Theory and Decision manuscript No.
(will be inserted by the editor)

PRM Inference Using Jaffray & Faÿ's Local Conditioning

Christophe Gonzales · Pierre-Henri Wuillemin

Received: date / Accepted: date

Abstract Probabilistic Relational Models (PRM) are a framework for compactly representing uncertainties (actually probabilities). They result from the combination of Bayesian networks (BN), Object Oriented languages and Relational Models. They are specifically designed for their efficient construction, maintenance and exploitation for very large scale problems, where Bayesian networks are known to perform poorly. Actually, in large scale problems, it is often the case that BNs result from the combination of patterns (small BN fragments) repeated many times. PRMs exploit this feature by defining these patterns only once (the so-called PRM's *classes*) and using them through multiple *instances*, as prescribed by the Object Oriented paradigm. This design induces low construction and maintenance costs. In addition, by exploiting the classes' structures, PRM's state-of-the-art inference algorithm "Structured Variable Elimination" (SVE) significantly outperforms BN's classical inference algorithms (e.g., Variable Elimination (VE), Local Conditioning (LC)). SVE is actually an extension of VE that simply exploits classes to avoid redundant computations. In this paper, we show that SVE can be enhanced using Local Conditioning. Although LC is often thought as being outperformed by VE-like algorithms in BNs, we do think that it should play an important role for PRMs because its features are very well suited for best exploiting PRM classes. In this paper, relying on Faÿ and Jaffray's works, we show how Local Conditioning can be used in conjunction with Variable Elimination and deduce an extension of SVE that outperforms it for large scale problems. Numerical experiments highlight the practical efficiency of our algorithm.

Keywords Bayesian networks · PRM · Probabilistic Relation Models · Lifted inference · SVE · Structured Variable Elimination · Local Conditioning

This work was supported by the French National Research Agency, project ANR SKOOB, grant number: ANR Project 07 TLOG 021.

LIP6 – UPMC
Département DESIR, 104, avenue du président Kennedy, F-75016 Paris
E-mail: firstname.lastname@lip6.fr

1 Introduction

For the past twenty years, Bayesian networks (BN) have been the most popular knowledge representation framework for reasoning under uncertainty. They actually have many attractive features: first, they can represent very compactly different popular uncertainty models such as probabilities (Pearl, 1988) or belief functions (Cozman, 2000). This compacity enables BNs to handle probability distributions that exceed by far the storage capacity of all modern computers (Naïm et al, 2007). Second, their learning from either databases or experts can be achieved efficiently and is quite flexible as different sources of data can be mixed together with the guarantee that the result is actually a probability distribution (Heckerman, 1996). Finally, many powerful exact inference techniques exploit BN's structures to answer all sorts of probabilistic queries including *belief updating*, i.e., computing the posterior probability of some random variables given a set of observations (Pearl, 1988; Peot and Shachter, 1991; Jensen et al, 1990; Shafer, 1996; Madsen and Jensen, 1999; Faÿ and Jaffray, 2000; Allen and Darwiche, 2003), *finding the most probable explanations* (MPE), i.e., computing a maximum probability assignment of unobserved random variables (Nilsson, 1998), *finding the maximum a posteriori hypothesis* (MAP), i.e., computing an assignment to a subset of unobserved variables maximizing their probability (Dechter, 1999; Park and Darwiche, 2003; Sun et al, 2007; Yuan and Hansen, 2009).

This success story stimulated the need for handling problems of ever increasing size. However, BNs often turn out to be inadequate for very large scale problems such as, e.g., military situation assessment where the goal is to evaluate the uncertain states of numerous battalions in a battlefield (Mahoney and Laskey, 1996; Pfeffer et al, 1999). BNs are actually static graphical representations, i.e., they are designed to encode probability distributions over a given set of random variables and are not able to adjust their graphical structure to shifting situations where the set of random variables is not constant over time. Therefore, whenever the number of battalions on the battlefield or their configuration changes, a fresh new BN must be designed to take into account the new uncertain situation. Moreover, in large scale problems, many parts of the BN are repetitions of the same pattern: for instance, many battalions have the same configuration, and creating them all within the BN structure requires numerous copy/paste operations. This is of course an inefficient design technique for very large scale problems but, in addition, by not notifying the BN about these repeated patterns, this one is unable exploit this knowledge to speed-up probabilistic inference. All these disadvantages led researchers to seek for BN extensions with a much higher descriptive power. State-of-the-art extensions can be roughly divided into two main classes: i) those resulting from the combination of First Order Logic with uncertainty representation, e.g., Markov Logic Networks, Bayesian Logic (Getoor and Taskar, 2007); and ii) those combining BNs with Object-Oriented languages and Relational Models, e.g., Probabilistic Relational Models (Pfeffer, 2000; Getoor et al, 2007), Multi-Entity Bayesian Networks (Laskey, 2008) and Relational Bayesian Networks (Jaeger, 1997).

In this paper, we focus on Probabilistic Relational Models (PRM) because they are very expressive and flexible. Our aim is show how the PRM's state-of-the-art inference algorithm called Structure Variable Elimination (SVE) (Pfeffer, 2000) can be improved by combining it with Local Conditioning, a BN-based inference method introduced in Diez (1996) and fully developed mathematically in Faÿ and Jaffray (2000). Local Conditioning (LC) is basically a BN "directed" inference algorithm, that is, it relies on the very BN graphical structure to perform its computations. As such, it is often thought

as being outperformed by BN's Variable Elimination-based inference algorithms that rely on BN-induced undirected graphs optimized for computations (Jensen et al, 1990; Zhang and Poole, 1994; Dechter, 1999; Madsen and Jensen, 1999). This belief makes sense for pure BN inference as it was shown in Shachter et al (1994) that some conditioning methods were special cases of Variable Elimination. However, inference methods in PRMs depart significantly from those used in classical BNs and we will show that Jaffray and Faÿ's pioneering work on Local Conditioning can serve as the foundation for new improved versions of SVE in PRMs.

The paper is organized as follows. In Section 2, we recall the basics of Bayesian networks: we define BNs and present two of their inference algorithms: Variable Elimination and Local Conditioning. In Section 3, we present PRMs as well as Structure Variable Elimination, their inference engine. Section 4 is devoted to our combination of SVE and LC. In particular, we show experimental results highlighting its efficiency. Finally, we conclude in Section 5.

2 Bayesian Networks

Bayesian networks are compact representations of joint probability distributions over sets of random variables. Although they can support continuous variables (Lauritzen, 1992; Cowell et al, 2007; Kjærulff and Madsen, 2008), we shall assume throughout this paper that all random variables have a finite domain. This actually simplifies our exposition and does not restrict the key ideas of the paper. For any random variable X_i, $\mathrm{Dom}(X_i)$ denotes the domain of X_i, that is, the set of possible values for X_i, and the probability distribution $P(X_i)$ refers to $P(X_i = x_i)$ for all $x_i \in \mathrm{Dom}(X_i)$.

Definition 1 A Bayesian network is a triple $(\mathbf{V}, \mathbf{A}, \mathbf{P})$, where $\mathbf{V} = \{X_1, \ldots, X_n\}$ is a set of random variables; $\mathbf{A} \subseteq \mathbf{V} \times \mathbf{V}$ is a set of arcs which, together with $\mathbf{V}$, constitutes a directed acyclic graph $\mathbf{G} = (\mathbf{V}, \mathbf{A})$; $\mathbf{P} = \{P(X_i|\mathbf{Pa}_i) : X_i \in \mathbf{V}\}$ is the set of conditional probability tables (CPT) of each node/random variable X_i given the values of its parents $\mathbf{Pa}_i$ in graph $\mathbf{G}$. The BN represents a joint probability distribution over $\mathbf{V}$ having the product form: $P(\mathbf{V}) = P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}_i)$.

Thus, the BN of Figure 1 represents a probability distribution over $\mathbf{V} = \{A, B, C, D, E, F, G, H\}$ decomposable as $P(\mathbf{V}) = P(A)P(B)P(C|A)P(D|A, B)P(E|C)P(F|D)$ $P(G|E, F)P(H|F)$. Note the compacity of the representation: if all random variables are of domain size 10, storing in extension the joint probability would require a table of size 10^8, whereas the sum of the sizes of the tables stored in the BN is only 2420 (2 tables with 1000 cells, 4 with 100 cells and 2 with 10 cells). By definition, the arcs linking the nodes represent direct probabilistic dependencies between the corresponding random variables. Longer paths such as $C \leftarrow A \rightarrow D$ represent indirect dependencies:

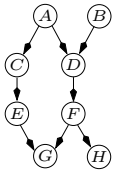


Fig. 1 A Bayesian network.

on Fig. 1, C and D are dependent but only through the value of A. For instance, A may be the age of a person, C be her reading skill and D her shoe size; D and C are probabilistically dependent since knowing that her shoe size is only 2 implies that she is a baby and, hence, that she does not know how to read. The lack of an arc between two nodes reflects the lack of a direct probabilistic dependence between these two nodes.

BNs have essentially three major advantages over joint probabilities stored in extension. First, their compacity enables handling distributions that exceed by far the storing capacity of modern computers (Naïm et al, 2007). Second, by their very definition, their construction requires only the specification of conditional probabilities. As a consequence, different sources of data can be used to construct different parts of the network, the product of these conditional probabilities being guaranteed to produce a joint probability. Finally, the BN's graphical structure can be efficiently exploited for answering all sorts of probabilistic queries such as the computations of marginal *a priori* and *a posteriori* probabilities (Jensen et al, 1990; Shafer, 1996; Diez, 1996; Dechter, 1999; Madsen and Jensen, 1999; Faÿ and Jaffray, 2000; Allen and Darwiche, 2003), of the most probable explanations (MPE) (Nilsson, 1998), etc. Marginal *a posteriori* probability queries are of utmost importance in practical applications: consider a BN designed for the diagnosis of potential malfunctions in a nuclear plant. This diagnosis essentially amounts to estimating the probability distribution of node "problem" given (i.e., conditionally to) data brought by sensors within the power plant. Optimal decisions can then be made on the basis of this *a posteriori* marginal probability.

In this paper, we will thus focus only on the efficient computation of the marginal probability of a given node X_i conditionally to some evidence (new information) on some variables of $\mathbf{V}$, i.e., we received some noisy information $e_{i_1}, \ldots, e_{i_k}$ about some variables $X_{i_1}, \ldots, X_{i_k}$ respectively and we wish to compute $P(X_i|e_{i_1}, \ldots, e_{i_k})$. For instance, if $\mathrm{Dom}(X_{i_j}) = \{1, 2, 3, 4\}$, e_{i_j} may be "we now know that X_{i_j} can have neither value 1 nor 3". Such evidence is entered into the BN through the conditional probability vector $P(e_{i_j}|X_{i_j})$. In our example, $P(e_{i_j}|X_{i_j}) = [0, 1, 0, 1]$, that is, the vector contains 1's for X_{i_j}'s remaining possible values and 0's for the now impossible values. Evidence may also be noisy: e_{i_j} = "our belief is that there is not more than a 20% chance that X_{i_j} can have value 1". In this case, $P(e_{i_j}|X_{i_j}) = [0.2, 1, 1, 1]$. In addition, it is assumed that e_{i_j} is independent of the rest of the network conditionally to X_{i_j}. This makes sense as knowing the "true" value of X_{i_j} is the most informative possible evidence. Therefore, conditionally to X_{i_j}, that is, given the knowledge of the "true" value of X_{i_j}, evidence e_{i_j} can only bring redundant information to the rest of the network and is thus independent of it. As a consequence of this hypothesis, $P(\mathbf{V}, e_{i_1}, \ldots, e_{i_k}) = P(\mathbf{V}) \times \prod_{j=1}^{k} P(e_{i_j}|X_{i_j})$. Hence, after substituting $P(X_{i_j}|\mathbf{Pa}_{i_j})$ by $P(X_{i_j}|\mathbf{Pa}_{i_j}) \times P(e_{i_j}|X_{i_j})$, the BN represents the *a posteriori* joint probability distribution $P(\mathbf{V}, e_{i_1}, \ldots, e_{i_k})$.

