Handbook of Knowledge Representation
Edited by F. van Harmelen, V. Lifschitz and B. Porter

DOI: 10.1016/S1574-6526(07)03011-8

Chapter 11

Bayesian Networks

A. Darwiche

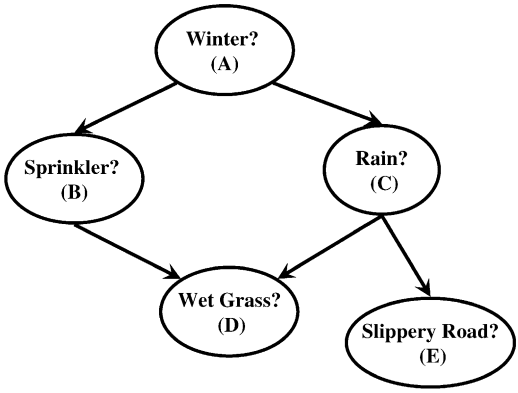
11.1 Introduction

A Bayesian network is a tool for modeling and reasoning with uncertain beliefs. A Bayesian network consists of two parts: a qualitative component in the form of a directed acyclic graph (DAG), and a quantitative component in the form conditional probabilities; see Fig. 11.1. Intuitively, the DAG of a Bayesian network explicates variables of interest (DAG nodes) and the direct influences among them (DAG edges). The conditional probabilities of a Bayesian network quantify the dependencies between variables and their parents in the DAG. Formally though, a Bayesian network is interpreted as specifying a unique probability distribution over its variables. Hence, the network can be viewed as a factored (compact) representation of an exponentially-sized probability distribution. The formal syntax and semantics of Bayesian networks will be discussed in Section 11.2.

The power of Bayesian networks as a representational tool stems both from this ability to represent large probability distributions compactly, and the availability of inference algorithms that can answer queries about these distributions without necessarily constructing them explicitly. Exact inference algorithms will be discussed in Section 11.3 and approximate inference algorithms will be discussed in Section 11.4.

Bayesian networks can be constructed in a variety of ways, depending on the application at hand and the available information. In particular, one can construct Bayesian networks using traditional knowledge engineering sessions with domain experts, by automatically synthesizing them from high level specifications, or by learning them from data. The construction of Bayesian networks will be discussed in Section 11.5.

There are two interpretations of a Bayesian network structure, a standard interpretation in terms of probabilistic independence and a stronger interpretation in terms of causality. According to the stronger interpretation, the Bayesian network specifies a family of probability distributions, each resulting from applying an intervention to the situation of interest. These causal Bayesian networks lead to additional types of queries, and require more specialized algorithms for computing them. Causal Bayesian networks will be discussed in Section 11.6.



A	Θ_A
true	0.6
false	0.4

A	B	$\Theta_{B\|A}$
true	true	0.2
true	false	0.8
false	true	0.75
false	false	0.25

A	C	$\Theta_{C\|A}$
true	true	0.8
true	false	0.2
false	true	0.1
false	false	0.9

B	C	D	$\Theta_{D\|B,C}$
true	true	true	0.95
true	true	false	0.05
true	false	true	0.9
true	false	false	0.1
false	true	true	0.8
false	true	false	0.2
false	false	true	0
false	false	false	1

C	E	$\Theta_{E\|C}$
true	true	0.7
true	false	0.3
false	true	0
false	false	1

Figure 11.1: A Bayesian network over five propositional variables. A table is associated with each node in the network, containing conditional probabilities of that node given its parents.

11.2 Syntax and Semantics of Bayesian Networks

We will discuss the syntax and semantics of Bayesian networks in this section, starting with some notational conventions.

11.2.1 Notational Conventions

We will denote variables by upper-case letters (A) and their values by lower-case letters (a). Sets of variables will be denoted by bold-face upper-case letters ($\mathbf{A}$) and their instantiations by bold-face lower-case letters ($\mathbf{a}$). For variable A and value a, we will often write a instead of $A = a$ and, hence, $\Pr(a)$ instead of $\Pr(A = a)$ for the probability of $A = a$. For a variable A with values true and false, we may use A or a to denote $A =$ true and $\neg A$ or $\overline{a}$ to denote $A =$ false. Therefore, $\Pr(A)$, $\Pr(A =$ true$)$ and $\Pr(a)$ all represent the same probability in this case. Similarly, $\Pr(\neg A)$, $\Pr(A =$ false$)$ and $\Pr(\overline{a})$ all represent the same probability.

Table 11.1. A probability distribution Pr(.) and the result of conditioning it on evidence Alarm, Pr(.|Alarm)

World	Earthquake	Burglary	Alarm	Pr(.)	Pr(.|Alarm)
ω_1	true	true	true	0.0190	0.0190/0.2442
ω_2	true	true	false	0.0010	0
ω_3	true	false	true	0.0560	0.0560/0.2442
ω_4	true	false	false	0.0240	0
ω_5	false	true	true	0.1620	0.1620/0.2442
ω_6	false	true	false	0.0180	0
ω_7	false	false	true	0.0072	0.0072/0.2442
ω_8	false	false	false	0.7128	0

11.2.2 Probabilistic Beliefs

The semantics of Bayesian networks is given in terms of probability distributions and is founded on the notion of probabilistic independence. We review both of these notions in this section.

Let $X_1, \ldots, X_n$ be a set of variables, where each variable X_i has a finite number of values x_i. Every instantiation $x_1, \ldots, x_n$ of these variables will be called a *possible world*, denoted by ω, with the set of all possible worlds denoted by Ω. A *probability distribution* Pr over variables $X_1, \ldots, X_n$ is a mapping from the set of worlds Ω induced by variables $X_1, \ldots, X_n$ into the interval [0, 1], such that $\sum_\omega \Pr(\omega) = 1$; see Table 11.1. An *event* η is a set of worlds. A probability distribution Pr assigns a probability in [0, 1] to each event η as follows: $\Pr(\eta) = \sum_{\omega \in \eta} \Pr(\omega)$.

Events are typically denoted by *propositional sentences*, which are defined inductively as follows. A sentence is either primitive, having the form $X = x$, or complex, having the form $\neg\alpha$, $\alpha \vee \beta$, $\alpha \wedge \beta$, where α and β are sentences. A propositional sentence α denotes the event $Mods(\alpha)$, defined as follows: $Mods(X = x)$ is the set of worlds in which X is set to x, $Mods(\neg\alpha) = \Omega \setminus Mods(\alpha)$, $Mods(\alpha \vee \beta) = Mods(\alpha) \cup Mods(\beta)$, and $Mods(\alpha \wedge \beta) = Mods(\alpha) \cap Mods(\beta)$. In Table 11.1, the event $\{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}$ can be denoted by the sentence Burglary $\vee$ Earthquake and has a probability of 0.28.

