_J,e)\propto h(x_{\sigma(i)},x_{J\cap s(h)}), \tag{13}$$

where symbol $\propto$ means "proportional to".

This provides a method to obtain a configuration x with probability proportional to $p(x,e)$: we simulate values in the opposite deletion order: $x_{\sigma(n)},\ x_{\sigma(n-1)},\ldots,x_{\sigma(2)},\ x_{\sigma(1)}$. To get a value $x_{\sigma(i)}$ we use a sampling distribution $p^*_{\sigma(i)}(\cdot|x_J)=p(x_{\sigma(i)}|x_J,e)$, which is calculated using expression (13).

This is the procedure to obtain an exact sampling distribution, i.e. $p^*(x|e)=p(x|e)$. In some cases, the computation of exact sampling functions h will not be possible. The difficulty is in the combination by multiplication of all the potentials in $H_{\sigma(i)}$.

The resulting function h is defined for variables $X_{s(h)}$, where $s(h) = \bigcup_{h_j \in H_{\sigma(i)}} s(h_j)$, and the number of values necessary to specify h is exponential in the number of variables for which it is defined.

Hernández et al. (1998) propose to follow the same scheme as in exact propagation, but when the computation of h' is not feasible, they try to obtain an approximation. The approximation consists in not multiplying all the functions before marginalizing, but considering some partition of H and then applying the combination-marginalization steps to all the elements of the partition. Several criteria have been considered to determine the partition, but in general they are based in not surpassing and upper limit for the size of a potential. A similar approximate approach (the minibucket elimination) has been used in Dechter and Rish (1997) to calculate the configuration of maximum probability.

Previous algorithms are based on the representation of probabilistic potentials by means of probability tables. In this paper we propose probability trees (Boutilier et al., 1996) as a more convenient representation of potentials to obtain approximations. Probability trees take advantage of context-specific independence to produce more efficient representations of potentials. Cano and Moral (1997) have shown the capabilities of probability trees to produce good approximations under limited resources. In this paper, we propose a new approximate method for deleting variables. This method is based on the use of probability trees instead of tables, and leads to the definition of an efficient algorithm for approximate probabilistic propagation. Probability trees have more power to represent potentials in a compact form. So, in this way we can obtain better approximations using the same resources (space and time) and, as a final consequence, a higher relative efficiency value.

One important aspect of the elimination procedure described here is the order in which variables are removed. Depending on this order, potentials of different sizes may be obtained. This fact can be seen in the following example:

Example 2. Assume we have two potentials $h_1(X_1,X_4,X_5)$ and $h_2(X_1,X_2,X_3)$, and we want to remove variables X_1 and X_2. If we remove first X_1, we have to combine h_1 and h_2, since both contain variable X_1, obtaining a potential defined for the five variables $X_1,\dots,X_5$. Then, we marginalize and obtain a new potential h' defined for X_2,X_3,X_4 and X_5. Now we proceed to remove X_2. To this end, we just have to marginalize h' obtaining a new potential h'' defined for X_3,X_4 and X_5. Observe that the biggest potential we have obtained is defined for five variables.

If instead we remove first X_2, we just have to marginalize h_2 obtaining a new potential h' defined for variables X_1 and X_3. Now, in order to delete X_1, we combine h_1 and h', obtaining a new potential h'' defined for X_1,X_3,X_4 and X_5. Note that, in this case, the biggest potential we have obtained is defined just for four variables.

It is convenient to find an elimination order producing small potentials, since in this way we will obtain approximate potentials closer to the exact ones in the approximation procedure described above, thus reducing the variability in the estimations.

In Hernández et al. (1998), the variables are deleted from leaves to roots, which implies that the simulation order is from roots to leaves, as it is in likelihood weighting (Fung and Chang, 1990) and bounded variance (Dagum and Luby, 1997).

The problem of obtaining an optimal deletion sequence is the same as obtaining an optimal triangulation of the moral graph associated with a Bayesian network. Good heuristic algorithms for triangulating graphs have been developed by Cano and Moral (1995) and Kjærulff (1992).

The complexity of the resulting variable elimination algorithm will depend on the size of intermediate potentials. Bad deletion sequences will give rise to big size potentials and therefore to the need of important approximation steps with the consequence of poor approximations of the sampling distribution $p(\cdot|e)$. A good deletion sequence may give rise to small potentials allowing an exact calculation of the sampling distribution $p(\cdot|e)$. Obtaining an optimal sequence is an NP-hard problem, but there are good heuristics allowing to obtain good sequences in reasonable time.

In this work we have considered the following heuristic: first of all remove variables that are contained in the domain of only one potential. If there are not such variables, then select one according to the minimum size heuristic (see Kjærulff, 1992), that is, one variable such that the size of the combination of all the potentials for which the variable is defined is the smallest.

4. Probability trees

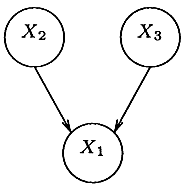
A *probability tree* is a directed labeled tree, each of its inner nodes representing a variable and each of its leaf nodes representing a probability value. Each inner node will have as many outgoing arcs as possible values the variable it represents has. Each leaf of the tree contains a real number. We define the *size* of a tree as the number of leaves it has.

A probability tree $\mathcal{T}_f$ on variables X_I represents potential $f : U_I \rightarrow \mathbb{R}$ if for each $x_I \in U_I$ the value $f(x_I)$ is the number stored in the leaf node that is obtained starting in the root node and selecting for each inner node labeled with X_i the child corresponding to coordinate x_i. We will denote by $\mathcal{T}_f(x_I)$ the value $f(x_I)$.

Each leaf of a tree T has associated a configuration $X_J = x_J$ where X_J are the variables appearing in the path from the root node to the leaf, and x_J are the values of these variables corresponding to this path.

Probability trees are appropriate tools for representing regularities in probability potentials. Such regularities are characterized by the concept of *context-specific independence* (see Boutilier et al., 1996).