2.1 Computation of *a posteriori* Marginal Probabilities by Variable Elimination

Many different algorithms do exist from computing marginal probabilities in BNs. Some of them exploit the very structure of the BN (Pearl, 1988; Peot and Shachter, 1991; Diez, 1996; Faÿ and Jaffray, 2000; Allen and Darwiche, 2003) while others exploit other derived graphs called either a *junction tree* (Jensen et al, 1990; Jensen, 1996; Shafer, 1996; Shenoy, 1997; Madsen and Jensen, 1999) or a *pseudo tree* (Dechter, 1999). However, all these variants exploit the same key idea: that of variable elimination. Formally, let $\mathbf{Z} = \{X_{i_1}, \ldots, X_{i_k}\} \subseteq \mathbf{V}$ be a set of nodes on which we received new

evidence $e_{i_1}, \ldots, e_{i_k}$. For simplicity, let us call $e_{\mathbf{Z}} = \{e_{i_1}, \ldots, e_{i_k}\}$. Then:

$$P(X_i|e_{\mathbf{Z}}) = \sum_{\mathbf{V}\backslash\{X_i\}} P(\mathbf{V}|e_{\mathbf{Z}}) = \sum_{\mathbf{V}\backslash\{X_i\}} \frac{P(\mathbf{V}, e_{\mathbf{Z}})}{P(e_{\mathbf{Z}})} \propto \sum_{\mathbf{V}\backslash\{X_i\}} P(\mathbf{V}, e_{\mathbf{Z}}).$$

Here, there is no need to compute explicitly the fraction's denominator because $P(e_{\mathbf{Z}})$ is just a normalizing constant ensuring that $\sum_{X_i} P(X_i|e_{\mathbf{Z}}) = 1$. Therefore, the only quantity that needs be computed is $\sum_{\mathbf{V}\backslash\{X_i\}} P(\mathbf{V}, e_{\mathbf{Z}}) = P(X_i, e_{\mathbf{Z}})$ which can be normalized afterward by dividing it by $\sum_{X_i} P(X_i, e_{\mathbf{Z}})$. The idea to achieve these computations is simply to remove the variables of $\mathbf{V}\backslash\{X_i\}$ one by one by marginalizing them out, hence the name of the algorithm "Variable Elimination" (VE). Note however that the order in which variables are "eliminated" is critical for the efficiency of the computation. Finding the optimal order is unfortunately NP-Hard (Arnborg et al, 1987) but many efficient heuristics based on the BN graphical structure can be found in the literature (Rose et al, 1976; Kjærulff, 1990; Shoikhet and Geiger, 1997; van den Eijkhof and Bodlaender, 2002).

Before giving a formal algorithm for Variable Elimination, let us show on an example how it can be applied and how it is related to the BN's graphical structure. Consider the BN of Fig. 1 and assume that nodes B and E have received evidence e_B and e_E respectively, as shown in Fig. 2.a. Let us compute $P(C, e_B, e_E)$. As remarked in the preceding subsection, the joint probability distribution $P(\mathbf{V}, e_B, e_E)$ is equal to:

$$P(A)P(B)P(C|A)P(D|A,B)P(E|C)P(F|D)P(G|E,F)P(H|F)P(e_B|B)P(e_E|E) \quad (1)$$

and is compactly represented by the BN of Fig. 2.a. The first step of the algorithm consists of creating a new undirected graph called a *moral graph* (see Fig. 2.b) whose nodes are the random variables of $\mathbf{V}$ and whose edges join those nodes that appear in a same conditional probability of the decomposition of $P(\mathbf{V}, e_B, e_E)$. For instance, $P(G|E,F)$ induces edges (E,F), (E,G) and (F,G). Maximal complete subgraphs of the moral graph thus represent the different "factors" of the decomposition of the joint probability. The VE algorithm will enforce this property all along the elimination process. Let G be the first node to be removed from Eq. (1), that is, Eq. (1) is substituted by $\sum_G P(\mathbf{V}, e_B, e_E)$. As G appears only in factor $P(G|E,F)$, it is sufficient to substitute it by $\sum_G P(G|E,F)$ in Eq. (1) to get $\sum_G P(\mathbf{V}, e_B, e_E)$. Let us denote by $\phi_1(E,F)$ table $\sum_G P(G|E,F)$. Note that this is a function defined over $\mathrm{Dom}(E)\times\mathrm{Dom}(F)$. The resulting joint distribution is $P(A)P(B)P(C|A)P(D|A,B)P(E|C)P(F|D)\phi_1(E,F)P(H|F)$ $P(e_B|B)P(e_E|E)$ and is represented by the network of Fig. 2.c. In terms of graphical operations, performing $\sum_G P(G|E,F)$ just amounts to removing G and its adjacent edges (those joining it to the other variables in $P(G|E,F)$) from Fig. 2.b. Similarly, H can be marginalized out by substituting $P(H|F)$ by $\phi_2(F) = \sum_H P(H|F)$ and replacing the network of Fig. 2.c by that of Fig. 2.d. Marginalizing out variable B amounts to sum over B all the factors[1] involving B, that is, computing $\phi_3(A,D) = \sum_B P(B)P(e_B|B)P(D|A,B)$ and substituting these 3 potentials by $\phi_3(A,D)$. Here again, the "elimination" graphical counterpart consists of removing B and its adjacent edges. At this point, the joint probability distribution over the remaining variables is:

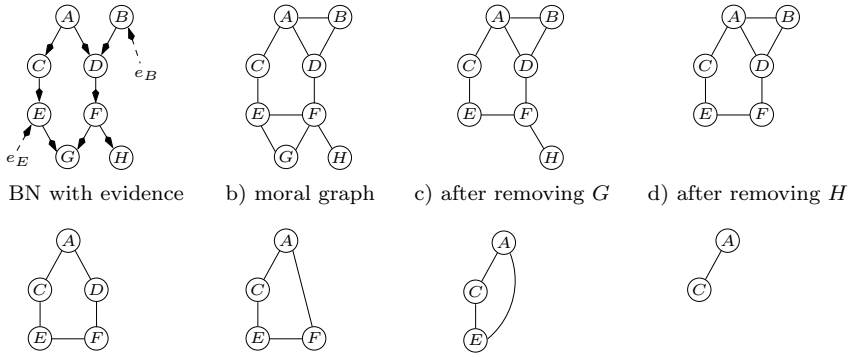
$$P(A)P(C|A)\phi_3(A,D)P(E|C)P(F|D)\phi_1(E,F)\phi_2(F)P(e_E|E) \quad (2)$$

[1] In the literature, factors are usually referred to as "*potentials*".

and is represented by the graph of Fig. 2.e. Until now, each time a node was eliminated, it was sufficient to remove it (and its adjacent edges) from the graph. But consider now the elimination of node D. It amounts to substitute $\phi_3(A, D) \times P(F|D)$, i.e., the product of the potentials involving D in Eq. (2), by $\phi_4(A, F) = \sum_D \phi_3(A, D) \times P(F|D)$. The graphical counterpart operation consists of adding a new edge (A, F) and, then, of removing D and its adjacent edges, thus resulting in the graph of Fig. 2.f. This is the general rule to apply (and actually that which we applied right from the start):

Rule 1 (Node elimination) *To eliminate a node, first create edges between all pairs of its adjacent nodes that were not linked yet (hence creating a maximal complete subgraph, i.e., a clique) and, then, remove the "eliminated" node and its adjacent edges.*

Now eliminate, say, node F. This amounts to substitute $\phi_4(A, F)\phi_1(E, F)\phi_2(F)$ by $\phi_5(A, E) = \sum_F \phi_4(A, F)\phi_1(E, F)\phi_2(F)$ and, in graphical terms, to substitute Fig. 2.f by Fig. 2.g. At this point, the joint distribution is equal to $P(A)P(C|A)P(E|C)P(e_E|E)$ $\phi_5(A, E)$. Thus, the elimination of E leads to substitute $P(E|C)P(e_E|E)\phi_5(A, E)$ by $\phi_6(A, C) = \sum_E P(E|C)P(e_E|E)\phi_5(A, E)$ and Fig. 2.g by Fig. 2.h. Finally, eliminating A is performed by substituting $P(A) \times P(C|A) \times \phi_6(A, C)$ by $\sum_A P(A)P(C|A)\phi_6(A, C)$ $= P(C, e_B, e_E)$ and the computation is completed. There just remains to normalize this quantity to get $P(C|e_B, e_E)$. This process is formalized in Algorithm 1, where $\mathbf{V}$ is the set of random variables of the BN and $\mathbf{P}$ is its set of CPTs.



a) BN with evidence b) moral graph c) after removing G d) after removing H

e) after removing B f) after removing D g) after removing F h) after removing E

Fig. 2 Computing $P(C, e_B, e_E)$ by Variable Elimination.

Rule 1 is important because it characterizes the complexity of the computation:

Proposition 1 (Dechter (1999)) *Let w^* be the maximal number of nodes in the cliques created by Rule 1. Let n denote the number of variables X_i and let d denote the maximal domain size of these variables. Then the space and time complexities of* VE *are $O(nd^{w^*-1})$ and $O(nd^{w^*})$ respectively.*

Note that, whenever the BN is singly-connected —i.e., it has no loops—, it is possible to find an elimination ordering such that the created cliques are precisely those of the BN's moral graph. In such a case, the complexity of VE is polynomial in the size of its input. Otherwise, as shown in Proposition 1, the complexity is doomed to be exponential in the treewidth w^*, hence possibly resulting in intractable instances.

Input: a set of random variables $\mathbf{V}$, a set of CPTs $\mathbf{P}$, a set of evidence $e_{\mathbf{Z}}$, a set of target nodes $\mathbf{X}$
Output: $P(\mathbf{X}, e_{\mathbf{Z}})$
$\mathbf{P} \leftarrow \mathbf{P} \cup \{P(e_{i_j}|X_{i_j}) : e_{i_j} \in e_{\mathbf{Z}}\}$
while $\mathbf{V}\backslash\mathbf{X} \neq \emptyset$ **do**
  let X_j be some node in $\mathbf{V}\backslash\mathbf{X}$
  remove X_j from $\mathbf{V}$
  let $\mathbf{Q}$ be the set of tables in $\mathbf{P}$ containing node X_j
  compute table $q = \sum_{X_j} \prod_{f \in \mathbf{Q}} f$
  $\mathbf{P} \leftarrow (\mathbf{P}\backslash\mathbf{Q}) \cup \{q\}$
end
return *table* $\prod_{f \in \mathbf{P}} f$

Algorithm 1: Variable Elimination for computing $P(\mathbf{X}, e_{\mathbf{Z}})$.

This remark suggests that, if there is a way to transform a multiply-connected BN into a singly-connected one, then computations will probably be faster in the latter than in the former. This is precisely the approach taken by Local Conditioning.

2.2 Computation of *a posteriori* Marginal Probabilities by Local Conditioning

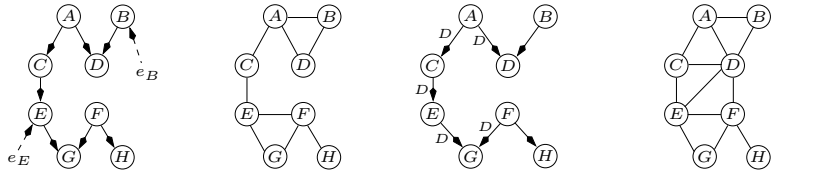
Ever since a paper by Shachter et al (1994), conditioning techniques have been considered less efficient than VE by the BN community. However, this is certainly overreacted as this paper concerned only "*Global Conditioning*", but not in any case other more flexible schemes such a "*Local Conditioning*" (LC). And in fact, it can be shown that VE is a particular case of a variant of LC (Gonzales et al, 2007). LC was introduced through an example in Diez (1996) and was later justified mathematically in Faÿ and Jaffray (2000). The idea is twofold: first, remember that the arcs in a BN represent direct probabilistic dependencies between random variables. As such, eliminating a node from the BN can be viewed as making the relevant information it contains flow toward its neighbors. The elimination scheme can thus be performed directly on the BN, that is, on the directed graph, by sending appropriate messages along the arcs (the information flow) (Pearl, 1988; Peot and Shachter, 1991). Second, if the value of a node, say X_j, is known, then there is no need to keep X_j's outgoing arcs: indeed, after updating accordingly the CPTs of its children, no additional information can flow to them from X_j since the data flowed are probabilities concerning the possible values of X_j and we already have the most informative knowledge, that of the true value of X_j. Conversely, no data needs be flowed from its children to X_j since knowing X_j's true value, no additional data can be probabilistically informative for X_j. Therefore, if we know the "true" values of enough nodes, removing their outgoing arcs may result in a singly-connected network in which computations are of polynomial complexity, as seen in the preceding subsection. Of course, it is most unusual that we know the true values of enough nodes to make the BN singly-connected. But this situation can be enforced by reasoning by cases and this is precisely what is advocated by LC: assume that, once the values of X_{j_1}, X_{j_2} are known, the BN becomes singly-connected, then $P(X_i|e_{\mathbf{Z}}) = \sum_{x_{j_1}, x_{j_2}} P(X_i, X_{j_1}{=}x_{j_1}, X_{j_2}{=}x_{j_2}|e_{\mathbf{Z}})$. But then, for each pair (x_{j_1}, x_{j_2}), this quantity can be computed in a singly-connected network since we know the values of X_{j_1}, X_{j_2}. In the end, there just remains to add the probabilities computed for each pair (x_{j_1}, x_{j_2}) to get the desired marginal *a posteriori* probability of X_i. A complete

mathematical specification of this method can be found in Faÿ and Jaffray (2000). Note that not all the outgoing arcs of the "conditioned" nodes need be removed but only sufficiently many to remove all the loops while keeping the graph connected.