If some event β is observed and does not have a probability of 0 according to the current distribution Pr, the distribution is updated to a new distribution, denoted $\Pr(.|\beta)$, using *Bayes conditioning*:

$$\Pr(\alpha|\beta) = \frac{\Pr(\alpha \wedge \beta)}{\Pr(\beta)}. \tag{11.1}$$

Bayes conditioning follows from two commitments: worlds that contradict evidence β must have zero probabilities, and worlds that are consistent with β must maintain their relative probabilities.[1] Table 11.1 depicts the result of conditioning the given distribution on evidence Alarm = true, which initially has a probability of 0.2442.

When evidence β is accommodated, the belief is some event α may remain the same. We say in this case that α is independent of β. More generally, event α is inde-

[1] This is known as the principle of probability kinematics [88].

pendent of event β given event γ iff

$$\Pr(\alpha|\beta \wedge \gamma) = \Pr(\alpha|\gamma) \quad \text{or} \quad \Pr(\beta \wedge \gamma) = 0. \tag{11.2}$$

We can also generalize the definition of independence to variables. In particular, we will say that variables $\mathbf{X}$ are independent of variables $\mathbf{Y}$ given variables $\mathbf{Z}$, written $I(\mathbf{X}, \mathbf{Z}, \mathbf{Y})$, iff

$$\Pr(\mathbf{x}|\mathbf{y}, \mathbf{z}) = \Pr(\mathbf{x}|\mathbf{z}) \quad \text{or} \quad \Pr(\mathbf{y}, \mathbf{z}) = 0$$

for all instantiations $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ of variables $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$. Hence, the statement $I(\mathbf{X}, \mathbf{Z}, \mathbf{Y})$ is a compact representation of an exponential number of independence statements of the form given in (11.2).

Probabilistic independence satisfies some interesting properties known as the graphoid axioms [130], which can be summarized as follows:

$$I(\mathbf{X}, \mathbf{Z}, \mathbf{Y}) \quad \text{iff} \quad I(\mathbf{Y}, \mathbf{Z}, \mathbf{X})$$
$$I(\mathbf{X}, \mathbf{Z}, \mathbf{Y}) \;\&\; I(\mathbf{X}, \mathbf{ZW}, \mathbf{Y}) \quad \text{iff} \quad I(\mathbf{X}, \mathbf{Z}, \mathbf{YW}).$$

The first axiom is called Symmetry, and the second axiom is usually broken down into three axioms called decomposition, contraction and weak union; see [130] for details.

We will discuss the syntax and semantics of Bayesian networks next, showing the key role that independence plays in the representational power of these networks.

11.2.3 Bayesian Networks

A *Bayesian network* over variables $\mathbf{X}$ is a pair (G, Θ), where

- G is a directed acyclic graph over variables $\mathbf{X}$;
- Θ is a set of conditional probability tables (CPTs), one CPT $\Theta_{X|\mathbf{U}}$ for each variable X and its parents $\mathbf{U}$ in G. The CPT $\Theta_{X|\mathbf{U}}$ maps each instantiation $x\mathbf{u}$ to a probability $\theta_{x|\mathbf{u}}$ such that $\sum_x \theta_{x|\mathbf{u}} = 1$.

We will refer to the probability $\theta_{x|\mathbf{u}}$ as a *parameter* of the Bayesian network, and to the set of CPTs Θ as a *parametrization* of the DAG G.

A Bayesian network over variables $\mathbf{X}$ specifies a unique probability distributions over its variables, defined as follows [130]:

$$\Pr(\mathbf{x}) \stackrel{\text{def}}{=} \prod_{\theta_{x|\mathbf{u}}: x\mathbf{u} \sim \mathbf{x}} \theta_{x|\mathbf{u}}, \tag{11.3}$$

where $\sim$ represents the compatibility relationship among variable instantiations; hence, $x\mathbf{u} \sim \mathbf{x}$ means that instantiations $x\mathbf{u}$ and $\mathbf{x}$ agree on the values of their common variables. In the Bayesian network of Fig. 11.1, Eq. (11.3) gives:

$$\Pr(a, b, c, d, e) = \theta_{e|c}\theta_{d|b,c}\theta_{c|a}\theta_{b|a}\theta_a,$$

where a, b, c, d, e are values of variables A, B, C, D, E, respectively.

The distribution given by Eq. (11.3) follows from a particular interpretation of the structure and parameters of a Bayesian network (G, Θ). In particular:

- *Parameters:* Each parameter $\theta_{x|\mathbf{u}}$ is interpreted as the conditional probability of x given $\mathbf{u}$, $\Pr(x|\mathbf{u})$.
- *Structure:* Each variable X is assumed to be independent of its nondescendants $\mathbf{Z}$ given its parents $\mathbf{U}$: $I(X, \mathbf{U}, \mathbf{Z})$.[2]

The above interpretation is satisfied by a unique probability distribution, the one given in Eq. (11.3).

11.2.4 Structured Representations of CPTs

The size of a CPT $\Theta_{X|\mathbf{U}}$ in a Bayesian network is exponential in the number of parents $\mathbf{U}$. In general, if every variable can take up to d values, and has at most k parents, the size of any CPT is bounded by $O(d^{k+1})$. Moreover, if we have n network variables, the total number of Bayesian network parameters is bounded by $O(nd^{k+1})$. This number is usually quite reasonable as long as the number of parents per variable is relatively small. If number of parents $\mathbf{U}$ for variable X is large, the Bayesian network representation looses its main advantage as a compact representation of probability distributions, unless one employs a more structured representation for network parameters than CPTs.

The solutions to the problem of large CPTs fall in one of two categories. First, we may assume that the parents $\mathbf{U}$ interact with their child X according to a specific model, which allows us to specify the CPT $\Theta_{X|\mathbf{U}}$ using a smaller number of parameters (than exponential in the number of parents $\mathbf{U}$). One of the most popular examples of this approach is the *noisy-or model* of interaction and its generalizations [130, 77, 161, 51]. In its simplest form, this model assumes that variables have binary values true/false, that each parent $U \in \mathbf{U}$ being true is sufficient to make X true, except if some exception α_U materializes. By assuming that exceptions α_U are independent, one can induce the CPT $\Theta_{X|\mathbf{U}}$ using only the probabilities of these exceptions. Hence, the CPT for X can be specified using a number of parameters which is linear in the number of parents $\mathbf{U}$, instead of being exponential in the number of these parents.