Example 3. Fig. 1 displays a network with variables X_1,X_2 and X_3, each of them taking two possible values, $X_i = 1$ or $X_i = 2$, $i = 1, 2, 3$. The table represents the conditional distribution $P(X_1|X_2,X_3)$. It can be seen that X_1 and X_3 are independent given context $X_2 = 2$. This independence is reflected in the tree in the figure: it contains the same information as the table, but using five values instead of eight. The tree in Fig. 1 represents potential $f(x_1,x_2,x_3) = P(X_1 = x_1|X_2 = x_2,X_3 = x_3)$.



x_1	x_2	x_3	$p(x_1 \mid x_2, x_3)$
1	1	1	0.2
1	1	2	0.5
1	2	1	0.7
1	2	2	0.7
2	1	1	0.8
2	1	2	0.5
2	2	1	0.3
2	2	2	0.3

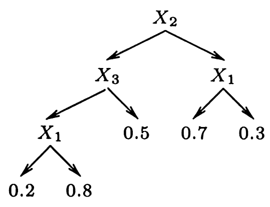


Fig. 1. Probability table and its corresponding tree.

For instance, value 0.2 corresponds to $f(1,1,1) = P(X_1 = 1 | X_2 = 1, X_3 = 1)$, and 0.7 corresponds to $f(1,2,1)$ and $f(1,2,2)$.

Note that probability trees can be used to represent any potential, not only conditional distributions. Hence, they can be used to represent potentials resulting in intermediate steps in simulation algorithms, that may be unnormalized.

We have to fix the following aspects before using probability trees in our simulation algorithm. First, the initial conditional distributions in the network may be given as tables. Thus, we need a way of translating a probability table into a tree. Second, we have to define the product of potentials and the marginal of a potential within the context of probability trees, because otherwise, each time an operation is going to be carried out, we would have to move from one representation to another, which is computationally inefficient. Third, it may happen that the tree resulting from an operation has a size bigger than the maximum allowed. Hence, we need a way of pruning a tree until its size is below the threshold. We will approach these tasks following the methodology developed by Cano and Moral (1997).

4.1. Constructing a probability tree

Let f be a potential over a set of variables X_I. Constructing a tree $\mathcal{T}$ representing potential f without any other additional restriction is a trivial task: the tree will contain one branch for every configuration of X_I, and one leaf for every value $f(x_I)$ with $x_I \in U_I$. However, this procedure can lead to unnecessarily big trees. For example, consider the trees in Fig. 2. Assume tree (a) is the result of the procedure above. Changing the positions of variables X_1 and X_2 we obtain tree (b). Now, we can realize that the value of X_1 is irrelevant given the value of X_2. Thus, we can construct tree (c) which represents the same potential as (a) and (b) but with lower size.

Two problems can be considered: how to represent a potential with a minimal tree? or how to approximate a potential with a tree of smaller size? For the second question we need a distance to measure the goodness of the approximation. These questions are addressed by Cano and Moral (1997). The criterion they use is that the resulting tree $\mathcal{T}$ must minimize the distance to potential f. If we denote by $p_{\mathcal{T}}$ and p_f the probability distributions proportional to $\mathcal{T}$ and f respectively, the *distance* from a

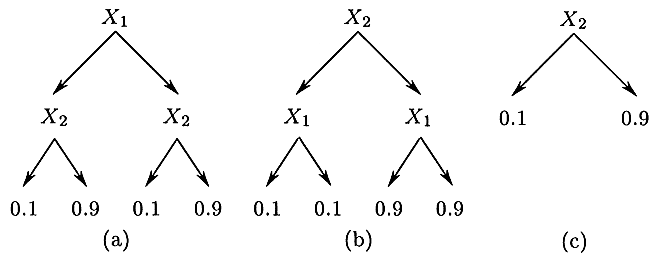


Fig. 2. Three trees containing the same information.

tree $\mathcal{T}$ to a potential f is measured by Kullback–Leibler's divergence (Kullback and Leibler, 1951):

$$D(f,\mathcal{T}) = -\sum_{x_I \in U_I} p_f(x_I) \log \frac{p_f(x_I)}{p_{\mathcal{T}}(x_I)}. \tag{14}$$

For the approximation, there are two subproblems, what is the structure (shape of the tree and variables on inner nodes) of the representation? and which are the numbers on the tree leaves? The most difficult problem is the first one. Given the structure, the numbers minimizing the distance can be easily calculated using the following result (Cano and Moral, 1997): given a tree structure the optimal approximation of f can be obtained by putting in every leaf corresponding to configuration $X_J = x_J$ the average of the values $f(y_I)$ with $y \in U_N$ and $y_J = x_J$.

The methodology proposed in Cano and Moral (1997) to build a minimal probability tree is based on methods for inducing classification rules from examples. One of these methods is Quinlan's ID3 algorithm (Quinlan, 1986), that builds a *decision tree* from a set of examples. A decision tree represents a sequential procedure for deciding the class membership of a given instance of the attributes of the problem. That is, the leaves of the decision tree give us the class for a given instance of the attributes. ID3 builds a decision tree in a greedy way, by choosing a *good test attribute* to refine the actual structure. To determine which attribute should be the test attribute for a leaf node of the tree, the algorithm applies an information-theoretic measure $Gain(A_i)$ over all the possible attributes A_i. This information measure gives an idea of the gain of information by partitioning a leaf node in the tree with an attribute.

In the case of probability trees, Cano and Moral (1997) use the same algorithm, but the information measure is different, because each leaf in a decision tree represents a class, while in a probability tree, each leaf represents a probability value. Then, we need a measure particularly adapted to probabilities.

The process of constructing a tree can be seen as follows. Assume $\mathcal{T}$ is the tree we are constructing. A construction step requires to decide which branch to expand and which variable to place in the new node. This selection must be done in such a way that the distance to the potential is minimized. If a leaf node in $\mathcal{T}$ corresponds to a configuration $X_J = x_J$, we denote by $\mathcal{T}(X_J = x_J, X_k)$ the tree obtained from $\mathcal{T}$ by expanding the leaf defined by $X_J = x$ with variable X_k. At each moment, both the branch and the variable are selected in order to minimize the distance to f, that is,

$$D(\mathscr{T}(X_J = x_J, X_k), f) = \min\{D(\mathscr{T}(X_{J'} = x_{J'}, X_{k'}), f)\}, \tag{15}$$

where $X_{J'} = x_{J'}$ is a leaf of $\mathscr{T}$ and $k' \in I - J'$. The following proposition shows a way of computing that minimum.