In this paper, our aim is to use VE in conjunction with LC, hence our presentation of the latter will depart somewhat from that of Faÿ and Jaffray (2000) which relies on the directed structure of the BN. Consider the graph of Fig. 3.a, in which arc (D, F) does not exist anymore. Its moral graph is given in Fig. 3.b. The reader can easily check that, using the same elimination order as in the preceding subsection, i.e., G, H, B, D, F, E, A, no additional edge is added to the graph by VE. Now, how can we remove arc (D, F) from the network while still ensuring that our computations are correct? The answer can be found in the following rule derived from Faÿ and Jaffray (2000):

Rule 2 (Arc labeling) *Let* $\mathbf{L} \subseteq \mathbf{A}$ *be the set of arcs removed by local conditioning. Assume that this set keeps the connectedness of the graph and that this one is now singly-connected. For any removed arc* $(X_s, X_t) \in \mathbf{L}$*, there still exists exactly one undirected path between* X_s *and* X_t*. On all the arcs of this path, add label* X_s*.*

Messages flowing on the arcs (X_i, X_j) *of the BN should have the dimensions of these arc's labels as well as that of* X_i*.*



a) BN with evidence b) moral graph c) labelized graph d) LC's moral graph

Fig. 3 Computing $P(C|e_B, e_E)$ by Local Conditioning.

Rule 2 is illustrated on Fig. 3.c, where (D, F) is the only removed arc. In practice, whenever a message is sent on an arc of this network, the nodes of its label cannot be marginalized out. To describe inference in terms of VE, it is sufficient to add dimension D to all the conditional probability tables stored into the nodes at the head of the labeled arcs prior to performing VE. This is achieved by duplicating $|\mathrm{Dom}(D)|$ times the original conditional probability tables. In our case, this amounts to state that the joint probability distribution is decomposable as:

$$\begin{aligned} P(\mathbf{V}, e_B, e_E) = {} & P_D(A)P(B)P(e_B|B)P(C|A,D)P(D|A,B)P(E|C,D) \\ & P(e_E|E)P(F|D)P(G|E,F)P(H|F), \end{aligned}$$

where $P_D(A)$ refers to $P(A)$ duplicated $|D|$ times. This corresponds to the moral graph of Fig. 3.d. Now, VE can be performed as described in Subsection 2.1 (see Table 1). Note that, unlike usual VE, LC sets an additional rule on the possible elimination orders:

Rule 3 *Let* $\mathbf{L} \subseteq \mathbf{A}$ *be the set of arcs removed by* LC*. Let* (X_s, X_t) *be any arc in* $\mathbf{L}$ *and let* $\{X_s, X_{i_1}, \ldots, X_{i_k}, X_t\}$ *be the remaining path linking* X_s *and* X_t*. Then, in* VE*,* X_s *must be eliminated after all nodes* $X_{i_1}, \ldots, X_{i_k}, X_t$ *have been eliminated.*

Table 1 Computing $P(C, e_B, e_E)$ with Local Conditioning.

elim. var	potentials substituted	substituting potentials
G	$P(G\|E,F)$	$\phi_1(E,F)$
H	$P(H\|F)$	$\phi_2(F)$
B	$P(B)P(e_B\|B)P(D\|A,B)$	$\phi_3(A,D)$
F	$\phi_1(E,F)\phi_2(F)P(F\|D)$	$\phi_4(E,D)$
E	$P(E\|C,D)P(e_E\|E)\phi_4(E,D)$	$\phi_5(C,D)$
A	$P(A)P(C\|A,D)\phi_3(A,D)$	$\phi_6(C,D)$
D	$\phi_5(C,D)\phi_6(C,D)$	$P(C,e_B,e_E)$

At first sight, LC seems far less efficient than VE since the moral graph on which it works (Fig. 3.d) corresponds to a much less decomposable probability distribution than the original one. But note that no edge is ever added to this moral graph by LC (observe that functions ϕ_i of Table 1 fit the graph of Fig. 3.d) whereas the usual VE can and actually does add many edges. Therefore, we should not compare VE and LC on the basis of the original moral graphs but rather on the graphs of cliques of the "substituted potentials", that is, the cliques formed during the whole elimination process. Those are displayed in Fig. 4. As can be seen, LC and VE construct very similar graphs. Actually, these graphs have in common the following property: they are triangulated, that is, every cycle of length 4 or more can be cut by at least one edge not belonging to the cycle (called a *fill-in*). For instance, in Fig. 4.b, cycle $ACEGF$ can be cut into two pieces by edges (A, E) and (E, F). The only difference between VE and LC is that the latter sets a constraint on the fill-ins added to the original moral graph: they must be adjacent to the node at the tail of the arc removed by Local Conditioning. For instance, on Fig. 4.a, LC removed arc (D, F), hence all fill-ins, i.e., (D, C) and (D, E), are adjacent to D. If we allow LC to add new edges to the moral graph before performing its arc removals, this constraint on fill-ins is sufficiently relaxed that VE becomes a special case of LC (Gonzales et al, 2007).

a) LC's triangulated graph b) VE's triangulated graph

Fig. 4 The "substituted potentials" graphs of VE and LC.

3 Probabilistic Relational Models

Bayesian networks have been widely used in real-world applications. But, over the years, an increasing need for tractable large scale complex systems arose, for which BNs turned out to be inadequate. Actually, in such systems, the effort they require for their construction, their maintenance and their exploitation for answering probabilistic queries is unrealistically high to be of practical interest. The BN language contains only the concepts of random variables and their probabilistic dependencies. This is simply too basic a language for very large scale problems where advanced concepts such

as "similarity" between random variables, repeated "patterns" in BN sub-structures, "contexts", are compulsory to be efficient. This led to the development of new BN-based graphical languages having these advanced concepts as part of their primitives. These new models can mainly be divided into two parts : i) those that were derived from the unification of First Order Logic with uncertainty representations, e.g., Markov Logic Networks, Bayesian Logic (Getoor and Taskar, 2007); and ii) those that resulted from analogies with Object-Oriented languages and Relational Models, e.g., Probabilistic Relational Models (Pfeffer, 2000; Getoor et al, 2007), Multi-Entity Bayesian Networks (Laskey, 2008), Relational Bayesian Networks (Jaeger, 1997).

To simplify our presentation without loss of generality, we focus in this paper on Probabilistic Relational Models (PRM). We now provide a minimal set of definitions of the primitives of the PRM language. For a more detailed presentation of PRMs, see Pfeffer (2000) and Getoor and Taskar (2007).

3.1 PRM Specification

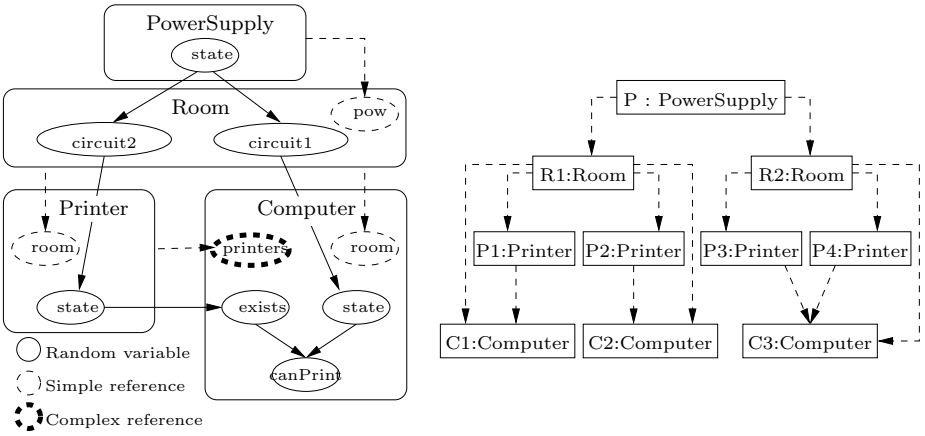
PRMs exploit the Object Oriented paradigm to specify large scale Bayesian networks. Roughly, in an Object Oriented programming language like Java or C++, a program is made of a set of *classes*, which are generic definitions of complex data structures, and of a set of *instances*, which are the instantiations of these data structures that are created and handled at runtime. For instance, to manage the employees of a company, we may define a generic class `employee` whose fields are the name and the address of an employee. For a given employee, say John Smith, we may create an instance of `employee` with the name field set to `John Smith` and the address field set to `New York, NY, USA`. We may also create a second instance of class `employee` for another person by setting this instance's name field to `Mary Doe` and its address field to `Paris, France`. In the PRM framework, classes correspond to BN fragments and a large scale BN is created by putting together multiple instances of these fragments. The resulting network is called a *system* or a *relational skeleton*. Of course, as in usual Object Oriented programming languages, PRM support inheritance, i.e., it is possible to define subclasses specializing general classes (e.g., `engineer` and `manager` may be subclasses of `employee`). PRM also support encapsulation, i.e., classes may include other classes (here BN fragments) as some of their fields. Encapsulation requires an additional notion, that of a reference: as classes are only definitions, defining that a given field of a class `X` is of class `Y` does not give any insight at runtime as to which instance of class `Y` a given instance of type `X` should refer to. *References* fill this gap by indicating at runtime which particular instances should be used by encapsulation. They are in essence functions mapping instances of class `X` to instances of class `Y`. For instance, let `department` be a new class for our company and let `accounting` and `engineering` be two instances of this class. Let us add a new field "dpt" of type `department` to our `employee` class. Then, using references, it is possible to express that John Smith's "dpt" field is actually `engineering`. As can be seen, the richness of the PRM language enables easily to hierarchically decompose a large scale complex system into multiple subsystems easier to handle. Let us now formalize the above concepts:

Definition 2 (Classes and instances) A *class* $\mathfrak{C}$ is defined by a Directed Acyclic Graph (DAG) over a set of random variables $\mathbf{V}(\mathfrak{C})$, a set of references (or slots) $\mathbf{R}(\mathfrak{C})$, and a probability distribution over $\mathbf{V}(\mathfrak{C})$. To refer to a given random variable X (resp.

reference ρ) of class $\mathfrak{C}$, we use the standard Object Oriented notation $\mathfrak{C}.X$ (resp. $\mathfrak{C}.\rho$). In addition, $\mathbb{C}$ denotes the set of classes of a PRM.

An *instance* $\mathfrak{c}$ is the use (or the instantiation) of a given class $\mathfrak{C}$ in a system s. There are usually numerous instantiations of a given class $\mathfrak{C}$ in a system. $\mathbf{I}(\mathfrak{C})$ denotes the set of all instantiations of $\mathfrak{C}$. Notation $\mathfrak{c}.X$ (resp. $\mathfrak{c}.\rho$) refers to the instantiation of $\mathfrak{C}.X \in \mathbf{V}(\mathfrak{C})$ (resp. $\mathfrak{C}.\rho \in \mathbf{R}(\mathfrak{C})$) in $\mathfrak{c}$. By abuse of notation, we denote the sets of such instantiations as $\mathbf{V}(\mathfrak{c})$ and $\mathbf{R}(\mathfrak{c})$ respectively.