The second approach for dealing with large CPTs is to appeal to nontabular representations of network parameters that exploit the *local structure* in network CPTs. In broad terms, local structure refers to the existence of nonsystematic redundancy in the probabilities appearing in a CPT. Local structure typically occurs in the form of *determinism*, where the CPT parameters take extreme values (0, 1). Another form of local structure is *context-specific independence* (*CSI*) [15], where the distribution for X can sometimes be determined by only a subset of its parents $\mathbf{U}$. Rules [136, 134] and decision trees (and graphs) [61, 80] are among the more common structured representations of CPTs.

11.2.5 Reasoning about Independence

We have seen earlier how the structure of a Bayesian network is interpreted as declaring a number of independence statements. We have also seen how probabilistic independence satisfies the graphoid axioms. When applying these axioms to the independencies declared by a Bayesian network structure, one can derive new independencies.

[2] A variable Z is a nondescendant of X if $Z \notin X\mathbf{U}$ and there is no directed path from X to Z.

In fact, any independence statement derived this way can be read off the Bayesian network structure using a graphical criterion known as *d-separation* [166, 35, 64]. In particular, we say that variables **X** are d-separated from variables **Y** by variables **Z** if every (undirected) path from a node in **X** to a node in **Y** is blocked by **Z**. A path is blocked by **Z** if it has a *sequential* or *divergent* node in **Z**, or if it has a *convergent* node that is not in **Z** nor any of its descendants are in **Z**. Whether a node $Z \in \mathbf{Z}$ is sequential, divergent, or convergent depends on the way it appears on the path: $\rightarrow Z \rightarrow$ is sequential, $\leftarrow Z \rightarrow$ is divergent, and $\rightarrow Z \leftarrow$ is convergent. There are a number of important facts about the d-separation test. First, it can be implemented in polynomial time. Second, it is sound and complete with respect to the graphoid axioms. That is, **X** and **Y** are d-separated by **Z** in DAG G if and only if the graphoid axioms can be used to show that **X** and **Y** are independent given **Z**.

There are secondary structures that one can build from a Bayesian network which can also be used to derive independence statements that hold in the distribution induced by the network. In particular, the *moral graph* G_m of a Bayesian network is an undirected graph obtained by adding an undirected edge between any two nodes that share a common child in DAG G, and then dropping the directionality of edges. If variables **X** and **Y** are separated by variables **Z** in moral graph G_m, we also have that **X** and **Y** are independent given **Z** in any distribution induced by the corresponding Bayesian network.

Another secondary structure that can be used to derive independence statements for a Bayesian network is the jointree [109]. This is a tree of clusters, where each cluster is a set of variables in the Bayesian network, with two conditions. First, every family (a node and its parents) in the Bayesian network must appear in some cluster. Second, if a variable appears in two clusters, it must also appear in every cluster on the path between them; see Fig. 11.4. Given a jointree for a Bayesian network (G, Θ), any two clusters are independent given any cluster on the path connecting them [130]. One can usually build multiple jointrees for a given Bayesian network, each revealing different types of independence information. In general, the smaller the clusters of a jointree, the more independence information it reveals. Jointrees play an important role in exact inference algorithms as we shall discuss later.

11.2.6 Dynamic Bayesian Networks

The *dynamic Bayesian network* (*DBN*) is a Bayesian network with a particular structure that deserves special attention [44, 119]. In particular, in a DBN, nodes are partitioned into *slices*, $0, 1, \ldots, t$, corresponding to different time points. Each slice has the same set of nodes and the same set of inter-slice edges, except possibly for the first slice which may have different edges. Moreover, intra-slice edges can only cross from nodes in slice t to nodes in a following slice $t + 1$. Because of their recurrent structure, DBNs are usually specified using two slices only for t and $t + 1$; see Fig. 11.2.

By restricting the structure of a DBN further at each time slice, one obtains more specialized types of networks, some of which are common enough to be studied outside the framework of Bayesian networks. Fig. 11.3 depicts one such restriction, known as a *Hidden Markov Model* [160]. Here, variables S_i typically represent unobservable states of a dynamic system, and variables O_i represent observable sensors

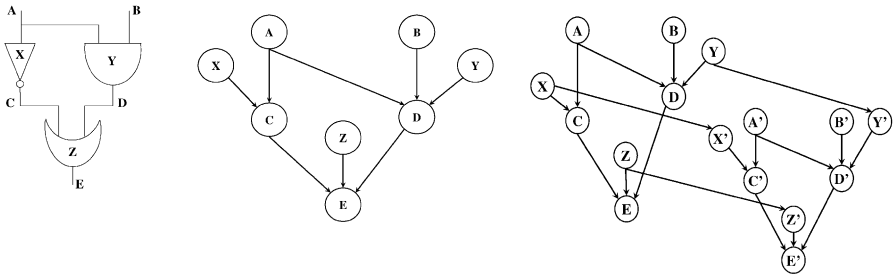


Figure 11.2: Two Bayesian network structures for a digital circuit. The one on the right is a DBN, representing the state of the circuit at two times steps. Here, variables $A, \ldots, E$ represent the state of wires in the circuit, while variables X, Y, Z represent the health of corresponding gates.

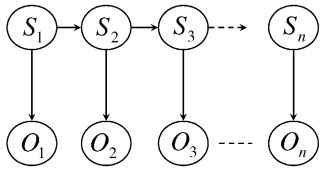


Figure 11.3: A Bayesian network structure corresponding to a Hidden Markov Model.

that may provide information on the corresponding system state. HMMs are usually studied as a special purpose model, and are equipped with three algorithms, known as the *forward–backward*, *Viterbi* and *Baum–Welch* algorithms (see [138] for a description of these algorithms and example applications of HMMs). These are all special cases of Bayesian network algorithms that we discuss in later sections.

Given the recurrent and potentially unbounded structure of DBNs (their size grows with time), they present particular challenges and also special opportunities for inference algorithms. They also admit a more refined class of queries than general Bayesian networks. Hence, it is not uncommon to use specialized inference algorithms for DBNs, instead of applying general purpose algorithms that one may use for arbitrary Bayesian networks. We will see examples of such algorithms in the following sections.