Proposition 2 (Cano and Moral, 1997). *The pair* $(X_J = x_J, X_k)$ *minimizing expression* (15) *is that one maximizing the measure of information*

$$I(X_J = x_J, X_k | f) = S_{X_J = x_J} \cdot (\log|U_k| - \log S_{X_J = x_J} - E[X_k | X_J = x_J]), \tag{16}$$

where

$$S_{X_J = x_J} = \sum_{\substack{z \in U_N \\ z_J = x_J}} f(z_I),$$

$$E[X_k | X_J = x_J] = -\sum_{y_k \in U_k} f^{\downarrow k}(y_k | X_J = x_J) \log f^{\downarrow k}(y_k | X_J = x_J),$$

$$f^{\downarrow k}(y_k | X_J = x_J) = \sum_{\substack{z \in U_N \\ z_J = x_J \\ z_k = y_k}} f(z_I).$$

The value of information $I(X_J = x_J, X_k | f)$ measures the distance from a tree $\mathscr{T}$ to a potential f before and after expanding the branch $X_J = x_J$ with variable X_k. The proposition above means that we must select branches leading to configurations with high probability, and variables with small entropy.

With this, a procedure to construct an *exact* tree is to select nodes maximizing function I. The procedure would finish when, for every branch $X_J = x_J$, the values of f are uniform, that is, $f(y_I) = f(y'_I)$ for all $y_I, y'_I \in U_I$ such that $y_J = y'_J = x$. That is, the idea is to include nodes until no new information is provided by adding new nodes.

For constructing an *approximate* tree, Cano and Moral (1997) propose different alternatives. One of the alternatives consists of adding nodes until an exact representation is obtained or a maximum number of nodes is reached.

The other alternative is to construct the entire tree and prune it afterwards. If $\mathscr{T}$ is such tree, a *pruning* consists of selecting a node such that all its children are leaves and replacing it and its children by one node containing the average of the values of the leaf nodes being removed. We have two ways of performing a pruning.

The first way of pruning is to remove nodes while the maximum size is exceeded. The selection of a node will be determined by that pair $(X_J = x_J, X_k)$ minimizing the measure $I(X_J = x_J, X_k | f)$, that is, the pair minimizing the increment of the distance to potential f.

The following algorithms performs this task:

APPROXIMATE($\mathcal{T}, M$)

```
1. While the size of $\mathcal{T}$ is greater than $M$,
   (a) Let $\mathcal{N}$ be the set of nodes in $\mathcal{T}$ such that all their children are leaves.
   (b) For each node $X_k \in \mathcal{N}$,
       • Let $X_J = x_J$ be the configuration leading to node $X_k$.
       • Compute $I(X_J = x_J, X_k | f)$.
   (c) Replace in $\mathcal{T}$ the node from $\mathcal{N}$ minimizing the information measure $I(\cdot | f)$
       by the average of the values of its children.
```

The other way of pruning is to remove nodes determined by pairs $(X_J = x_J, X_k)$ such that

$$I(X_J = x_J, X_k | f) \leq \Delta, \tag{17}$$

where Δ is a threshold for the increment of the distance. The pruning would finish when there are no more pairs verifying condition (17). The way of selecting Δ is as follows: given a parameter $0 < \varepsilon < 0.5$, we compute Δ as the entropy of the distribution $(0.5 - \varepsilon, 0.5 + \varepsilon)$, i.e., the distribution obtained by moving in a fraction ε from the uniform distribution for a binary variable. That is,

$$\Delta = -((0.5 - \varepsilon)\log(0.5 - \varepsilon) + (0.5 + \varepsilon)\log(0.5 + \varepsilon)). \tag{18}$$

The goal of this procedure is to detect leaves in the tree with very similar values (i.e. close to an uniform distribution), since they can be replaced by the average value without an important increment of the distance to the exact tree.

The details of this procedure are described in the following algorithm:

PRUNE($\mathcal{T}, \epsilon$)

```
1. Compute $\Delta$ as in formula (18).
2. Let $\mathcal{N}$ be the set of nodes in $\mathcal{T}$ such that all their children are leaves.
3. While $\mathcal{N} \neq \emptyset$,
   (a) Remove a node $X_k$ from $\mathcal{N}$.
   (b) Let $X_J = x_J$ be the configuration leading to node $X_k$.
   (c) If $I(X_J = x_J, X_k | f) \leq \Delta$,
       • Replace in $\mathcal{T}$ node $X_k$ by the average of the values of its children.
       • Let $X_k'$ be the parent of $X_k$ in $\mathcal{T}$.
       • If all the children of $X_k'$ are leaves, insert $X_k'$ in $\mathcal{N}$.
```

Note that we will always be able to obtain the exact trees representing the initial conditional probability tables given in the network, since the trees are, at most, of the same size as the corresponding tables.

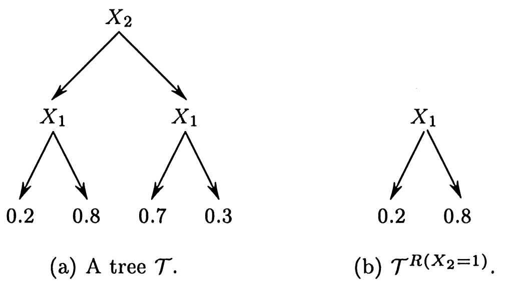


Fig. 3. The restriction operation.

4.2. Operations over probability trees

Propagation algorithms require three operations over potentials: *restriction, combination and marginalization*. In this section we briefly describe the algorithms proposed in Kozlov and Koller (1997). Cano and Moral (1997) give more general algorithms in which the structure of the operands is mixed trying to obtain an optimal representation. Here the algorithms are simpler and will take one of the operands as basis expanding it with the structure of the other tree.