Thus, a class defines a family of objects (or instances) sharing the same properties: DAG, random variables and class references. The DAG, together with the random variables, defines a BN fragment. As a consequence, in a PRM, each random variable $\mathfrak{C}.X$ of $\mathbf{V}(\mathfrak{C})$ corresponds to a node in $\mathfrak{C}$'s own DAG and is assigned a conditional probability table defined over X and its parents $\mathbf{Pa}_X$ in $\mathfrak{C}$. Classes are represented by rounded boxes, as shown in Fig. 5.a. This figure illustrates a diagnosis problem in a local network containing both printers and computers: each printer and computer is located in a room; each room uses the power supply delivered by a dedicated circuit supplied by a main power supply. There may be several computers and/or printers per room and each computer may be connected to zero, one or more printers. This example is modeled by 4 classes: `PowerSupply`, `Room`, `Printer` and `Computer`. Class `Computer` exhibits a BN fragment with 3 random variables `Computer.exists`, `Computer.state` and `Computer.canPrint` which represent the fact that the computer is connected to at least one ready printer, the computer's power state and the fact that the computer is able to print respectively. Random variables `state`, `circuit1` and `cicuit2` represent the power states of `PowerSupply`, `Printer` and `Room` respectively. Arcs contained within a given class form the DAG of the BN fragment of the class. Arcs from one class to another indicate the use of a reference. For instance, arc (`Room.circuit1`,`Computer.state`) indicates that the state of a computer depends on the state of its dedicated electric circuit (and the reference represented by the dashed arc pointing to the dashed ellipse "`room`" shows which one it is). On the graph of Fig. 5.b, we show different instantiations of these classes (displayed as boxes): here, we have a system with 2 rooms, the first one containing 2 computers with one printer attached to each computer, and the second one



a) class dependency graph b) one corresponding system

Fig. 5 A Class Dependency Graph and a system (relational skeleton).

containing only one computer connected to two printers. Note that several instances can be created for a given class (e.g., there are 3 computers). The dashed edges represent the actual references. Those are indeed used both to define dependencies between classes and between instances. Note the compacity of the PRM representation: the PRM classes and instances of Fig. 5 represent the plain BN of Fig. 6. Before going further into the details of PRM, we should give more precise definitions of references:

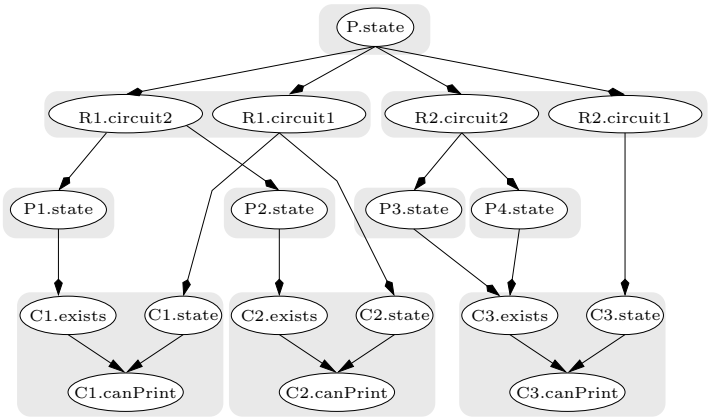


Fig. 6 Grounded network of the system of Figure 5.

Definition 3 (References and reference chains) A *reference* $\mathfrak{C}.\rho$ is a relation between class $\mathfrak{C}$ and another class, say $\mathfrak{B}$. If this relation is binary, the reference is said to be *simple*; if it is n-ary, the reference is called *complex*. Each reference ρ of $\mathfrak{C}$ has an *owner* denoted by $\text{Owner}[\rho] = \mathfrak{C}$ and a *range* denoted by $\text{Range}[\rho] = \mathfrak{B}$. For a given instance $\mathfrak{c} \in \mathbf{I}(\mathfrak{C})$, we denote $\mathfrak{c}.\rho$'s instance owner by $\text{Owner}[\mathfrak{c}.\rho] = \mathfrak{c}$ and $\mathfrak{c}.\rho$'s instance range by $\text{Range}[\mathfrak{c}.\rho] = \mathfrak{b}$, where $\mathfrak{b}$ is an instance of class $\mathfrak{B}$.

A *reference chain*, or *slot chain*, $\rho_1, \rho_2, \ldots, \rho_i$ is a sequence of references in which $\text{Range}[\rho_k] = \text{Owner}[\rho_{k+1}]$ for all $k = 1, \ldots, i-1$. A reference chain is simple if $\forall i, \rho_i$ is simple, else it is said to be complex.

In the graph of Fig. 5.a, class `Computer` contains a *simple* reference whose *owner* is of course `Computer` and whose *range* is `Room`. This reference enables to condition the power state of the computer to the power state of its electric circuit. Class `Computer` also contains a *complex* reference whose *range* is `Printer`. This reference indicates which printers are connected to a given computer. As several printers can actually be connected to a computer (see, e.g., computer `C3` on Fig 5.b), the reference is complex. Given definitions of classes, instances and references, it is now possible to define formally a system:

Definition 4 (System, relational skeleton) A *system* (or relational skeleton) s is a set of instances $\mathbf{I}$ with every reference of every instance $\mathfrak{c} \in \mathbf{I}$ correctly defined, i.e., $\forall \mathfrak{C} \in \mathbb{C}, \forall \mathfrak{c} \in \mathbf{I}(\mathfrak{C}), \forall \rho \in \mathbf{R}(\mathfrak{c}), \exists y \in \mathbf{I}(\text{Range}[\mathfrak{C}.\rho])$ such that $\text{Range}[\mathfrak{c}.\rho] = y$.

As shown with the printers-computers connections, there are cases where the number of instances referenced by a given *reference* varies from one instance to the other

(computer C3 references 2 printers while C1 references only one). As a consequence, the number of parents of some random variables like Computer.exists cannot be known at the class level, i.e., when the class is defined, but is known only at the instance level. This is an issue since the classical method for encoding probabilities in a BN is to use Conditional Probability Tables (CPT), which implies that the number of parents must be known and fixed in advance. Thus, specifying a CPT for Computer.exists would require either to define a family of CPTs at class level —one for each possible number of parents—, or to specify the CPT at instance level for each instance of the Computer class. Both solutions are cumbersome: first, it is impossible to specify a family for all the possible numbers of parents; and, second, specifying CPTs at instance level is unattractive because it requires too much work and, in addition, it is desirable to separate the semantics of these CPTs from the instantiations. This calls for a new, more flexible, way of specifying these CPTs and leads to the concept of aggregators:

Definition 5 (Aggregator) An *aggregator* is a deterministic function defined over a set of random variables $\{\mathfrak{c}_i.X\}$ and such that $\forall i, \mathfrak{c}_i \in \mathbf{I}(\mathfrak{C})$.

In our example of Figure 5, the CPT of Computer.exists is actually an aggregator defined over the states of all the printers connected to a computer and returning a Boolean indicating whether at least one of these printers is ready. For instance, for computer C3, the aggregator returns false if and only if both printers P3 and P4 have a non-ready state. For computer C1, the aggregator returns false if and only if printer P1 is not ready. By enabling the PRM framework to contain a family of generic deterministic functions such as "exists", "forall", "mean", "min", "max", etc, it is possible use aggregators at class level: since these functions are deterministic, it is possible to automatically generate their corresponding CPT when the aggregators are instantiated. As a consequence, using aggregators, it is not necessary to distinguish at class level between variables with usual CPTs and variables with aggregators. Note however that this solution is less expressive than the pseudo code used in MEBN (Laskey, 2008), but it is much simpler to use and to implement. It is also generally sufficient for the majority of real-world problems.

We finish our overview of PRMs with the definition of the grounded BN of a system, that is, the BN that is equivalent to the PRM in terms of uncertainty representation. We consider in this paper only closed world systems, i.e., systems in which all instances and relations between these instances are completely specified[2]. For closed world systems, grounded BNs are defined as:

Definition 6 (Grounded BN) A *grounded Bayesian Network* is the BN equivalent of a system. It is built straightforwardly by creating a node for each random variable of each instance and adding arcs between those nodes in order to respect the instances relations and the DAGs of the classes. Aggregators are mapped into their corresponding CPTs. An algorithm linear in time for generating such grounded BN can be found in Pfeffer (2000). Fig. 6 shows the grounded BN associated with the system of Fig. 5.

PRM support several other features that we will not detail here such as random variable typing, inheritance, structural and existential uncertainties. Even if they are important for the PRM formalism, they are not for the purpose of the paper, i.e., for showing that Jaffray and Faÿ's local condition can improve PRM inference algorithms.

[2] By opposition, in open world systems, all required but absent instances are created as anonymous instances and added to the system.

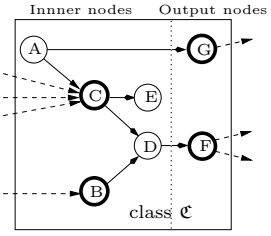
So far, we have described a language much more expressive than the simple *node-arc* language of Bayesian networks. It is important to understand that i) PRMs are an extension of BNs (every BN can be defined as a PRM); and ii) very large and very complex models can easily be built using PRMs. This raises new challenges for inference in such large scale problems. In the next section, we describe SVE, the state-of-the-art inference algorithm in PRMs.

3.2 SVE: Structured Variable Elimination

Structured Variable Elimination (SVE, Pfeffer (2000)) is an extension of Variable Elimination (Zhang and Poole, 1994; Dechter, 1999) adapted to the PRM formalism. The main idea of SVE is to reduce (if not erase completely) redundant computations by performing them as much as possible at class level instead of at instance level. More precisely, classical Variable Elimination performs inference on the grounded BN, that is, at instance level (directly on the random variables) and, as a consequence, it most often performs distinct computations for all the instances of the same class, even when all these computations are exactly identical. To avoid such redundancies, SVE tries to perform as much as possible computations at a class level, that is, whenever possible, it first performs one single computation per class and, then, uses the result for all the instances of the class. As such, it avoids as much as possible the redundancies resulting from the existence of multiple instances of the same class in the PRM. Note however that it is not always possible to preprocess computations at class level for all the instances because some instances contain target random variables (those for which we compute *a posteriori* marginal probabilities) while others contain random variables with different evidence.

To avoid redundancies, SVE exploits the structural information encoded in the classes. To do this, it distinguishes two kinds of random variables/nodes: the inner nodes and the output nodes. The inner nodes of a class $\mathfrak{C}$ are those nodes of $\mathfrak{C}$ whose children all belong to $\mathfrak{C}$. By opposition, output nodes have at least one child defined in another class. In Fig. 7, variables A, B, C, D and E are inner nodes whereas F and G are output nodes. Inner nodes can be considered as internal to the class as they interact with other classes only through the output nodes[3]. As a consequence, they can be eliminated without any interaction with other classes or, in other words, they can be eliminated without affecting in any way the structure of the system. Hence they can be eliminated at class level (once for all the instances) and should be eliminated

[3] In terms of Object Oriented programming languages, inner nodes are very similar to private fields, and output nodes correspond to public fields.



$$P(\mathbf{V}(\mathfrak{C})|\mathbf{Pa}_{\mathbf{V}(\mathfrak{C})}) = P(A)P(B|\mathbf{Pa}_B)P(C|A,\mathbf{Pa}_C) \\ P(D|B,C)P(E|C)P(F|D)P(G|A)$$

elim.	potentials substituted	substituting potentials
A	$P(A)P(C\|A,\mathbf{Pa}_C)P(G\|A)$	$\phi_1(C,G,\mathbf{Pa}_C)$
B	$P(B\|\mathbf{Pa}_B)P(D\|B,C)$	$\phi_2(C,D,\mathbf{Pa}_B)$
C	$P(E\|C)\phi_1(C,G,\mathbf{Pa}_C)\phi_2(C,D,\mathbf{Pa}_B)$	$\phi_3(D,E,G,\mathbf{Pa}_B,\mathbf{Pa}_C)$
D	$P(F\|D)\phi_3(D,E,G,\mathbf{Pa}_B,\mathbf{Pa}_C)$	$\phi_4(E,F,G,\mathbf{Pa}_B,\mathbf{Pa}_C)$
E	$\phi_4(E,F,G,\mathbf{Pa}_B,\mathbf{Pa}_C)$	$\phi_5(F,G,\mathbf{Pa}_B,\mathbf{Pa}_C)$

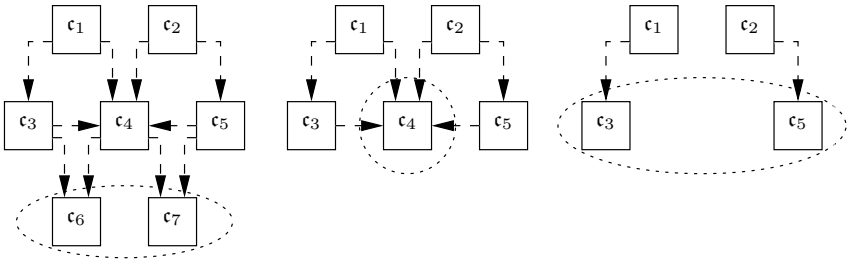
Fig. 7 Inner and outer nodes in a class, and the inner nodes elimination.

first. The table in Fig. 7 shows the potentials resulting from the elimination of inner nodes. Once all inner nodes have been eliminated, the remaining potentials of the class are defined over the output nodes and the set of referenced parents (e.g., in Fig. 7, the actual parents except A of B and C). SVE thus eliminates the inner nodes at class level and uses the resulting potentials each time an instance of the given class is found in the system, thus avoiding redundant computations. There is however a limitation to this method: it cannot be used when some inner nodes have received evidence because evidence usually differ from one instance to the other and thus it is not possible to add the $P(e_{i_j}|X_{i_j})$ probabilities on a per class inner nodes elimination basis. Therefore, for instances that received observations, inner nodes elimination must be performed locally, i.e., at the instance level.