11.3 Exact Inference

Given a Bayesian (G, Θ) over variables $\mathbf{X}$, which induces a probability distribution Pr, one can pose a number of fundamental queries with respect to the distribution Pr:

- *Most Probable Explanation* (*MPE*): What's the most likely instantiation of network variables $\mathbf{X}$, given some evidence $\mathbf{e}$?

$$MPE(\mathbf{e}) = \operatorname*{argmax}_{\mathbf{x}} \Pr(\mathbf{x}, \mathbf{e}).$$

- *Probability of Evidence* (*PR*): What's the probability of evidence $\mathbf{e}$, $\Pr(\mathbf{e})$? Related to this query is *Posterior Marginals*: What's the conditional probability $\Pr(X|\mathbf{e})$ for every variable X in the network[3]?
- *Maximum a Posteriori Hypothesis* (*MAP*): What's the most likely instantiation of some network variables $\mathbf{M}$, given some evidence $\mathbf{e}$?

$$MAP(\mathbf{e}, \mathbf{M}) = \operatorname*{argmax}_{\mathbf{m}} \Pr(\mathbf{m}, \mathbf{e}).$$

These problems are all difficult. In particular, the decision version of MPE, PR, and MAP, are known to be NP-complete, PP-complete and NP^{PP}-complete, respectively [32, 158, 145, 123]. We will discuss exact algorithms for answering these queries in this section, and then discuss approximate algorithms in Section 11.4. We start in Section 11.3.1 with a class of algorithms known as *structure-based* as their complexity is only a function of the network topology. We then discuss in Section 11.3.2 refinements of these algorithms that can exploit local structure in network parameters, leading to a complexity which is both a function of network topology and parameters. Section 11.3.3 discusses a class of algorithms based on search, specialized for MAP and MPE problems. Section 11.3.4 discusses an orthogonal class of methods for *compiling* Bayesian networks, and Section 11.3.5 discusses the technique of reducing exact probabilistic reasoning to logical inference.

It should noted here that by *evidence*, we mean a variable instantiation $\mathbf{e}$ of some network variables $\mathbf{E}$. In general, one can define evidence as an arbitrary event α, yet most of the algorithms we shall discuss assume the more specific interpretation of evidence. These algorithms can be extended to handle more general notions of evidence as discussed in Section 11.3.6, which discusses a variety of additional extensions to inference algorithms.

11.3.1 Structure-Based Algorithms

When discussing inference algorithms, it is quite helpful to view the distribution induced by a Bayesian network as a product of *factors*, where a factor $f(\mathbf{X})$ is simply a mapping from instantiations $\mathbf{x}$ of variables $\mathbf{X}$ to real numbers. Hence, each CPT $\Theta_{X|\mathbf{U}}$ of a Bayesian network is a factor over variables $X\mathbf{U}$; see Fig. 11.1. The product of two factors $f(\mathbf{X})$ and $f(\mathbf{Y})$ is another factor over variables $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$: $f(\mathbf{z}) = f(\mathbf{x})f(\mathbf{y})$ where $\mathbf{z} \sim \mathbf{x}$ and $\mathbf{z} \sim \mathbf{y}$.[4] The distribution induced by a Bayesian network (G, Θ) can then be expressed as a product of its CPTs (factors) and the inference problem in Bayesian networks can then be formulated as follows. We are given a function $f(\mathbf{X})$ (i.e., probability distribution) expressed as a product of factors $f_1(\mathbf{X}_1), \ldots, f_n(\mathbf{X}_n)$ and our goal is to answer questions about the function $f(\mathbf{X})$ without necessarily computing the explicit product of these factors.

We will next describe three computational paradigms for exact inference in Bayesian networks, which share the same computational guarantees. In particular, all methods can solve the PR and MPE problems in time and space which is exponential

[3] From a complexity viewpoint, all posterior marginals can be computed using a number of PR queries that is linear in the number of network variables.

[4] Recall, that $\sim$ represents the compatibility relation among variable instantiations.

only in the network *treewidth* [8, 144]. Moreover, all can solve the MAP problem exponential only in the network *constrained treewidth* [123]. Treewidth (and constrained treewidth) are functions of the network topology, measuring the extent to which a network resembles a tree. A more formal definition will be given later.

Inference by variable elimination

The first inference paradigm we shall discuss is based on the influential concept of variable elimination [153, 181, 45]. Given a function $f(\mathbf{X})$ in factored form, $\prod_{i=1}^{n} f_i(\mathbf{X}_i)$, and some corresponding query, the method will eliminate a variable X from this function to produce another function $f'(\mathbf{X} - X)$, while ensuring that the new function is as good as the old function as far as answering the query of interest. The idea is then to keep eliminating variables one at a time, until we can extract the answer we want from the result. The key insight here is that when eliminating a variable, we will only need to multiply factors that mention the eliminated variable. The order in which variables are eliminated is therefore important as far as complexity is concerned, as it dictates the extent to which the function can be kept in factored form.

The specific method for eliminating a variable depends on the query at hand. In particular, if the goal is to solve PR, then we eliminate variables by *summing* them out. If we are solving the MPE problem, we eliminate variables by *maxing* them out. If we are solving MAP, we will have to perform both types of elimination. To sum out a variable X from factor $f(\mathbf{X})$ is to produce another factor over variables $\mathbf{Y} = \mathbf{X} - X$, denoted $\sum_X f$, where $(\sum_X f)(\mathbf{y}) = \sum_x f(\mathbf{y}, x)$. To max out variable X is similar: $(\max_X f)(\mathbf{y}) = \max_x f(\mathbf{y}, x)$. Note that summing out variables is commutative and so is maxing out variables. However, summing out and maxing out do not commute. For a Bayesian network (G, Θ) over variables $\mathbf{X}$, map variables $\mathbf{M}$, and some evidence $\mathbf{e}$, inference by variable elimination is then a process of evaluating the following expressions:

- *MPE*: $\max_{\mathbf{X}} \prod_X \Theta_{X|\mathbf{U}} \lambda_X$.
- *PR*: $\sum_{\mathbf{X}} \prod_X \Theta_{X|\mathbf{U}} \lambda_X$.
- *MAP*: $\max_{\mathbf{M}} \sum_{\mathbf{X}-\mathbf{M}} \prod_X \Theta_{X|\mathbf{U}} \lambda_X$.