The *restriction* operation is trivial. Given a tree $\mathcal{T}$, a set of variables X_J, and $x_J \in U_J$, $\mathcal{T}^{R(X_J=x_J)}$ denotes the *restriction* of $\mathcal{T}$ to the values x_J of the variables in X_J, that is, the tree obtained by substituting in $\mathcal{T}$ every node corresponding to variables $X_k, k \in J$ by the subtrees $\mathcal{T}_k$ children of X_k corresponding to $X_k = x_k$. The restriction operation is illustrated in Fig. 3.

The other two operations, *combination* and *marginalization*, are more complicated. In the next, we give the detailed algorithms for performing those operations.

Given a tree $\mathcal{T}$ representing a potential f over a set of variables X_I, we denote by $\mathrm{Vr}(\mathcal{T})$ the set of variables corresponding to inner nodes in $\mathcal{T}$. It holds that $\mathrm{Vr}(\mathcal{T}) \subseteq X_I$, but after a pruning is performed, it can happen that $\mathrm{Vr}(\mathcal{T}) \neq X_I$. We will denote by $\mathrm{Vr}^*(\mathcal{T})$ the set of variables for which the potential represented by $\mathcal{T}$ is defined. In this case, even after any pruning, it holds that $\mathrm{Vr}^*(\mathcal{T}) = X_I$.

The label of a node of a tree will be equal to the variable corresponding to the node. If the node is a leaf, then the label will be equal to the probability value attached to that leaf.

Given two trees $\mathcal{T}_1$ and $\mathcal{T}_2$ representing potentials f_1 and f_2 respectively, the following algorithm computes a tree representing potentials $f = f_1 \cdot f_2$ (combination).

```
COMBINE($\mathcal{T}_1, \mathcal{T}_2$)

1. Create a tree node $\mathcal{T}_r$ initially with no label.
2. Let $L_1$ and $L_2$ be the labels of the root nodes of $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively.
3. If $L_1$ and $L_2$ are numbers, then make $L_1 \cdot L_2$ be the label of $\mathcal{T}_r$.
4. If $L_1$ is a number but $L_2$ is a variable, then
   (a) Make $L_2$ be the label of $\mathcal{T}_r$.
```

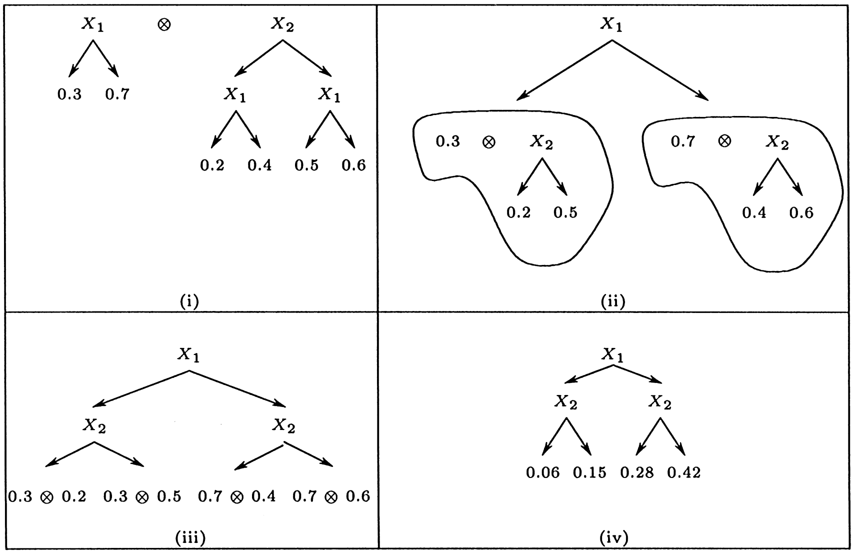


Fig. 4. Combination of two trees.

```
   (b) For every tree $\mathcal{T}$ child of the root node of $\mathcal{T}_2$, make $\mathcal{T}_h :=$ **COMBINE**($\mathcal{T}_1, \mathcal{T}$) be a child of $\mathcal{T}_r$.
5. If $L_1$ is a variable, assume that $X_k$ is that variable.
   (a) Make $X_k$ be the label of $\mathcal{T}_r$.
   (b) For each $x_k \in U_k$,
       • Make $\mathcal{T}_h$:=**COMBINE**($\mathcal{T}_1^{R(X_k=x_k)}, \mathcal{T}_2^{R(X_k=x_k)}$) be a child of $\mathcal{T}_r$.
6. Return $\mathcal{T}_r$.
```

We will denote the combination of trees by symbol ⊗. With this notation, the algorithm above returns a tree $\mathcal{T}_r = \mathcal{T}_1 \otimes \mathcal{T}_2$.

The combination process is illustrated in Fig. 4.

Given a tree $\mathcal{T}$ representing a potential f defined over a set of variables X_I, the following algorithm computes a tree representing potential $f^{\downarrow(I-\{i\})}$, with $i \in I$. That is, it removes variable X_i form $\mathcal{T}$.

MARGINALIZE($\mathcal{T}$,X_i)

```
1. Let $L$ be the label of the root node of $\mathcal{T}$.
2. If $L$ is a number, create a node $\mathcal{T}_r$ with label $L \cdot |U_i|$.
3. Otherwise, let $X_k$ be the variable corresponding to label $L$.
   (a) If $X_k = X_i$, then
       i. Let $\mathcal{T}_1, \ldots, \mathcal{T}_s$ be the children of the root node of $\mathcal{T}$.
       ii. $\mathcal{T}_r := \mathcal{T}_1$.
       iii. For $i := 2$ to $s$, $\mathcal{T}_r :=$ **ADD**($\mathcal{T}_r$, $\mathcal{T}_i$).
```