Applied to all classes and instances in a system, this inner elimination builds a set of potentials defined only over output nodes. Now, remember that i) output nodes have at least one child in another class; and ii) the edges in the system (the relational skeleton) correspond to references, i.e., to edges between nodes belonging to different classes. Hence, if there does not exist any reference chain of size greater than 1, the graphical structure of the system precisely corresponds to the BN of the output nodes and we can thus use VE to eliminate all the nodes that are not target nodes. If there exist longer reference chains, then these are represented by directed paths in the graphical structure of the system. In any case, the actual random variables referenced by references and reference chains do belong to ancestor instances in the system. Hence, using VE with a bottom-up elimination order, it is possible to recursively eliminate all the instances from the PRM while ensuring that, at each step, the PRM still represents a joint distribution. The bottom-up order indeed guarantees that before an instance is eliminated, all of its child instances have already been eliminated. Therefore, its output nodes can also be eliminated, thus reducing the number of variables in the set of potentials.

Fig. 8, in which the different $\mathfrak{c}_i$ represent instances of different classes, shows how a bottom-up elimination proceeds. Assume the target variable is located in $\mathfrak{c}_1$. Since $\mathfrak{c}_6$ is a leaf instance, it can be eliminated. Similarly, instance $\mathfrak{c}_7$, which is also a leaf, can be eliminated. Fig. 8(b) shows the remaining instances after these two eliminations. The recursive elimination goes on with $\mathfrak{c}_4$, which is now a leaf and can be eliminated (Fig. 8(c)), and proceeds until all instances except $\mathfrak{c}_1$ have been eliminated. Now, the *a posteriori* probability of the target variable can be extracted from this instance since, being an instance, it contains a *grounded* BN (fragment). Note that if $\mathfrak{c}_6$ and $\mathfrak{c}_7$ are two instances of a same class, then computations performed to eliminate $\mathfrak{c}_7$ are the same as those for eliminating $\mathfrak{c}_6$. Hence, it is possible to reuse the latter for $\mathfrak{c}_7$'s elimination: those computations can be performed at class level and are said to be *lifted*.

SVE is formalized in Algorithm 2. Note that we did not show the most general version of SVE: we assumed that the system is a DAG, that is, it contains no directed cycles. In the most general version of PRMs, there exist situations in which cycles in the system are allowed as they do not induce incoherent underlying probability distributions. This is the case, for instance, for the system of Fig. 9. To deal with such cases, more complex versions of SVE do exist (Pfeffer, 2000). However, we chose not to describe them as they are more complicated and require additional treatment to distinguish the situations in which cycles are allowed from those in which they are forbidden (as they may induce incoherent probability distributions).



(a) Step 1. (b) Step 2. (c) Step 3.

Fig. 8 Illustration of SVE's Bottom-Up Elimination.

4 A new algorithm combining SVE and Local Conditioning

4.1 A generic combining algorithm

Algorithm 2 is most of the time very efficient for computing marginal *a posteriori* probabilities. However, it has two major drawbacks: i) when there are numerous evidence, computations are essentially performed at instance level (the "if" part of the algorithm) and, thus, SVE's efficiency is quite limited; ii) when the system contains numerous loops (i.e., undirected cycles), the bottom-up elimination scheme creates multiple fill-ins that can result in inefficient computations. To illustrate this point, consider the application of Algorithm 2 to compute the probability that computer `C3` can print in the example of Fig. 5. Computer `C1`'s instance has no child and no evidence, so it can be eliminated at class level. This results in a potential, say ψ_1(`Printer.state`,`Room.circuit1`) added to $\mathbf{T}$ and another potential ϕ_1(`P1.state`,`R1.circuit1`) $= \psi_1$ added to $\mathbf{S}$. In graphical terms, the system of Fig. 10.a is substituted by that of Fig. 10.b. Note that potential ϕ_1 induced a fill-in between nodes `P1.state` and `R1.circuit1`. Eliminating Computer `C2`'s instance amounts to get ψ_1 in $\mathbf{T}$ and add a new potential ϕ_2(`P2.state`,`R1.circuit1`) $= \psi_1$ to $\mathbf{S}$, thus creating new fill-in (`P2.state`,`R1.circuit1`) as shown in Fig. 10.c. Now Printers `P1` and `P2`'s instances can be removed as they have no child. Here, class-level computations do nothing since variable `state` is an output node. For `P1`, ϕ_1 is multiplied by potential P(`P1.state`|`R1.circuit2`) before variable `P1.state` is eliminated. This produces a new potential ϕ_3(`R1.circuit1`,`R1.circuit2`) and a new fill-in (`R1.circuit1`,`R1.circuit2`) as shown Fig. 10.d. Similarly, printer `P2`'s elimination adds a new potential ϕ_4(`R1.circuit1`,`R1.circuit2`) to $\mathbf{S}$ and the same fill-in. And so on.

As can be seen, SVE's bottom-up elimination scheme is quite similar to VE in terms of the fill-ins added: in our example, substitute each instance by a single node and

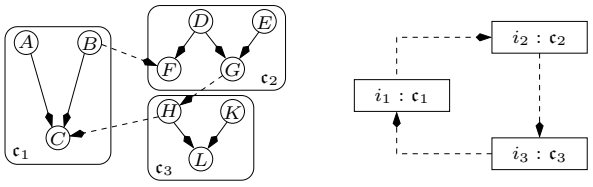


Fig. 9 A system with a directed cycle.

Input: a set of classes $\mathbf{C}$, a set of instances $\mathbf{I}$, a set of evidence $e_{\mathbf{Z}}$, a target node X_i
Output: $P(X_i, e_{\mathbf{Z}})$
let $\mathbf{S}$ and $\mathbf{T}$ be empty sets of potentials
while $\mathbf{I} \neq \emptyset$ **do**
  let $i \in \mathbf{I}$ be an instance without any child in the system
  let $\mathbf{e}$ be the set of evidence on the nodes of $\mathbf{V}(i)$
  let $\mathbf{Q}$ be the potentials in $\mathbf{S}$ containing some variables belonging to i
  let $\mathbf{V}(\mathbf{Q})$ be the variables of the potentials in $\mathbf{Q}$ that do not belong to instance i
  if *some inner nodes of* i *contain evidence in* $\mathbf{e}$ **then**
    let $\mathbf{P}$ be the set of CPTs/aggregators of the nodes in $\mathbf{V}(i)$
    $\phi \leftarrow$ VariableElimination $(\mathbf{V}(i) \cup \mathbf{V}(\mathbf{Q}), \mathbf{P} \cup \mathbf{Q}, \mathbf{e}, \mathbf{R}(i) \cup \mathbf{V}(\mathbf{Q}) \cup \{X_i\})$
  else
    let $\mathfrak{c}_i$ be the class of i
    let $\mathbf{P}$ be the set of CPTs/aggregators of the nodes in $\mathbf{V}(\mathfrak{c}_i)$
    let $\mathbf{O}$ be the set of output nodes of $\mathbf{V}(\mathfrak{c}_i)$ (i.e., referenced by other classes)
    if *class* $\mathfrak{c}_i$ *has not been visited yet* **then**
      $\psi \leftarrow$ VariableElimination $(\mathbf{V}(\mathfrak{c}_i), \mathbf{P}, \emptyset, \mathbf{R}(\mathfrak{c}_i) \cup \{X_i\} \cup \mathbf{O})$
      add ψ as the potential of class $\mathfrak{c}_i$ to set $\mathbf{T}$
      mark class $\mathfrak{c}_i$ as visited
    end
    let ψ be the potential of class $\mathfrak{c}_i$ in set $\mathbf{T}$
    $\phi \leftarrow$ VariableElimination $(\mathbf{R}(i) \cup \mathbf{O} \cup \mathbf{V}(\mathbf{Q}), \{\psi\} \cup \mathbf{Q}, \mathbf{e}, \mathbf{R}(i) \cup \mathbf{V}(\mathbf{Q}) \cup \{X_i\})$
  end
  remove i from $\mathbf{I}$
  $\mathbf{S} \leftarrow (\mathbf{S} \backslash \mathbf{Q}) \cup \{\phi\}$
end
return *table* $\prod_{\phi \in \mathbf{S}} \phi$

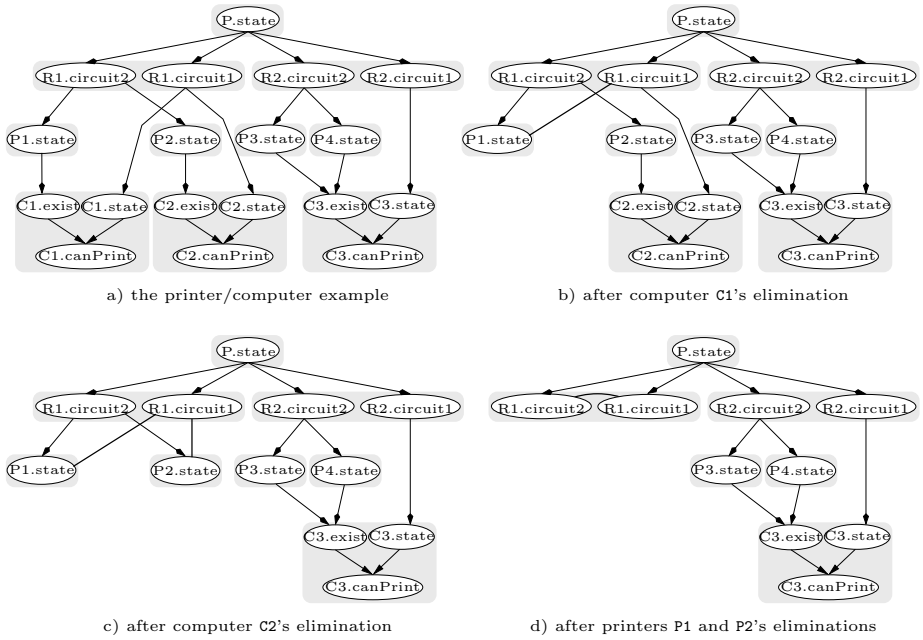
Algorithm 2: Structured Variable Elimination for computing $P(X_i, e_{\mathbf{Z}})$.

each reference by an arc, then the application of a bottom-up elimination scheme on the resulting graph (that of Fig. 11.a) produces the fill-ins of Fig. 11.b. Now assume that, before performing the bottom-up elimination, we apply Local Conditioning by cutting node `P1State`. Then, as shown in Subsection 2.2, this amounts to add variable `P1State` to the CPTs of all the other nodes and removing arc (`P1State`,`C1`). Then, applying the bottom-up elimination produces the fill-ins indicated in Fig. 11.c. It is easily seen that whenever variable `circuit2` has a bigger domain size than that of `P1State`, the computations with Local Conditioning outperform those of SVE.

Let us now see how LC can be introduced into the PRM framework. The principle of LC consists of cutting loops in BNs by removing some arcs and enforcing the property that there remains exactly one (undirected) path joining their extremities. Actually, we can generalize LC by dropping the constraint that there remains precisely one path and only require that the extremities of the removed arcs be still connected by *at least* one path. In this case, LC's arc labeling rule can be extended as follows:

Rule 4 (Extended Arc labeling) *Let* $(\mathbf{V}, \mathbf{A}, \mathbf{P})$ *be a BN. Let* $\mathbf{E} \subseteq \mathbf{A}$ *be the set of arcs removed by local conditioning. Assume that this set keeps the connectedness of the graph. For any removed arc* $(X_s, X_t) \in \mathbf{E}$*, add label* X_s *to all the arcs of all the undirected paths linking* X_s *and* X_t*.*

Proposition 2 (Extended Local Conditioning) *Let* $(\mathbf{V}, \mathbf{A}, \mathbf{P})$ *be a BN and let* $\mathbf{E} \subseteq \mathbf{A}$ *be a set of arcs satisfying Rule 4. Denote by* $\mathbf{L}_i$ *the union of the labels of the arcs in* $\mathbf{A}$ *whose heads are* X_i *(if no such arc exists, then* $\mathbf{L}_i = \emptyset$*). For any* $P(X_i|\mathbf{Pa}_i) \in \mathbf{P}$*, let* $f(X_i, \mathbf{Pa}_i, \mathbf{L}_i)$ *denote* $P(X_i|\mathbf{Pa}_i)$ *if* $\mathbf{L}_i = \emptyset$ *else the potential obtained from* $P(X_i|\mathbf{Pa}_i)$ *by adding to it dimensions* $\mathbf{L}_i$*, i.e., by duplicating it* $|\mathbf{L}_i \backslash \mathbf{Pa}_i|$ *times. Let*



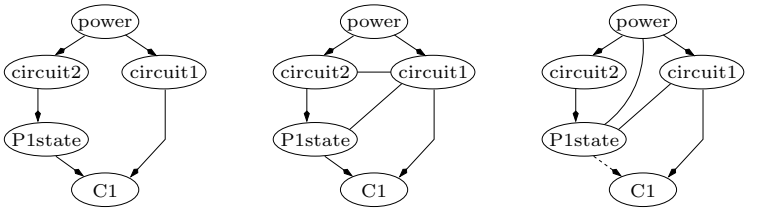
a) the printer/computer example

b) after computer C1's elimination

c) after computer C2's elimination

d) after printers P1 and P2's eliminations

Fig. 10 Computing the probability that computer C3 can print.



a) the PRM substitution

b) SVE's fill-ins

c) SVE with local conditioning

Fig. 11 SVE's Fill-ins and those resulting from Local Conditioning.