Here, λ_X is a factor over variable X, called an *evidence indicator*, used to capture evidence $\mathbf{e}$: $\lambda_X(x) = 1$ if x is consistent with evidence $\mathbf{e}$ and $\lambda_X(x) = 0$ otherwise. Evaluating the above expressions leads to computing the probability of MPE, the probability of evidence, and the probability of MAP, respectively. Some extra bookkeeping allows one to recover the identity of MPE and MAP [130, 45].

As mentioned earlier, the order in which variables are eliminated is critical for the complexity of variable elimination algorithms. In fact, one can define the width of an elimination order as one smaller than the size of the largest factor constructed during the elimination process, where the size of a factor is the number of variables over which it is defined. One can then show that variable elimination has a complexity which is exponential only in the width of used elimination order. In fact, the treewidth of a Bayesian network can be defined as the width of its best elimination order. Hence, the time and space complexity of variable elimination is bounded by $O(n \exp(w))$, where n is the number of network variables (also number of initial factors), and w is

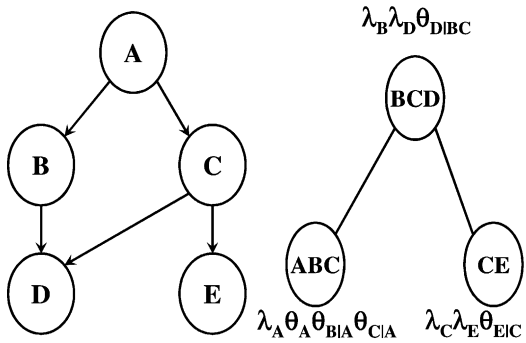


Figure 11.4: A Bayesian network (left) and a corresponding jointree (right), with the network factors and evidence indicators assigned to jointree clusters.

the width of used elimination order [45]. Note that w is lower bounded by the network treewidth. Moreover, computing an optimal elimination order and network treewidth are both known to be NP-hard [9].

Since summing out and maxing out do not commute, we must max out variables $\mathbf{M}$ last when computing MAP. This means that not all variable orders are legitimate; only those in which variables $\mathbf{M}$ come last are. The $\mathbf{M}$-constrained treewidth of a Bayesian network can then be defined as the width of its best elimination order having variables $\mathbf{M}$ last in the order. Solving MAP using variable elimination is then exponential in the constrained treewidth [123].

Inference by tree clustering

Tree clustering is another algorithm for exact inference, which is also known as the jointree algorithm [89, 105, 157]. There are different ways for deriving the jointree algorithm, one of which treats the algorithm as a refined way of applying variable elimination.

The idea is to organize the given set of factors into a tree structure, using a jointree for the given Bayesian network. Fig. 11.4 depicts a Bayesian network, a corresponding jointree, and assignment of the factors to the jointree clusters. We can then use the jointree structure to control the process of variable elimination as follows. We pick a leaf cluster $\mathbf{C}_i$ (having a single neighbor $\mathbf{C}_j$) in the jointree and then eliminate variables that appear in that cluster but in no other jointree cluster. Given the jointree properties, these variables are nothing but $\mathbf{C}_i \setminus \mathbf{C}_j$. Moreover, eliminating these variables requires that we compute the product of all factors assigned to cluster $\mathbf{C}_i$ and then eliminate $\mathbf{C}_i \setminus \mathbf{C}_j$ from the resulting factor. The result of this elimination is usually viewed as a message sent from cluster $\mathbf{C}_i$ to cluster $\mathbf{C}_j$. By the time we eliminate every cluster but one, we would have projected the factored function on the variables of that cluster (called the root). The basic insight of the jointree algorithm is that by choosing different roots, we can project the factored function on every cluster in the jointree. Moreover, some of the work we do in performing the elimination process towards one root (saved as messages) can be reused when eliminating towards another root. In fact, the amount of work that can be reused is such that we can project the function f on all clusters in the jointree with time and space bounded by $O(n \exp(w))$, where n is

the number of jointree clusters and w is the width of given jointree (size of its largest cluster minus 1). This is indeed the main advantage of the jointree algorithm over the basic variable elimination algorithm, which would need $O(n^2 \exp(w))$ time and space to obtain the same result. Interesting enough, if a network has treewidth w, then it must have a jointree whose largest cluster has size $w + 1$. In fact, every jointree for the network must have some cluster of size $\geqslant w + 1$. Hence, another definition for the treewidth of a Bayesian network is as the width of its best jointree (the one with the smallest maximum cluster).[5]

The classical description of a jointree algorithm is as follows (e.g., [83]). We first construct a jointree for the given Bayesian network; assign each network CPT $\Theta_{X|\mathbf{U}}$ to a cluster that contains $X\mathbf{U}$; and then assign each evidence indicator λ_X to a cluster that contains X. Fig. 11.4 provides an example of this process. Given evidence $\mathbf{e}$, a jointree algorithm starts by setting evidence indicators according to given evidence. A cluster is then selected as the root and message propagation proceeds in two phases, inward and outward. In the *inward phase*, messages are passed toward the root. In the *outward phase*, messages are passed away from the root. The inward phase is also known as the *collect* or *pull* phase, and the outward phase is known as the *distribute* or *push* phase. Cluster i sends a message to cluster j only when it has received messages from all its other neighbors k. A message from cluster i to cluster j is a factor M_{ij} defined as follows:

$$M_{i,j} = \sum_{\mathbf{C}_i \setminus \mathbf{C}_j} \Phi_i \prod_{k \neq j} M_{k,i},$$

where Φ_i is the product of factors and evidence indicators assigned to cluster i. Once message propagation is finished, we have the following for each cluster i in the jointree:

$$\Pr(\mathbf{C}_i, \mathbf{e}) = \Phi_i \prod_k M_{k,i}.$$

Hence, we can compute the joint marginal for any subset of variables that is included in a cluster.

The above description corresponds to a version of the jointree algorithm known as the Shenoy–Shafer architecture [157]. Another popular version of the algorithm is the Hugin architecture [89]. The two versions differ in their space and time complexity on arbitrary jointrees [106]. The jointree algorithm is quite versatile allowing even more architectures (e.g., [122]), more complex types of queries (e.g., [91, 143, 34]), including MAP and MPE, and a framework for time space tradeoffs [47].

Inference by conditioning

A third class of exact inference algorithms is based on the concept of *conditioning* [129, 130, 39, 81, 162, 152, 37, 52]. The key concept here is that if we know the value of a variable X in a Bayesian network, then we can remove edges outgoing from X, modify the CPTs for children of X, and then perform inference equivalently on the simplified network. If the value of variable X is not known, we can still exploit this idea by doing a case analysis on variable X, hence, instead of computing

[5] Jointrees correspond to tree-decompositions [144] in the graph theoretic literature.