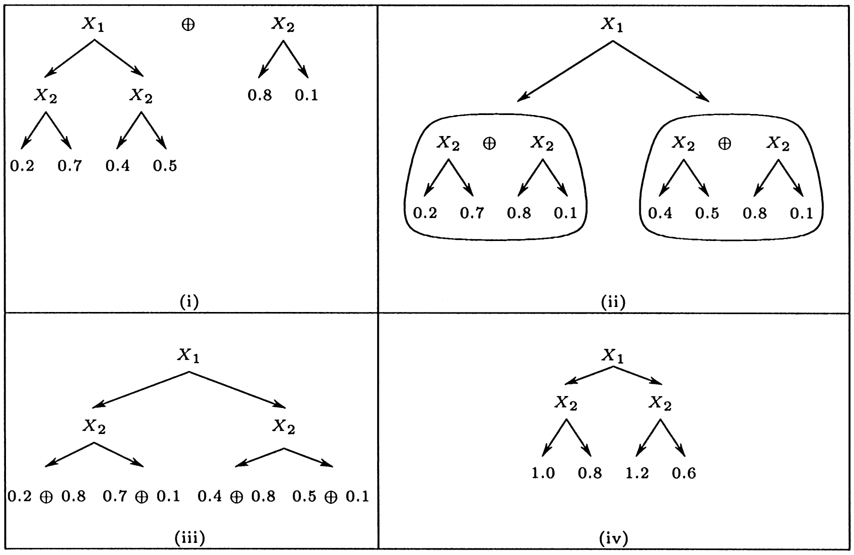


Fig. 5. Addition of two trees.

```
   (b) Otherwise,
       i. Create a node $\mathcal{T}_r$ with label $X_k$.
       ii. For each $x_k \in U_k$
           A. Make $\mathcal{T}_h :=$ **MARGINALIZE**($\mathcal{T}^{R(X_k=x_k)}, X_i$) be the next child of $\mathcal{T}_r$.
4. Return $\mathcal{T}_r$.
```

This algorithm uses procedure **ADD**($\mathcal{T}_1, \mathcal{T}_2$), which computes the addition of $\mathcal{T}_1$ and $\mathcal{T}_2$. The procedure is as follows:

ADD($\mathcal{T}_1$,$\mathcal{T}_2$)

```
1. Create a tree node $\mathcal{T}_r$ initially with no label.
2. Let $L_1$ and $L_2$ be the labels of the root nodes of $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively.
3. If $L_1$ and $L_2$ are numbers, then make $L_1 + L_2$ be the label of $\mathcal{T}_r$.
4. If $L_1$ is a number but $L_2$ is a variable, then
   (a) Make $L_2$ be the label of $\mathcal{T}_r$.
   (b) For every child $\mathcal{T}$ of the root node of $\mathcal{T}_2$, make $\mathcal{T}_h :=$ ADD($\mathcal{T}_1$, $\mathcal{T}$) be
       a child of $\mathcal{T}_r$.
5. If $L_1$ is a variable, assume that $X_k$ is that variable.
   (a) Make $X_k$ be the label of $\mathcal{T}_r$.
   (b) For each $x_k \in U_k$,
       • Make $\mathcal{T}_h :=$ ADD($\mathcal{T}_1^{R(X_k=x_k)}, \mathcal{T}_2^{R(X_k=x_k)}$) be a child of $\mathcal{T}_r$.
6. Return $\mathcal{T}_r$.
```

The addition of two trees is illustrated in Fig. 5, where symbol $\oplus$ represents the addition operation.

5. Importance sampling using probability trees

In this section we describe an importance sampling algorithm in which the potentials are represented by probability trees and the approximation is performed by pruning the tree.

Assume we are carrying out a deletion algorithm and potentials are represented by probability trees. We consider a set of trees T (initially, those corresponding to the conditional distributions in the network). Now, each time we delete a variable X_i, we proceed as in the exact algorithm, multiplying all the trees in T defined for X_i, and then marginalizing to remove this variable. After marginalization, two basic approximations are carried out, based on algorithms **APPROXIMATE** and **PRUNE** as described in Section 4.1.

The detailed algorithm for deleting a variable is as follows:

```
**DELETE**($T, X_i, \epsilon, S$)

1. Let $S(i) = \{\mathscr{T} \in T \mid X_i \in \mathrm{Vr}^*(\mathscr{T})\}$ be the set of trees defined for variable $X_i$.
   Remove $S(i)$ from $T$.
2. Until there is only one tree in $S(i)$,
   (a) Take two trees $\mathscr{T}_1, \mathscr{T}_2$ from $S(i)$.
   (b) Compute $\mathscr{T} :=$ COMBINE($\mathscr{T}_1, \mathscr{T}_2$).
   (c) Replace in $S(i)$, $\mathscr{T}_1$ and $\mathscr{T}_2$ by $\mathscr{T}$.
3. Let $\mathscr{T}$ be the only tree in $S(i)$.
4. Compute $\mathscr{T}' :=$ MARGINALIZE($\mathscr{T}, X_i$).
5. PRUNE($\mathscr{T}', \varepsilon$).
6. APPROXIMATE($\mathscr{T}', M$).
7. REDUCEVARIABLES($\mathscr{T}'$).
8. Add $\mathscr{T}'$ to $T$.
```

All the elements in this algorithm have been previously specified except **REDUCEVARIABLES**($\mathscr{T}'$). This procedure takes the variables in $\mathrm{Vr}^*(\mathscr{T}')$ and, for each one of them, checks whether this variable really appears in any branch of the tree (i.e. it is contained in $\mathrm{Vr}(\mathscr{T}')$). It can happen that after the approximation (pruning) some variable X_i does not appear in any of the tree labels (i.e. $X_i \in \mathrm{Vr}^*(\mathscr{T}')$ but $X_i \notin \mathrm{Vr}(\mathscr{T}')$). In that case, we check whether there is another tree $\mathscr{T}$ which is defined for this variable. If the result is positive, we delete the variable from the set $\mathrm{Vr}^*(\mathscr{T}')$. This does not change at all the factorization of the global probability distribution as product of trees, because $\mathscr{T}'$ does not give any information about this variable, and simplifies future calculations: $\mathscr{T}'$ will not be combined when deleting variable X_i. When the variable does not appear in other tree we keep it to obtain a sampling distribution for this variable without modifying the algorithm (it will be the uniform distribution). And, as this variable appears only in one potential its deletion will be very simple and will not increase the complexity of the algorithm.

In the algorithm above, S is an array that, at each position i, contains the combination of the trees corresponding to those functions defined for variable X_i when it was deleted. That tree will be used in the simulation step as the sampling distribution for X_i.