$\mathbf{P'} = \{f(X_i, \mathbf{Pa}_i, \mathbf{L}_i) : X_i \in \mathbf{V}\}$. *Finally, let* σ *be an elimination ordering satisfying Rule 3. Then applying* VE$(\mathbf{V}, \mathbf{P'}, e_{\mathbf{Z}}, \mathbf{X})$ *(Algorithm 1) eliminating nodes according to ordering* σ *results in a correct computation of* $P(\mathbf{X}, e_{\mathbf{Z}})$.

Proof of Proposition 2: Function $f(X_i, \mathbf{Pa}_i, \mathbf{L}_i)$ is either an original potential $P(X_i|\mathbf{Pa}_i)$ or potential $P(X_i|\mathbf{Pa}_i)$ with additional dimensions that are nodes of the Bayesian network. As a consequence, $\prod_{f \in \mathbf{P'}} f = \prod_{p \in \mathbf{P}} p$ since these products are tensorial. In other words, $\mathbf{P}$ and $\mathbf{P'}$ represent the same probability distributions. Consequently, applying VE with $\mathbf{P'}$ produces the same result as applying it with $\mathbf{P}$. In VE, any elimination ordering produces the same result (although some orders require more computations). Hence applying VE$(\mathbf{V}, \mathbf{P'}, e_{\mathbf{Z}}, \mathbf{X})$ with elimination ordering σ results in a correct computation of $P(\mathbf{X}, e_{\mathbf{Z}})$. □

Proposition 2 is attractive because it allows performing only *partial local conditioning*, that is, some cycles in a BN can be dealt with using Local Conditioning while

others can be dealt with using Variable Elimination. This will be useful for our combination of SVE and LC. Now, the important feature for applying this extended version of LC, which we may call ELC, is that for each removed arc (X_s, X_t) there still remains at least one path in the BN linking X_s and X_t. In terms of PRMs, this is equivalent to state that, whenever some arc (X_s, X_t) is removed from the grounded network, there should remain one path linking X_s and X_t in this network. Of course, it is never desirable to work directly with the grounded network since this one does not contain the knowledge about inner nodes used for lifting inference. In addition, this graph is much larger than both the Class Dependency Graph (CDG, see Fig. 5) and the Relational Skeleton. So, to combine SVE with ELC, it would seem better to work directly with the relational skeleton. However, this is not a good idea because it is possible to cut some arcs in this graph keeping the relational skeleton connected while disconnecting the grounded network. Actually, consider the CDG and its relational skeleton defined in Fig. 12. Clearly, removing arc $(c_1 : C_1, c_3 : C_3)$ in the relational skeleton keeps it connected. But, as seen in the grounded Network on the right of Fig. 12, this would disconnect node B from D and C. Thus, performing ELC in such a graph would not produce a correct result. So this calls for yet another graph, called a *Connected Instance Graph*, that is roughly as compact as the relational skeleton but that also contains information about the connected parts of grounded BN. This graph relies on the notion of a *Connected class*:

Definition 7 (Connected Class) A class $\mathfrak{C}$ is said to be connected if and only if its DAG is connected, that is, for any pair of attributes $(X, Y) \in \mathbf{V}(\mathfrak{C})$, there exists a path linking X and Y containing only attributes of $\mathfrak{C}$.

Any class $\mathfrak{C}$ can be mapped into a set of connected classes $\{\mathfrak{C}_1, \ldots, \mathfrak{C}_k\}$: it is sufficient to create a new class $\mathfrak{C}_i$ for each connected component of the DAG of $\mathfrak{C}$. The DAG of $\mathfrak{C}_i$ is this connected component, $\mathbf{V}(\mathfrak{C}_i)$ is the set of attributes of the connected component, set $\mathbf{R}(\mathfrak{C}_i)$ is the subset of references of $\mathbf{R}(\mathfrak{C})$ used by the nodes in $\mathbf{V}(\mathfrak{C}_i)$, and the conditional probability tables of the class are those of $\mathfrak{C}$ corresponding to the attributes in $\mathbf{V}(\mathfrak{C}_i)$. Of course, the relational skeleton can be mapped accordingly, that is, each instance of class $\mathfrak{C}$ can be substituted by one instance of each class $\mathfrak{C}_i$, $i = 1, \ldots, k$, and the references are mapped accordingly. We call the result (both the CDG of the connected classes and its relational skeleton) a *Connected PRM*.

Definition 8 (Connected Instance Graph) Let $\mathbf{G}$ be a PRM and $\mathbf{G}'$ be its corresponding *connected PRM*. The *Connected Instance Graph* $\mathbf{G}''$ of PRM $\mathbf{G}$ is a directed graph whose nodes are the instances of the *connected PRM* $\mathbf{G}'$ and whose arcs represent the references between these instances. In other words, there exists an arc

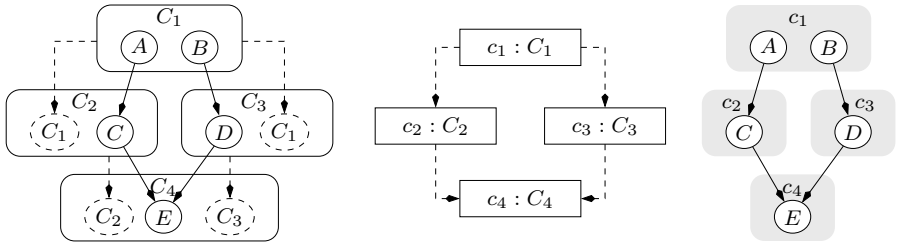


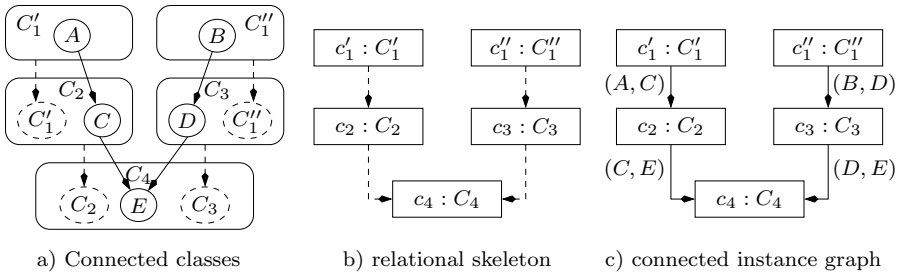
Fig. 12 Applying ELC in the relational skeleton.

$(X_s, X_t) \in \mathbf{G''}$ if and only if instance X_t has at least one reference whose range is X_s. In $\mathbf{G''}$, each arc is labeled by the set of references it represents. More precisely, each label is a set of pairs (A, B) meaning that there exists an arc $(X_s.A, X_t.B)$ in the grounded Bayes Net.

For instance, the PRM of Fig. 12 can be converted into the connected PRM of Fig. 13.a and the relational skeleton can be converted into that of Fig. 13.b. Finally, Fig. 13.c shows the connected instance graph. Using this graph, it is possible to characterize the sets of arcs that can be removed while preserving the connectedness of the grounded Bayes Net:

Proposition 3 *Let $\mathbf{E} = \{(X_{s_i}, X_{t_i}), i = 1, \ldots, k\}$ be a set of arcs of the Connected Instance Graph (CIG) such that, after their removal, there still remains at least one path in the CIG linking each pair of instances (X_{s_i}, X_{t_i}). Let $\mathbf{L}_i$ denote the union of the labels of the CIG's arcs $(X_{s_i}, X_{t_i}) \in \mathbf{E}$. Then, after removing all the arcs of $\cup_{i=1}^{k} \mathbf{L}_i$ from the grounded BN, there still remains at least one path in the grounded BN linking their extremities.*

Proof of Proposition 3: Let $\mathbf{G}$ denote the original grounded BN and $\mathbf{G'}$ that resulting from the removal of the arcs of $\cup_{i=1}^{k} \mathbf{L}_i$. Let (A, B) be an arc belonging to $\mathbf{G}$ but not to $\mathbf{G'}$, i.e., $(A, B) \in \cup_{i=1}^{k} \mathbf{L}_i$. Let X_{s_i} and X_{t_i} denote the instances in the CIG containing A and B respectively. By definition, there remains in the CIG a path, say $X_{p_1} = X_{s_i}, \ldots, X_{p_r} = X_{t_i}$, linking instances X_{s_i} and X_{t_i}. Consequently, for each pair of instances $(X_{p_j}, X_{p_{j+1}})$, there remain some arcs in $\mathbf{G'}$ linking some node(s) of X_{p_j} and some node(s) of $X_{p_{j+1}}$. Moreover, by definition of the connected classes, each of these instances corresponds to a connected subgraph of grounded BN $\mathbf{G'}$ (remember that the arcs internal to the instances do not belong to $\cup_{i=1}^{k} \mathbf{L}_i$). As a consequence, there remains at least one path in $\mathbf{G'}$ linking A and B. □



a) Connected classes b) relational skeleton c) connected instance graph

Fig. 13 A connected PRM and its connected instance graph.

Proposition 3 opens the path for adapting Local Conditioning to PRMs: the idea is to work directly in the CIG. In this graph, cutting a set of arcs $\mathbf{E}$ as indicated in the proposition is equivalent to cutting the arcs corresponding to the labels in the grounded BN. By the proposition, the extremities of each arc cut in the grounded BN are still connected by a path in this BN. Hence, ELC can safely be applied to perform inference. The labeling rule used by ELC can now be straightforwardly adapted for CIGs:

Rule 5 (Arc labeling in CIGs) *Let* $\mathbf{E} = \{(X_{s_i}, X_{t_i}), i = 1, \ldots, k\}$ *be a set of arcs removed from the CIG but yet satisfying the conditions of Proposition 3. Let* $\mathbf{L}_i$ *denote the set of labels of each arc* (X_{s_i}, X_{t_i}) *in the CIG. For any removed arc* $(X_{s_i}, X_{t_i}) \in \mathbf{E}$*, add label* $\mathbf{L}_i$ *to all the arcs of all the (undirected) paths linking* X_{s_i} *and* X_{t_i}*.*

By Rule 5, the set of arcs actually cut in the grounded BN are added as labels to all the arcs in the CIG that link the pairs of nodes in $\mathbf{E}$. Thus, after removing $\mathbf{E}$ from the CIG and updating the labels of the CIG's arcs accordingly, those labels precisely indicate the nodes that should not be eliminated from the potentials by SVE when eliminating instances: indeed, prior to removing $\mathbf{E}$, a pair (A, B) in the label of a given arc (X_s, X_t) indicates that there exists in the grounded BN an arc $(X_s.A, X_t.B)$. Hence, when eliminating instance X_t, SVE would not eliminate attribute $X_s.A$ from the potentials (else the computations would be incorrect). On the other hand, if pair (A, B) has been added by Rule 5 due to $\mathbf{E}$'s removals, then there existed an arc $(X_u, X_v) \in \mathbf{E}$ such that arc $(X_u.A, X_v.B)$ belonged to the grounded BN of the original PRM. As this arc has been removed from the grounded BN due to $\mathbf{E}$'s removals, ELC enforces that attribute $X_u.A$ shall not be eliminated when eliminating the potentials of instance X_t. Overall, if $\mathbf{L} = \{(A_j, B_j), j = 1, \ldots, r\}$ is the label of arc (X_s, X_t), then, when eliminating instance X_t, SVE shall never eliminate the set of attributes $\{A_j : j = 1, \ldots, r\}$. One simple way to enforce this is to add $\mathbf{L}' = \{A_j : j = 1, \ldots, r\}$ to the set of references of instance X_t (since SVE will never eliminate the attributes referenced by the instance it currently eliminates).