$\Pr(\mathbf{e})$, we compute $\sum_x \Pr(\mathbf{e}, x)$. This idea of conditioning can be exploited in different ways. The first exploitation of this idea was in the context of loop-cutset conditioning [129, 130, 11]. A loop-cutset for a Bayesian network is a set of variables $\mathbf{C}$ such that removing edges outgoing from $\mathbf{C}$ will render the network a polytree: one in which we have a single (undirected) path between any two nodes. Inference on polytree networks can indeed be performed in time and space linear in their size [129]. Hence, by using the concept of conditioning, performing case analysis on a loop-cutset $\mathbf{C}$, one can reduce the query $\Pr(\mathbf{e})$ into a set of queries $\sum_{\mathbf{c}} \Pr(\mathbf{e}, \mathbf{c})$, each of which can be answered in linear time and space using the polytree algorithm.

This algorithm has linear space complexity as one needs to only save modest information across the different cases. This is a very attractive feature compared to algorithms based on elimination. The bottleneck for loop-cutset conditioning, however, is the size of cutset $\mathbf{C}$ since the time complexity of the algorithm is exponential in this set. One can indeed construct networks which have a bounded treewidth, leading to linear time complexity by elimination algorithms, yet an unbounded loop-cutset. A number of improvements have been proposed on loop-cutset conditioning (e.g., [39, 81, 162, 152, 37, 52]), yet only *recursive conditioning* [39] and its variants [10, 46] have a treewidth-based complexity similar to elimination algorithms.

The basic idea behind recursive conditioning is to identify a cutset $\mathbf{C}$ that is not necessarily a loop-cutset, but that can decompose a network $\mathcal{N}$ in two (or more) subnetworks, say, $\mathcal{N}^l_{\mathbf{c}}$ and $\mathcal{N}^r_{\mathbf{c}}$ with corresponding distributions $\Pr^l_{\mathbf{c}}$ and $\Pr^r_{\mathbf{c}}$ for each instantiation $\mathbf{c}$ of cutset $\mathbf{C}$. In this case, we can write

$$\Pr(\mathbf{e}) = \sum_{\mathbf{c}} \Pr(\mathbf{e}, \mathbf{c}) = \sum_{\mathbf{c}} \Pr^l_{\mathbf{c}}(\mathbf{e}^l, \mathbf{c}^l) \Pr^r_{\mathbf{c}}(\mathbf{e}^r, \mathbf{c}^r),$$

where $\mathbf{e}^l/\mathbf{c}^l$ and $\mathbf{e}^r/\mathbf{c}^r$ are parts of evidence/cutset pertaining to networks $\mathcal{N}^l$ and $\mathcal{N}^r$, respectively. The subqueries $\Pr^l_{\mathbf{c}}(\mathbf{e}^l, \mathbf{c}^l)$ and $\Pr^r_{\mathbf{c}}(\mathbf{e}^r, \mathbf{c}^r)$ can then be solved using the same technique, recursively, by finding cutsets for the corresponding subnetworks $\mathcal{N}^l_{\mathbf{c}}$ and $\mathcal{N}^r_{\mathbf{c}}$. This algorithm is typically driven by a structure known as a *dtree*, which is a binary tree with its leaves corresponding to the network CPTs. Each dtree provides a complete recursive decomposition over the corresponding network, with a cutset for each level of the decomposition [39].

Given a dtree where each internal node T has children T^l and T^r, and each leaf node has a CPT associated with it, recursive conditioning can then compute the probability of evidence $\mathbf{e}$ as follows:

$$rc(T, \mathbf{e}) = \begin{cases} \sum_{\mathbf{c}} rc(T^l, \mathbf{ec}) rc(T^r, \mathbf{ec}), & T \text{ is an internal node with cutset } \mathbf{C}; \\ \sum_{\mathbf{u}x \sim \mathbf{e}} \theta_{x|\mathbf{u}}, & T \text{ is a leaf node with CPT } \Theta_{X|\mathbf{U}}. \end{cases}$$

Note that similar to loop-cutset conditioning, the above algorithm also has a linear space complexity which is better than the space complexity of elimination algorithms. Moreover, if the Bayesian network has treewidth w, there is then a dtree which is both balanced and has cutsets whose sizes are bounded by $w + 1$. This means that the above algorithm can run in $O(n \exp(w \log n))$ time and $O(n)$ space. This is worse than the time complexity of elimination algorithms, due to the $\log n$ factor, where n is the number of network nodes.

A careful analysis of the above algorithm, however, reveals that it may make identical recursive calls in different parts of the recursion tree. By caching the value of a recursive call $rc(T, .)$, one can avoid evaluating the same recursive call multiple times. In fact, if a network has a treewidth w, one can always construct a dtree on which caching will reduce the running time from $O(n \exp(w \log n))$ to $O(n \exp(w))$, while bounding the space complexity by $O(n \exp(w))$, which is identical to the complexity of elimination algorithms. In principle, one can cache as many results as available memory would allow, leading to a framework for trading off time and space [3], where space complexity ranges from $O(n)$ to $O(n \exp(w))$, and time complexity ranges from $O(n \exp(w \log n))$ to $O(n \exp(w))$. Recursive conditioning can also be used to compute multiple marginals [4], in addition to MAP and MPE queries [38], within the same complexity discussed above.

We note here that the quality of a variable elimination order, a jointree and a dtree can all be measured in terms of the notion of *width*, which is lower bounded by the network treewidth. Moreover, the complexity of algorithms based on these structures are all exponential only in the width of used structure. Polynomial time algorithms exists for converting between any of these structures, while preserving the corresponding width, showing the equivalence of these methods with regards to their computational complexity in terms of treewidth [42].

11.3.2 Inference with Local (Parametric) Structure

The computational complexity bounds given for elimination, clustering and conditioning algorithms are based on the network topology, as captured by the notions of treewidth and constrained treewidth. There are two interesting aspects of these complexity bounds. First, they are independent of the particular parameters used to quantify Bayesian networks. Second, they are both best-case and worst-case bounds for the specific statements given for elimination and conditioning algorithms.

Given these results, only networks with reasonable treewidth are accessible to these structure-based algorithms. One can provide refinements of both elimination/clustering and conditioning algorithms, however, that exploit the parametric structure of a Bayesian network, allowing them to solve some networks whose treewidth can be quite large.