A deletion procedure is the basis to determine a sampling distribution. If the deletion of all the variables has been exact, then it can be shown (Hernández et al., 1998) that if we simulate variables in an order opposite to the elimination sequence, taking as sampling distribution the potential represented by the tree in $S(i)$ restricted to the values of the variables already simulated (notice that this is a potential depending only on variable X_i) then we are obtaining configurations, x, with probability equal to $p(x|e)$. The weights are constant and equal to $p(e)$. When the deletion has been approximate then this procedure should produce a sampling distribution close to $p(x|e)$.

The deletion sequence we propose is especially appropriate for the case in which few observations are given. In fact, it proceeds by deleting first those variables that are not observed and such that none of their descendants is observed either. Each time one of these variables is deleted, the conditional probability of this variable given its parents is removed, adding only the trivial potential identically equal to 1, which does not have any effect in posterior computations and can be ignored. This is a consequence of the fact that if a leaf node, say X, is not observed, then this variable only belongs to the potential representing the conditional probability of this node given its parents. Adding on X, we will obtain the potential identically equal to 1. After applying **REDUCEVARIABLES** the parent variables can be removed from their definition set. Repeating this task, we can delete all the variables that are not observed and without observed descendants. The extreme case is when none of the variables is observed. Then, this process will produce an exact sampling distribution $p(\cdot|e)$. When there are observations, the deletion sequence will improve this procedure in a certain degree, depending on the number of observed variables and their position in the directed acyclic graph.

In the following we give the details of the sampling algorithm. We start off with a network G with variables $X_1,\dots,X_n$, and a set of observations E. The algorithm is organized into four phases: initialization, approximate pre-computation, simulation and estimation.

Algorithm IS_T

- **Initialization phase**
  1. Let $H = \{p_i \mid i = 1,\dots,n\}$ be the set of conditional distributions in G, and $T = \{\mathcal{T}_1,\dots,\mathcal{T}_n\}$ the set of trees representing functions in H.
  2. Incorporate observations:
  (a) Compute $\mathcal{T}_i := \mathcal{T}_i^{R(X_E=e_E)}$, $i = 1,\dots,n$.
  (b) For each observed variable, X_l, $l \in E$, do $T := T \cup \{\mathcal{T}_{\delta_l(\cdot;e_l)}\}$, where $\mathcal{T}_{\delta_l(\cdot;e_l)}$ is a tree representing potential $\delta_l(\cdot;e_l)$.

```
• Collect phase (approximate pre-computation)
    3. Select an order $\sigma$ of variables in $G$, as described in Section 3.1.
    4. For $i := 1$ to $n$, DELETE($T, X_{\sigma(i)}, \varepsilon, S$).
• Simulation phase
    5. For $j := 1$ to $m$ (sample size),
    (a) $w_j := 1.0$.
    (b) For $i := n$ to 1,
        (i) Simulate a value for $X_{\sigma(i)}$, $x_i^{(j)}$, using $p_i^*$ as sampling distribution, where
            $p_i^*$ is the normalized potential corresponding to the tree $\mathcal{T}_i$ defined
            over variable $X_{\sigma(i)}$ and obtained as $\mathcal{T}_i = \mathcal{T}^{R(X_{\Sigma(i)}=x_0)}$, where $\mathcal{T}$ is the
            tree stored in $S(i)$, with $\Sigma(i) = \{\sigma(i+1), \ldots, \sigma(n)\}$, and $x_0$ the current
            configuration of variables already simulated ($X_{\Sigma(i)}$).
        (ii) Compute $w_j := w_j / p_i^*(x_i^{(j)})$.
    (c) Compute
        $w_j := w_j \left( \prod_{i=1}^{n} p_i(x_i^{(j)} | x_{F(i)}^{(j)}) \right) \cdot \left( \prod_{l \in E} \delta_l(x_l^{(j)}; e_l) \right)$.
• Estimation phase
    6. For each $x'_k \in U_k$, $k \in N - E$,
        (a) Estimate $p(x'_k, e)$ using formula (7).
    7. Normalize values $p(x'_k, e)$ to obtain $p(x'_k | e)$.
```

6. Experimental tests

Some experiments have been carried out to test the performance of the proposed algorithm. The experiments consisted of several propagations over a large network, comparing the performance of four algorithms: likelihood weighting (Fung and Chang, 1990; Shachter and Peot, 1990), bounded variance algorithm (Dagum and Luby, 1997), importance sampling using probability tables (referenced as IS) as described in Hernández et al. (1998), and importance sampling using probability trees (referenced as IS_T).

The network used is a subset of a pedigree one (Jensen et al., 1995), composed by 441 variables. Each node has two parents (but the roots) and a maximum of 43 children, and has three cases. The biggest initial conditional probability table has 27 values, since each variable has three possible values. We have considered two types of inferences over this network: in one case we have not considered observations and in the other we have considered 166 observations.

The reason to use this well-known kind of network in the experiments, is that traditional simulation procedures fail to provide good estimations of the posterior probabilities. Extreme cases are the likelihood weighting method and the bounded variance method: these methods do not even get any result, since all the configurations in the sample get a zero weight. This network has two features which make traditional methods fall into troubles. In one hand, the presence of probabilities very close to zero due to the evidence, and, on the other hand, the high connectivity of

Table 1
Results of the experiment with observations

Maximum	ALG. IS_T			ALG. IS		
potential size	Variance	Time (s)	Error	Variance	Time (s)	Error
27	2.8002	114.84	0.3915	14.2454	520.50	0.8652
54	0.7265	121.00	0.3066	15.2317	525.38	0.7428
108	0.1058	135.32	0.2287	0.5693	610.50	0.2882
216	0.0036	136.91	0.2131	0.5699	612.52	0.2958

Table 2
Results of the experiment with no observations

Maximum	ALG. IS_T			ALG. IS		
potential size	Variance	Time (s)	Error	Variance	Time (s)	Error
27	0.0	204.47	0.3582	0.8562	923.46	0.4520
54	0.0	204.56	0.3572	0.8628	922.35	0.4511
108	0.0	205.84	0.3583	0.4182	967.70	0.4056
216	0.0	208.07	0.3586	0.4183	970.08	0.4057

the graph, which results in very large potentials when computing the sampling distribution. Thus, we think this is a good example to check whether our new algorithms work or not.