So far, the principle for combining ELC and SVE is identical to that of LC. However, in the case of PRMs, an additional rule must be applied due to that fact that SVE eliminates all the nodes of a given instance except its references. Thus, if we were only to add $\mathbf{L}'$ as references to instance X_t but not to X_s, it could be the case that eliminating X_s would marginalize-out variables in $\mathbf{L}'$ even though there still remain other instances with references to $\mathbf{L}'$, hence resulting in an incorrect computation. For instance, in the CIG of Fig. 14, where arc (c_4, c_1) has been cut, label (A, B) has been added to all the other arcs of the CIG. Adding A as reference only to the heads of these arcs amounts to change the references of c_1, c_2 and c_4. Now, if SVE is applied with elimination sequence c_1, c_2, c_3, c_4 and c_5, then c_1's elimination produces a potential $\phi_1(G, A)$. The elimination of c_2 produces a potential $\phi_2(A, C, F)$ (remember that A is only a reference in c_2, hence A is not considered as belonging to c_2 and, consequently, eliminating c_2 does not require computing the product of $P(E|C, F, A)$ by $\phi_1(G, A)$). Now, c_3's elimination amounts to perform $\phi_3(C) = \sum_{A,F} P(F)\phi_2(A, C, F) \sum_G P(G|F)\phi_1(G, A)$. As A

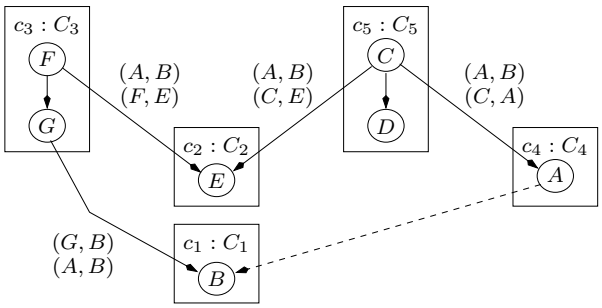


Fig. 14 References to be added to the CIG.

is not a reference in c_3, it has been eliminated. But this leads to an incorrect computation because there still remains references to A in c_4 and c_5, and marginalizing-out several times the same variable is not equivalent to marginalizing it out only once. This suggests a simple way to fix this problem: whenever an arc (X_s, X_t) is cut by ELC from the CIG, its set of labels, say $\mathbf{L}'$, should be added to both extremities of the arcs on the remaining paths between X_s and X_t except the last node of all these paths eliminated by SVE. Thus, only the last eliminated instance of all these paths can eliminate the attributes of $\mathbf{L}'$. To enforce this elimination, we must thus indicate that this instance contains all the attributes in $\mathbf{L}'$. In addition, for efficiency reasons, we shall also prevent SVE to combine potentials of variables $\mathbf{L}'$ within the other instances of the paths (i.e., to add such potentials in set $\mathbf{Q}$ of Algorithm 2) because these premature combinations increase the inference computational burden. For instance, in Fig. 14, if instances are eliminated w.r.t. order c_1, c_2, c_4, c_3, c_5, then c_1 and c_2's eliminations produce potentials $\phi_1(G, A)$ and $\phi_2(A, C, F)$ respectively. Now, SVE's elimination of c_4 shall amount to compute combination $P(A|C) \times \phi_1(G, A) \times \phi_2(A, C, F)$, hence resulting in a potential $\phi_4(A, C, F, G)$, which has a higher dimension than what can be achieved by VE. Actually, combinations are necessary only when attributes/variables are eliminated (see Lazy Propagation on this matter (Madsen and Jensen, 1999)). As A is not eliminated because it is also a reference in c_4, the combination can be postponed until instance c_5, the last one, is eliminated. The whole process can now be formalized in Algorithm 3.

Input: a set of classes $\mathbf{C}$, a set of instances $\mathbf{I}$, a set of evidence $e_{\mathbf{Z}}$, a target node X_i
Output: $P(X_i, e_{\mathbf{Z}})$
compute the CIG $\mathbf{G}$ of the PRM
select a set of arcs $\mathbf{E}$ in CIG $\mathbf{G}$ that satisfy the conditions of Proposition 3
remove these arcs from $\mathbf{G}$ and update the labels of the arcs according to Rule 5
compute an instance elimination order σ that will be used by SVE
foreach *arc* $(X_s, X_t) \in \mathbf{E}$ **do**
  foreach *node* X_r *on all the paths linking* X_s *and* X_t *in* $\mathbf{G}$ **do**
    if X_r *is not the last eliminated node (w.r.t.* σ*) on these paths* **then**
      add the tails of the arcs of (X_s, X_t)'s label as references to X_r
      and consider the attributes referenced as not belonging to instance X_r
    else
      add the tails of the arcs of (X_s, X_t)'s label as new attributes to X_r
    end
  end
end
return SVE$(\mathbf{C}, \mathbf{I}, e_{\mathbf{Z}}, X_i)$ *using elimination ordering* σ

Algorithm 3: Combining SVE with ELC for computing $P(X_i, e_{\mathbf{Z}})$.

Proposition 4 *The result of Algorithm 3 is mathematically sound.*

Proof of Proposition 4: Let $\cup_{i=1}^{k} \mathbf{L}_i$ be the set of arcs as described in Proposition 3 and let $\mathbf{G}'$ denote the grounded BN resulting from their removals. The set $\mathbf{E}$ of arcs chosen to be cut in the CIG satisfies Proposition 3. Hence, for each arc $(A, B) \in \cup_{i=1}^{k} \mathbf{L}_i$ cut in grounded BN $\mathbf{G}'$, there remains a path linking A and B in $\mathbf{G}'$. Let X_s and X_t be the instances containing A and B in the CIG respectively. By Rule 5, label (A, B) is added to all the paths in the CIG linking X_s and X_t and, in particular, A is added as a reference to all the nodes of these paths except the last one that will be eliminated

according to ordering σ. Now let $C_1 = A, \ldots, C_r = B$ be any path in $\mathbf{G}'$ linking A and B. If each arc on this path were labeled by A, then by adding dimension A to all the conditional probability tables of this path, ELC would produce a correct result (as shown in Proposition 2). Actually, a close look at the proof of Proposition 2 shows that adding dimension A is not strictly speaking compulsory: what is important is never to remove dimension A from the tables before attribute A is actually eliminated by VE (and A is always the last attribute eliminated in path $C_1 = A, \ldots, C_r = B$). But path $C_1 = A, \ldots, C_r = B$ corresponds to a path, say $D_1 = X_s, \ldots, D_k = X_t$, in the CIG. Moreover, by SVE, each time an instance is eliminated, the attributes it references are not eliminated. As a consequence, by adding A to the set of references of instances D_i (except the last one), we guarantee that A cannot be eliminated from probability tables until the last instance containing A is eliminated. This guarantees that the computations performed by SVE actually follow the conditions of Proposition 2 and, as a consequence, the correctness of the computation. □

Algorithm 3 provides a very generic way for combining SVE with ELC and, as such, does not preclude any elimination ordering σ compatible with the rules of SVE. However, as was shown at the beginning of this section, the speed of the computations is closely related to the ordering chosen. We shall now propose an efficiency way for determining good orderings.

4.2 Efficient elimination orderings

In usual Bayesian network inference, the best elimination orderings are determined by optimizing triangulations of moral graphs (Rose et al, 1976; Shoikhet and Geiger, 1997; Kjærulff, 1990; van den Eijkhof and Bodlaender, 2002). As such, the optimal ordering almost always differs from a bottom-up elimination. In this respect, SVE is not optimal since it enforces this bottom-up elimination. However, we shall see now how this can be overcome using Local Conditioning. For convenience, we will present the key idea on BNs rather than PRMs and, then, we will show how this can be adapted to PRMs.

We will first show how VE's computations can be organized within the very structure of a BN and illustrate this on Fig. 15. This is a variant of (Gonzales et al, 2007; Ben Naceur and Gonzales, 2004) adapted to bottom-up elimination schemes. The key idea is the following: at the beginning, all the CPTs of the BN are stored in their corresponding nodes in the BN. Whenever a random variable Y is eliminated, a node, say X, is selected among the set $\mathbf{V}_Y \subseteq \mathbf{V}$ of nodes having a potential containing this variable (note that X may differ from Y). Then all the nodes in $\mathbf{V}_Y \backslash \{X\}$ send to X their potentials. X can thus combine them all and eliminate random variable Y. As a result, X contains a new potential and all the nodes of $\mathbf{V}_Y \backslash \{X\}$ store no more potentials. The process is iterated until all the variables that need be eliminated have actually been eliminated. The arcs of the BN are used to send potentials from one node to another. Thus, when a node of $\mathbf{V}_Y \backslash \{X\}$ needs sending its potential to X but there exists no arc between this node and X in the BN, we add this arc to the BN. By this process, whenever a node has sent its potential to another one, it will never receive nor send any new potentials since it does not contain a potential anymore. Among all the nodes of $\mathbf{V}_Y$, the very node X that receives all the potentials is chosen so that no other node in $\mathbf{V}_Y$ is its ancestor. This rule will enable us to derive subsequently a bottom-up elimination rule.

Let us illustrate the process on the BN of Fig. 15.a. Assume that we perform VE with elimination ordering $H, I, E, D, C, B, A, F, G$. For the moment, as shown on Fig. 15.a, each node of the BN stores its own conditional probability table. Eliminating H and I amounts to substitute potentials $P(H|E,F)$ and $P(I|F,G)$ by $\mathbb{1}_{EF} = \sum_H P(H|E,F)$ and $\mathbb{1}_{FG} = \sum_I P(I|F,G)$ respectively. Here, no potentials had to be sent across the BN and, now, H and I store $\mathbb{1}_{EF}$ and $\mathbb{1}_{FG}$ respectively (see Fig. 15.b). Eliminating E requires computing $\mathbb{1}_{BF} = \sum_E P(E|B) \times \mathbb{1}_{EF}$ and, consequently, one of those two potentials need be sent over arc (E,H). By our rule, H must send its potential to E (see Fig. 15.c). Note that, from now on, H cannot receive any more potential since it does not contain one anymore, hence arc (F,H) will never be used during the computation and can thus be considered as cut (the dashed arc in Fig. 15.c). Similarly, eliminating D and C amounts to compute $P(F|A) = \sum_C P(C|A)P(F|C)$ and $P(G|A) = \sum_D P(D|A)P(G|D)$, and to send potentials from F to C and from G to D respectively. As F and G have sent their potentials, no potential will ever pass through arcs (F,I) and (G,I) (dashed arcs in Fig. 15.c). Shaded nodes indicate those nodes that have no more potentials and, thus, that will not be used anymore for the remaining inference. The elimination of B involves E sending its potential to B and the latter computing $\mathbb{1}_{AF} = \sum_B P(B|A)\mathbb{1}_{BF}$ (Fig. 15.d). Similarly, the elimination of A involves B, C and D sending their potentials to A (the highest node among A, B, C, D in the BN) and A computing $P(F,G) = \sum_A P(A)\mathbb{1}_{AF}P(F|A)P(G|A)$ (Fig. 15.e). Eliminating F now requires computing $P(G) = \sum_F P(F,G)\mathbb{1}_{FG}$ but these potentials are stored in A and I respectively and there exists no arc between A and I. We thus add arc (A,I) and I can send its potential to A (Fig. 15.f). Once this is done, I contains no more potential and only A contains a potential (i.e., $P(G)$) in the BN.

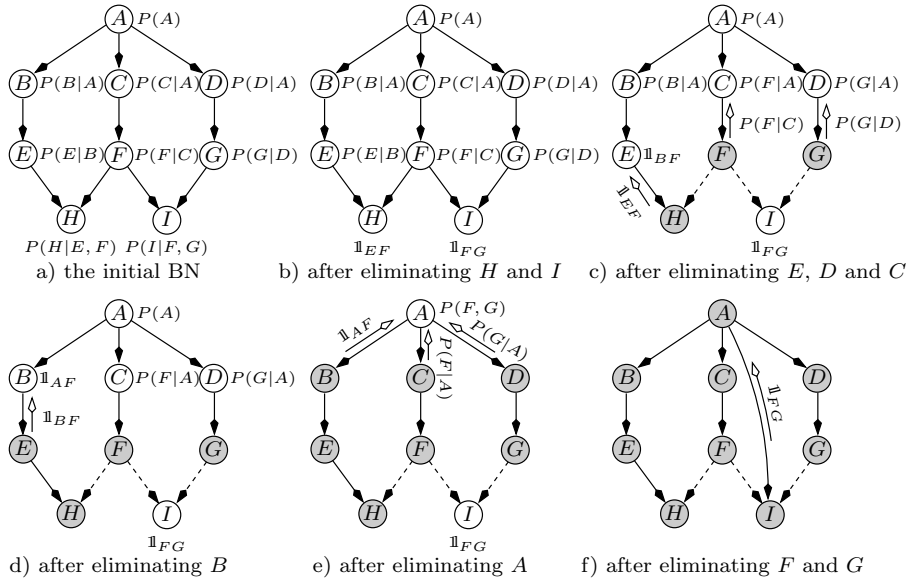
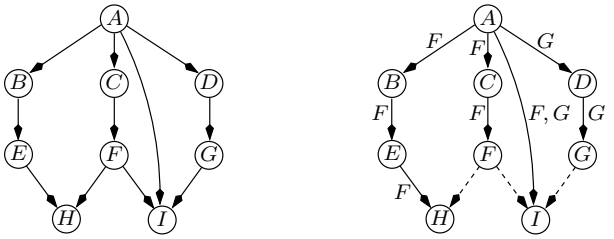


Fig. 15 Variable Elimination on a BN.



a) the BN before labeling b) the BN after labeling

Fig. 16 Arc labeling and Local Conditioning.