For elimination algorithms, the key is to adopt nontabular representations of factors as initially suggested by [182] and developed further by other works (e.g., [134, 50, 80, 120]). Recall that a factor $f(\mathbf{X})$ over variables $\mathbf{X}$ is a mapping from instantiations $\mathbf{x}$ of variables $\mathbf{X}$ to real numbers. The standard statements of elimination algorithms assume that a factor $f(\mathbf{X})$ is represented by a table that has one row of each instantiation $\mathbf{x}$. Hence, the size of factor $f(\mathbf{X})$ is always exponential in the number of variables in $\mathbf{X}$. This also dictates the complexity of factor operations, including multiplication, summation and maximization. In the presence of parametric structure, one can afford to use more structured representations of factors that need not be exponential in the variables over which they are defined. In fact, one can use any factor representation as long as they provide corresponding implementations of the factor operations of multiplication, summing out, and maxing out, which are used in the context of elimination algorithms. One of the more effective structured representations of factors is the *algebraic decision diagram* (*ADD*) [139, 80], which provides efficient implementations of these operations.

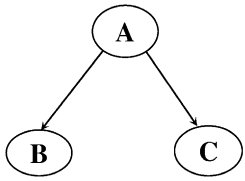
In the context of conditioning algorithms, local structure can be exploited at multiple levels. First, when considering the cases **c** of a cutset **C**, one can skip a case **c** if it is logically inconsistent with the logical constraints implied by the network parameters. This inconsistency can be detected by some efficient logic propagation techniques that run in the background of conditioning algorithms [2]. Second, one does not always need to instantiate all cutset variables before a network is disconnected or converted into a polytree, as some partial cutset instantiations may have the same effect if we have context-specific independence [15, 25]. Third, local structure in the form of equal network parameters within the same CPT will reduce the number of distinct subproblems that need to be solved by recursive conditioning, allowing caching to be much more effective [25]. Considering various experimental results reported in recent years, it appears that conditioning algorithms have been more effective in exploiting local structure, especially determinism, as compared to algorithms based on variable eliminating (and, hence, clustering).

Network preprocessing can also be quite effective in the presence of local structure, especially determinism, and is orthogonal to the algorithms used afterwards. For example, preprocessing has proven quite effective and critical for networks corresponding to genetic linkage analysis, allowing exact inference on networks with very high treewidth [2, 54, 55, 49]. A fundamental form of preprocessing is CPT decomposition, in which one decomposes a CPT with local structure (e.g., [73]) into a series of CPTs by introducing auxiliary variables [53, 167]. This decomposition can reduce the treewidth of given network, allowing inference to be performed much more efficiently. The problem of finding an optimal CPT decomposition corresponds to the problem of determining tensor rank [150], which is NP-hard [82]. Closed form solutions are known, however, for CPTs with a particular local structure [150].

11.3.3 Solving MAP and MPE by Search

MAP and MPE queries are conceptually different from PR queries as they correspond to optimization problems whose outcome is a variable instantiation instead of a probability. These queries admit a very effective class of algorithms based on branch and bound search. For MPE, the search tree includes a leaf for each instantiation **x** of nonevidence variables **X**, whose probability can be computed quite efficiently given Eq. (11.3). Hence, the key to the success of these search algorithms is the use of evaluation functions that can be applied to internal nodes in the search tree, which correspond to partial variable instantiations **i**, to upper bound the probability of any completion **x** of instantiation **i**. Using such an evaluation function, one can possibly prune part of the search space, therefore, solving MPE without necessarily examining the space of all variable instantiations. The most successful evaluation functions are based on relaxations of the variable elimination algorithm, allowing one to eliminate a variable without necessarily multiplying all factors that include the variable [95, 110]. These relaxations lead to a spectrum of evaluation functions, that can trade accuracy with efficiency.

A similar idea can be applied to solving MAP, with a notable distinction. In MAP, the search tree will be over the space of instantiations of a subset **M** of network variables. Moreover, each leaf node in the search tree will correspond to an instantiation **m** in this case. Computing the probability of a partial instantiation **m** requires a PR query



A	θ_A
true	0.5
false	0.5

A	B	$\theta_{B\|A}$
true	true	1
true	false	0
false	true	0
false	false	1

A	C	$\theta_{C\|A}$
true	true	0.8
true	false	0.2
false	true	0.2
false	false	0.8

Figure 11.5: A Bayesian network.

though, which itself can be exponential in the network treewidth. Therefore, the success of search-based algorithms for MAP depends on both the efficient evaluation of leaf nodes in the search tree, and on evaluation functions for computing upper bounds on the completion of partial variable instantiations [123, 121]. The most successful evaluation function for MAP is based on a relaxation of the variable elimination algorithm for computing MAP, allowing one to use any variable order instead of insisting on a constrained variable order [121].

11.3.4 Compiling Bayesian Networks

The probability distribution induced by a Bayesian network can be compiled into an *arithmetic circuit*, allowing various probabilistic queries to be answered in time linear in the compiled circuit size [41]. The compilation time can be amortized over many online queries, which can lead to extremely efficient online inference [25, 27]. Compiling Bayesian networks is especially effective in the presence of local structure, as the exploitation of local structure tends to incur some overhead that may not be justifiable in the context of standard algorithms when the local structure is not excessive. In the context of compilation, this overhead is incurred only once in the offline compilation phase.

To expose the semantics of this compilation process, we first observe that the probability distribution induced by a Bayesian network, as given by Eq. (11.3), can be expressed in a more general form:

$$f = \sum_{\mathbf{x}} \prod_{\lambda_x : x \sim \mathbf{x}} \lambda_x \prod_{\theta_{x|\mathbf{u}} : x\mathbf{u} \sim \mathbf{x}} \theta_{x|\mathbf{u}}, \tag{11.4}$$

where λ_x is called an evidence indicator variable (we have one indicator λ_x for each variable X and value x). This form is known as the *network polynomial* and represents the distribution as follows. Given any evidence $\mathbf{e}$, let $f(\mathbf{e})$ denotes the value of polynomial f with each indicator variable λ_x set to 1 if x is consistent with evidence $\mathbf{e}$ and set to 0 otherwise. It then follows that $f(\mathbf{e})$ is the probability of evidence $\mathbf{e}$. Following is the polynomial for the network in Fig. 11.5:

$$f = \lambda_a \lambda_b \lambda_c \theta_a \theta_{b|a} \theta_{c|a} + \lambda_a \lambda_b \lambda_{\bar{c}} \theta_a \theta_{b|a} \theta_{\bar{c}|a} + \cdots \lambda_{\bar{a}} \lambda_{\bar{b}} \lambda_{\bar{c}} \theta_{\bar{a}} \theta_{\bar{b}|\bar{a}} \theta_{\bar{c}|\bar{a}}.$$