We have carried out 4 experiments for each case (with and without observations). The difference among them is the maximum potential size: 27 in the first, 54 in the second, 108 in the third and 216 in the fourth. The sample size was 5000 for all the experiments, and in the case of IS_T, we have considered a parameter $\varepsilon = 0.01$. Each trial has been repeated 100 times to average the results. In each trial, we have calculated the computing time, error and variance of the weights.

For one variable X_l, the error is measured as follows (see Fertig and Mann, 1980):

$$G(X_l) = \sqrt{\frac{1}{|U_l|} \sum_{a \in U_l} \frac{(\hat{p}(a|e) - p(a|e))^2}{p(a|e)(1 - p(a|e))}}, \tag{19}$$

where $p(a|e)$ is the true *posterior* probability, $\hat{p}(a|e)$ is the estimated value and $|U_l|$ is the number of cases of variable X_l. For a set of variables X_I, the error is:

$$G(X_I) = \sqrt{\sum_{i \in I} G(X_i)^2}. \tag{20}$$

The experiments have been carried out in an Intel Pentium II 450 MHz computer, with 384MB of RAM and operating system Linux 2.0.36. The results are displayed in Tables 1 and 2. The programs are written in Java language. Both the programs and the data set are available via ftp at
`ftp://rojo.ualm.es/pub/Elvira/ImportanceSampling`.

With regard to the obtained results, the following can be said:

- Without observations our algorithm is really fast and optimal (the variance of the weights is equal to 0). The pre-computation is very fast: the deletion procedure will select variables from leaves backward (variables belonging to only one potential). When deleting a variable, we sum in the conditional probability of this variable and then we obtain a potential equal to 1 in every case. In this way an exact sampling distribution is obtained in a very short time. The simulation associated to this exact sampling distributions is equivalent to the one carried out by the likelihood weighting and bounded variance algorithms, sharing all their advantages for the case of no observations.
  It is important to remark that in Jensen et al. (1995) the results of blocking Gibbs sampling are reported for the case of no observations and this algorithm has a clearly superior behaviour in this case.
- In both cases, tables are much slower than probability trees and the results are worse. Specially in the case of no observations, the differences on time are very significative: tables do not have a procedure to approximate computations in such a flexible way as trees do.
- The IS_T algorithm has a good behaviour even with very small trees (of size 27). In practice, we will usually afford bigger trees, but we wanted to test the algorithm in really hard conditions.

7. Conclusions

In this paper we have proposed an approximate propagation method able to deal with large networks. In such networks, it is shown that using probability trees instead of tables provides benefits, since trees allow to represent more efficiently the most important regions of a probability distribution, what leads to an improvement of importance sampling, as can be deduced from the experimental results.

There were methods able to deal with very large networks, like blocking Gibbs sampling, due to Jensen et al. (1995). However, there are cases in which our algorithm is superior to blocking Gibbs sampling, as in the experiment without observations. Situations in which blocking Gibbs behaves better than IS_T algorithm can exist, though we do not know them. But, in general, we think that it will be impossible to have a single algorithm better than all the other in every situation. The importance sampling based on approximate pre-computation provides a new paradigm to solve large problems, enlarging the set of cases that can be approximated and allowing future improvements. An example of a situation which can be solved with this new algorithm arises in problems of Information Retrieval. Campos et al. (1998) provide a model for retrieving information in which they learn a polytree with the relationships between terms or keywords of a collection of documents. They used it as a thesaurus, to expand queries adding new terms to them. Computation was easy because of the polytree structure. Experiments showed that expanded queries improved the recuperation of documents using standard procedures (Smart). In the paper, they propose to expand the terms graph with the document variables, creating structures representing the relationships among them, in such a way that the complete graph could be used as a document recuperation system, by calculating the marginal probability of each document given the terms. The documents should be added to the original network putting them as children of term variables. Our algorithm is specially appropriate for this task: documents are not observed and do not have observed descendents; then it is possible to delete these variables following the deletion procedure as was explained earlier for a network without observations. All the conditional probabilities are approximated by the constantly equal to 1 potential, which can be deleted afterwards. When we finish deleting document variables, we have an hypertree in which an exact deletion can be carried out in linear time. This fast procedure calculates exact sampling distributions and the quality of the sample of our procedure will be optimal (all the weights are equal). This allows to solve classical information retrieval problems with more than 10.000 variables. It is not at all clear how blocking Gibbs sampling would handle this kind of problems.

An important point to remark is that some kind of pre-computation is always necessary to solve large network problems by simulation, allowing to use as much as possible information stored in other parts of the network when simulating a concrete variable. If we only use local information, we are sure to obtain a poor behaviour when conflicting information is stored in different parts of the graph. In the case of Gibbs sampling algorithms, this is avoided by simulating values compatible with previous instantiations of the variables in the graph. Our procedure presents a direct way of doing it.

One important feature of trees is their flexibility, what makes the algorithm proposed susceptible of being improved, for instance, using entropy criteria to select the trees that must be combined. The possibility of approximating uniform regions in a distribution by an average value to free space for storing more of the most informative values, allow many modifications to the algorithm that were not possible before.

Other future alternative is to use different samples to estimate each one of the necessary joint probabilities $p(x_k', e)$ and $p(e)$ as in Dagum and Luby (1997). If we are interested in only a few conditional probabilities this may improve the final results without an increasing of the computations, but experimental evaluations would be necessary.

About the future directions of research we want to point out the possibility of improving the initial approximations by several iterations of the approximate stage. It has been shown in Kozlov and Koller (1997) that tree approximations can improve if we take into account information about all the potentials each time we calculate a tree approximation. This gives rise to an iterative propagation algorithm which can be organized in a join tree and such that the approximations in one part of the tree are improved by better approximations in another part. This methodology can improve our initial approximations based only in a deletion procedure.