If we do not take into account the dashed arcs, the whole process described above has constructed a tree. This is easily proved by induction since each node sends its potential only once. The dashed arcs can thus be considered as arcs cut by Local Conditioning. Labeling the arcs of Fig. 15.f, as described by Rule 2 in Section 2, results in the labeled graph of Fig. 16.b. Now, if LC is performed in the graph of Fig. 16.b using a bottom-up elimination order, say for instance $H, I, F, C, E, B, G, D, A$, then it is easily seen that precisely the same computations as those of the preceding paragraph will be performed. This result is general and its proof is similar to that of Proposition 4 of Ben Naceur and Gonzales (2004). The reason why it works is twofold: first, by construction, the graph of Fig. 16.a does not contain any directed cycle and, as such, can be considered as a BN; second, whenever an arc is cut, say for instance (F, H), when the node at the tail of this arc (here F) was at last eliminated in the preceding paragraph, all the potentials that contained it were sent through the arcs of the BN to a given node (here A) that combined them all. Thus, there existed in the BN a path linking F and H to A and, on this path, there was no marginalizing-out of F. This precisely corresponds to the principle of LC. Finally, as shown in the following proposition, LC's computations within the graph of Fig. 16.b with a bottom-up ordering produces the same computations as SVE in the preceding paragraph.

Proposition 5 *Let* $\mathbf{G}$ *be any BN and* σ *be any elimination ordering of the variables. Let* $\mathbf{G}'$ *be the labeled BN obtained as described above and let* σ' *be a bottom-up elimination ordering. Then applying* LC *on* $\mathbf{G}'$ *produces the same result as* VE *on* $\mathbf{G}$ *with ordering* σ*, with precisely the same computations.*

Proof of Proposition 5: The proof of the correctness of the computations is the same as that of Proposition 4 of Ben Naceur and Gonzales (2004) except that, here, we use a bottom-up elimination ordering instead of a top-down order. Let us now show that computations within $\mathbf{G}'$ with ordering σ' are the same as those within $\mathbf{G}$ with ordering σ. By construction, eliminations with SVE in $\mathbf{G}$ were actually performed by sending potentials along the arcs of $\mathbf{G}'$ and whenever a node had received the messages from all of its children, it computed their combination (product) before marginalizing-out a variable. LC in $\mathbf{G}'$ does precisely the same: whenever a node, say X, is eliminated by LC, its descendants have already been eliminated (bottom-up elimination) and thus have already sent their potentials to X so that the latter can combine them. As for the marginalization, the same variable is marginalized-out by SVE in $\mathbf{G}$ and LC in $\mathbf{G}'$ because, by construction of $\mathbf{G}'$, if Y is the parent of X, then the label of arc (Y, X) contains all the variables that SVE did not eliminate. Hence, such a variable cannot be eliminated by LC. By induction, this result applies for all the nodes eliminated. □

The adaptation of the above principle to PRMs is rather straightforward: to get an efficient inference engine, moralize and triangulate the CIG (Rose et al, 1976; Shoikhet and Geiger, 1997; Kjærulff, 1990; van den Eijkhof and Bodlaender, 2002). This triangulation provides a "good" instance elimination ordering. Now, create a new labeled CIG as described in Fig. 15. This new CIG is a tree and, as shown above, a bottom-up instance elimination ordering within this tree provides the most efficient computations while satisfying the bottom-up rule of SVE.

4.3 Experimentations

To conclude this section, let us highlight the practical efficiency of our combination of SVE with Local Conditioning with some numerical experiments. In these experiments, we always used the CDG of Fig. 5. In the first set of experiments, the PRM contains 40 rooms, each one containing precisely 30 computers. Each computer is connected to two printers unless when the room contains only one printer, in which case computers are connected to this printer. In addition, each room contains the same number of printers. To illustrate the gain brought by LC, we performed inferences with the numbers of printers per room varying from 1 to 10. Fig. 17 reports the execution times (in second) with a 2.4GHz Intel Core Duo running linux and the aGrUM graphical model library[4]. Conditioning was performed on node `Printer.State`. On the left side of the figure, random variable `Printer.State` and `Room.Circuit2`'s domain sizes are respectively 2 and 6. In this case, local conditioning creates better fill-ins than SVE. This is clearly confirmed by the response times. On the right side, we increased the domain size of `Printer.State` from 2 to 3. This resulted in SVE having better fill-ins, which is again confirmed by the experiment's results. These experiments illustrate the dependence of the efficiency of Algorithm 3 on the fill-ins added. Whenever LC can select better fill-ins than SVE with its bottom-up elimination order, Algorithm 3 outperforms SVE.

To illustrate another aspect involved in the complexity of the PRM, we fixed the number of printers per room to 6, the number of rooms to 40 and we modified the number of computers per room. Here, we reverted the domain size of `Printer.State` back to 2. As we can see on Fig. 18, our combination of SVE and LC outperforms SVE alone and, the bigger the PRM, the more the combination outperforms SVE.

[4] See `http://agrum.lip6.fr`

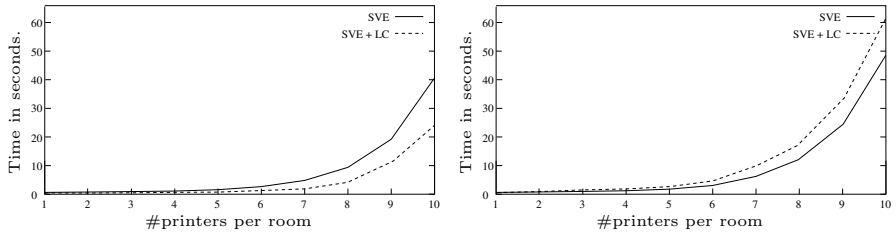


Fig. 17 Comparing SVE with the combination (SVE,LC) with different numbers of printers.

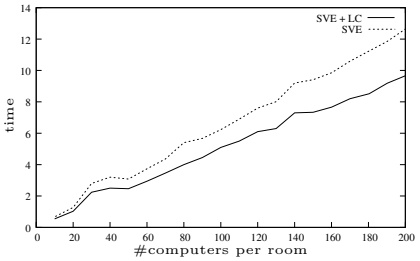


Fig. 18 Comparing SVE with the combination (SVE,LC) with different numbers of computers.

5 Conclusion

In this paper, we showed that Local Conditioning is a valuable tool for inference in the PRM framework. Its features are indeed particularly well suited for the efficient exploitation of PRM classes. This contrasts with the BN framework where Variable Elimination often outperforms Local Conditioning. In the paper, we essentially focused on the way LC could be combined harmoniously with SVE, the state-of-the-art algorithm for performing inference within PRMs. We also showed that, by selecting carefully the nodes on which to condition (or equivalently their removed outgoing arcs), the combination of LC with SVE could significantly outperform SVE alone. More importantly, we showed how this combination could free SVE from its bottom-up elimination order constraint, thus improving it to junction-tree based approaches efficiency level.

However, we think that Local Conditioning should not merely be used to select the best fill-ins to add during the computation, as we did here. Rather, it seems to have a great potential to improve inference when numerous evidence are entered into the PRM. Indeed, SVE is most efficient when it performs class-level computations. But when evidence are entered into the network, the computations in the instances they belong to are performed only at instance level, hence reducing drastically the efficiency of SVE. Now conditioning on the nodes which received evidence avoids this pitfall because it consists of reasoning separately on each possible value of these nodes before merging these separate results. As a consequence, evidence can be processed only at the final merging step and all the other computations —actually those that hold the main computational complexity— can be performed at class level. Thus, applying LC in these cases should significantly speed-up inference.

References

Allen D, Darwiche A (2003) New advances in inference by recursive conditioning. In: Proceedings of UAI, pp 2–10

Arnborg S, Corneil D, Proskurowski A (1987) Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic and Discrete Methods 8(2):277–284

Ben Naceur O, Gonzales C (2004) Une unification des algorithmes d'inférence de Pearl et de Jensen. Revue d'Intelligence Artificielle 18(2):229–260

Cowell R, Dawid A, Lauritzen S, Spiegelhalter D (2007) Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks. Information Science and Statistics, Springer

Cozman F (2000) Credal networks. Artificial Intelligence Journal 120:199–233
Dechter R (1999) Bucket elimination: A unifying framework for reasoning. Artificial Intelligence 113:41–85
Diez F (1996) Local conditioning in Bayesian networks. Artificial Intelligence 87:1–20
van den Eijkhof F, Bodlaender H (2002) Safe reduction rules for weighted treewidth. In: Proceedings of WG, Springer, LNCS, vol 2573, pp 176–185
Faÿ A, Jaffray JY (2000) A justification of local conditioning in Bayesian networks. International Journal of Approximate Reasoning 24(1):59–81
Getoor L, Taskar B (2007) Introduction to Statistical Relational Learning. MIT Press
Getoor L, Friedman N, Koller D, Pfeffer A, Taskar B (2007) Probabilistic relational models. In: Getoor L, Taskar B (eds) Introduction to Statistical Relational Learning, MIT Press, chap 5
Gonzales C, Mellouli K, Mourali O (2007) On directed and undirected propagation algorithms for Bayesian networks. In: Proceedings of ECSQARU, Lecture Notes in Artificial Intelligence, vol 4724, pp 598–610
Heckerman D (1996) A tutorial on learning with Bayesian networks. Tech. Rep. MSR-TR-95-06, Microsoft Research - Advanced Technology Division
Jaeger M (1997) Relational Bayesian networks. In: Proceedings of UAI, pp 266–273
Jensen F (1996) An introduction to Bayesian Networks. Taylor and Francis
Jensen F, Lauritzen S, Olesen K (1990) Bayesian updating in causal probabilistic networks by local computations. Computational Statistics Quarterly 4:269–282
Kjærulff U (1990) Triangulation of graphs — algorithms giving small total state space. Tech. Rep. R-90-09, Dept. of Math. and Computer Science, Aalborg University
Kjærulff U, Madsen A (2008) Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis. Information Science and Statistics, Springer
Laskey K (2008) MEBN: A language for first-order Bayesian knowledge bases. Artificial Intelligence 172(2-3):140–178
Lauritzen S (1992) Propagation of probabilities, means and variances in mixed graphical association models. Journal of the American Statistical Association 87:1098–1108
Madsen A, Jensen F (1999) LAZY propagation: A junction tree inference algorithm based on lazy inference. Artificial Intelligence 113(1–2):203–245
Mahoney S, Laskey K (1996) Network engineering for complex belief networks. In: Proceedings of UAI
Naïm P, Wuillemin PH, Leray P, Pourret O (2007) Réseaux bayésiens, 3rd edn. Eyrolles
Nilsson D (1998) An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. Statistics and Computing 8(2):159–173
Park J, Darwiche A (2003) Solving MAP exactly using systematic search. In: Proceedings of UAI, pp 459–468
Pearl J (1988) Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman
Peot M, Shachter R (1991) Fusion and propagation with multiple observations in belief networks. Artificial Intelligence 48:299–318
Pfeffer A (2000) Probabilistic reasoning for complex systems. PhD thesis, Stanford
Pfeffer A, Koller D, Milch B, Takusagawa K (1999) SPOOK: A system for probabilistic object-oriented knowledge representation. In: Proceedings of UAI, pp 541–550
Rose D, Tarjan R, Lueker G (1976) Algorithmic aspects of vertex elimination on graphs. SIAM journal on Computing 5:266–283
Shachter R, Andersen S, Szolovits P (1994) Global conditioning for probabilistic inference in belief networks. In: Proceedings of UAI

Shafer G (1996) Probabilistic expert systems. SIAM

Shenoy P (1997) Binary join trees for computing marginals in the Shenoy-Shafer architecture. International Journal of Approximate Reasoning 17(1):1–25

Shoikhet K, Geiger D (1997) Finding optimal triangulations via minimal vertex separators. In: Proceedings of AAAI

Sun X, Druzdzel M, Yuan C (2007) Dynamic weighted A^* search-based MAP algorithm for Bayesian networks. In: Proceedings of IJCAI, pp 2385–2390

Yuan C, Hansen E (2009) Efficient computation of jointree bounds for systematic MAP search. In: Proceedings of IJCAI, pp 1962–1969

Zhang N, Poole D (1994) A simple approach to Bayesian network computation. In: 10th Canadian Conference on Artificial Intelligence, pp 16–22