Another interesting possibility of this algorithm is the determination of bad approximate steps. If when we simulate a variable the normalization factor of the combined tree is 0, then the final weight is 0, and this configuration is finally discarded. The only way in which this may happen is due to a previous approximate deletion of the variable. If this zero normalization value is repeated very often for a given variable, then we can invest more effort (some additional computing time) in the approximate deletion of this variable, trying to improve the quality of the approximation so that 0 weights are avoided in the future. The same procedure can be applied to very small values.

Acknowledgements

We want to thank Claus Skaaning Jensen for providing the network used in the experiment.

We are also very grateful to the anonymous referees for their valuable comments and suggestions.

References

Boutilier, J., Friedman, N., Goldszmidt, M., Koller, D., 1996. Context-specific independence in Bayesian networks. In: Horvitz, E., Jensen, F. (Eds.) Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence. Morgan & Kauffman, San Francisco, CA, pp. 115–123.

Campos, L.M., de Fernández, J.M., Huete, J., 1998. Query expansion using Bayesian networks. In: Cooper, G.F., Moral, S. (Eds.) Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence. Morgan & Kauffman, San Francisco, CA, pp. 53–60.

Cano, A., Moral, S., 1995. Heuristic algorithms for the triangulation of graphs. In: Bouchon-Meunier, B., Yager, R., Zadeh, L. (Eds.) Advances in Intelligent Computing. Springer, Berlin, pp. 98–107.

Cano, A., Moral, S., 1997. Propagación exacta y aproximada con árboles de probabilidad. In: Actas de la VII Conferencia de la Asociación Española para la Inteligencia Artificial, pp. 635–644.

Cano, J., Hernández, L., Moral, S., 1996. Importance sampling algorithms for the propagation of probabilities in belief networks. Int. J. Approx. Reasoning 15, 77–92.

Cooper, G., 1990. The computational complexity of probabilistic inference using Bayesian belief networks. Artif. Intell. 42, 393–405.

Dagum, P., Luby, M., 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. Artif. Intell. 60, 141–153.

Dagum, P., Luby, M., 1997. An optimal approximation algorithm for Bayesian inference. Artif. Intell. 93, 1–27.

Dechter, R., Rish, I., 1994. Directional resolution: The Davis–Putnam procedure, revisited. In: Doyle, J., Sandewall, E., Torasso, P. (Eds.) Principles of Knowledge Representation and Reasoning (KR-94), pp. 134–145.

Dechter, R., Rish, I., 1997. A scheme for approximating probabilistic inference. In: Geiger, D., Shenoy, P. (Eds.) Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence. Morgan & Kauffman, San Francisco, CA, pp. 132–141.

Fertig, K., Mann, N., 1980. An accurate approximation to the sampling distribution of the studentized extreme-valued statistic. Technometrics 22, 83–90.

Fung, R., Chang, K., 1990. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In: Henrion, M., Shachter, R., Kanal, L., Lemmer, J. (Eds.) Uncertainty in Artificial Intelligence, vol. 5. North-Holland, Amsterdam, pp. 209–220.

Fung, R., Favero, B.D., 1994. Backward simulation in Bayesian networks. In: de Mántaras, R.L., Poole, D. (Eds.) Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence. Morgan & Kauffman, San Mateo, CA, pp. 227–234.

Geweke, J., 1989. Bayesian inference in econometric models using Monte Carlo integration. Econometrica 57, 1317–1339.

Henrion, M., 1988. Propagating uncertainty by logic sampling in Bayes' networks. In: Lemmer, J., Kanal, L. (Eds.) Uncertainty in Artificial Intelligence, vol. 2. North-Holland, Amsterdam, pp. 317–324.

Hernández, L., Moral, S., Salmerón, A., 1996. Importance sampling algorithms for belief networks based on approximate computation. Proceedings of the Sixth International Conference IPMU'96, vol. 2, Granada, Spain, pp. 859–864.

Hernández, L., Moral, S., Salmerón, A., 1998. A Monte Carlo algorithm for probabilistic propagation in belief networks based on importance sampling and stratified simulation techniques. Internat. J. Approximate Reasoning 18, 53–91.

Jensen, C., Kong, A., Kjærulff, U., 1995. Blocking gibbs sampling in very large probabilistic expert systems. Internat. J. Human–Computer Studies 42, 647–666.

Jensen, F., 1996. An Introduction to Bayesian Networks. UCL Press, London.

Kjærulff, U., 1992. Optimal decomposition of probabilistic networks by simulated annealing. Statist. Comput. 2, 1–21.

Kozlov, D., Koller, D., 1997. Nonuniform dynamic discretization in hybrid networks. In: Geiger, D., Shenoy, P. (Eds.) Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence. Morgan & Kauffman, San Francisco, CA, pp. 302–313.

Kullback, S., Leibler, R., 1951. On information and sufficiency. Ann. Math. Statist. 22, 76–86.

Lauritzen, S., Spiegelhalter, D., 1988. Local computations with probabilities on graphical structures and their application to expert systems. J. Roy. Statist. Soc. Ser. B 50, 157–224.

Pearl, J., 1987. Evidential reasoning using stochastic simulation of causal models. Artif. Intell. 32, 247–257.

Pearl, J., 1988. Probabilistic Reasoning in Intelligent Systems. Morgan-Kauffman, San Mateo, CA.

Poole, D., 1997a. Exploiting contextual independence and approximation in belief network inference. Technical Report, University of British Columbia.

Poole, D., 1997b. Probabilistic partial evaluation: exploiting rule structure in probabilistic inference. In: Proceedings of the 15th International Joint Conference on AI (IJCAI-97), pp. 1284–1291.

Quinlan, J., 1986. Induction of decision trees. Machine Learning 1, 81–106.

Rubinstein, R., 1981. Simulation and the Monte Carlo Method. Wiley, New York.

Shachter, R., Peot, M., 1990. Simulation approaches to general probabilistic inference on belief networks. In: Henrion, M., Shachter, R., Kanal, L., Lemmer, J. (Eds.) Uncertainty in Artificial Intelligence, vol. 5. North-Holland, Amsterdam, pp. 221–231.

Zhang, N., Poole, D., 1996. Exploiting causal independence in Bayesian network inference. J. Artif. Intell. Res. 5, 301–328.