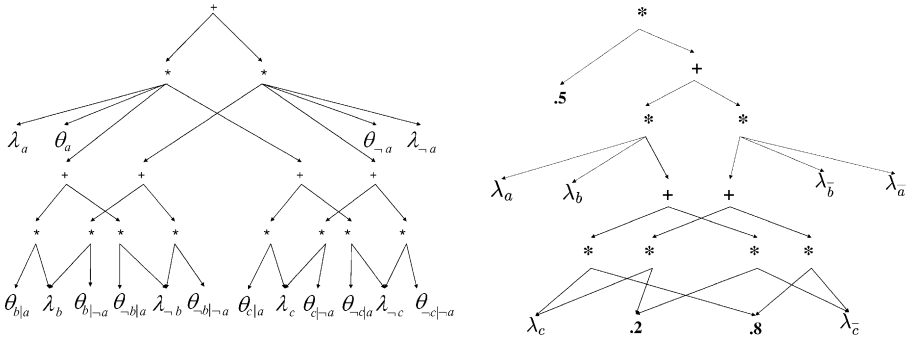


Figure 11.6: Two circuits for the Bayesian network in Fig. 11.5.

The network polynomial has an exponential number of terms, but can be factored and represented more compactly using an arithmetic circuit, which is a rooted, directed acyclic graph whose leaf nodes are labeled with evidence indicators and network parameters, and internal nodes are labeled with multiplication and addition operations. The size of an arithmetic circuit is measured by the number of edges that it contains. Fig. 11.6 depicts an arithmetic circuit for the above network polynomial. This arithmetic circuit is therefore a compilation of corresponding Bayesian network as it can be used to compute the probability of any evidence **e** by evaluating the circuit while setting the indicators to 1/0 depending on their consistency with evidence **e**. In fact, the partial derivatives of this circuit with respect to indicators λ_x and parameters $\theta_{x|\mathbf{u}}$ can all be computed in a single second pass on the circuit. Moreover, the values of these derivatives can be used to immediately answer various probabilistic queries, including the marginals over networks variables and families [41]. Hence, for a given evidence, one can compute the probability of evidence and posterior marginals on all network variables and families in two passes on the arithmetic circuit.

One can compile a Bayesian network using exact algorithms based on elimination [26] or conditioning [25], by replacing their addition and multiplication operations by corresponding operations for building the circuit. In fact, for jointree algorithms, the arithmetic circuit can be generated directly from the jointree structure [124]. One can also generate these compilations by reducing the problem to logical inference as discussed in the following section. If structure-based versions of elimination and conditioning algorithms are used to compile Bayesian networks, the size of compiled arithmetic circuits will be exponential in the network treewidth in the best case. If one uses versions that exploit parametric structure, the resulting compilation may not be lower bounded by treewidth [25, 27]. Fig. 11.6 depicts two arithmetic circuits for the same network, the one on the right taking advantage of network parameters and is therefore smaller than the one on the left, which is valid for any value of network parameters.

11.3.5 Inference by Reduction to Logic

One of the more effective approaches for exact probabilistic inference in the presence of local structure, especially determinism, is based on reducing the problem to one of

A	Θ_A
a_1	0.1
a_2	0.9

A	B	$\Theta_{B\|A}$
a_1	b_1	0.1
a_1	b_2	0.9
a_2	b_1	0.2
a_2	b_2	0.8

A	C	$\Theta_{C\|A}$
a_1	c_1	0.1
a_1	c_2	0.9
a_2	c_1	0.2
a_2	c_2	0.8

Figure 11.7: The CPTs of Bayesian network with two edges $A \rightarrow B$ and $A \rightarrow C$.

logical inference. The key technique is to encode the Bayesian network as a propositional theory in conjunctive normal form (CNF) and then apply algorithms for model counting [147] or knowledge compilation to the resulting CNF [40]. The encoding can be done in multiple ways [40, 147], yet we focus on one particular encoding [40] in this section to illustrate the reduction technique.

We will now discuss the CNF encoding for the Bayesian network in Fig. 11.7. We first define the CNF variables which are in one-to-one correspondence with evidence indicators and network parameters as defined in Section 11.3.4, but treated as propositional variables in this case. The CNF Δ is then obtained by processing network variables and CPTs, writing corresponding clauses as follows:

Variable A:	$\lambda_{a_1} \vee \lambda_{a_2}$	$\neg\lambda_{a_1} \vee \neg\lambda_{a_2}$
Variable B:	$\lambda_{b_1} \vee \lambda_{b_2}$	$\neg\lambda_{b_1} \vee \neg\lambda_{b_2}$
Variable C:	$\lambda_{c_1} \vee \lambda_{c_2}$	$\neg\lambda_{c_1} \vee \neg\lambda_{c_2}$
CPT for A:	$\lambda_{a_1} \Leftrightarrow \theta_{a_1}$	
CPT for B:	$\lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1\|a_1}$	$\lambda_{a_1} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2\|a_1}$
	$\lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1\|a_2}$	$\lambda_{a_2} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2\|a_2}$
CPT for C:	$\lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1\|a_1}$	$\lambda_{a_1} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2\|a_1}$
	$\lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1\|a_2}$	$\lambda_{a_2} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2\|a_2}$

The clauses for variables are simply asserting that exactly one evidence indicator must be true. The clauses for CPTs are establishing an equivalence between each network parameter and its corresponding indicators. This resulting CNF has two important properties. First, its size is linear in the network size. Second, its models are in one-to-one correspondence with the instantiations of network variables. Table 11.2 illustrates the variable instantiations and corresponding CNF models for the previous example.

We can now either apply a model counter to the CNF queries [147], or compile the CNF to obtain an arithmetic circuit for the Bayesian network [40]. If we want to apply a model counter to the CNF, we must first assign weights to the CNF variables (hence, we will be performing weighted model counting). All literals of the form λ_x, $\neg\lambda_x$ and $\neg\theta_{x|\mathbf{u}}$ get weight 1, while literals of the form $\theta_{x|\mathbf{u}}$ get a weight equal to the value of parameter $\theta_{x|\mathbf{u}}$ as defined by the Bayesian network; see Table 11.2. To compute the probability of any event α, all we need to do then is computed the weighted model count of $\Delta \wedge \alpha$.

This reduction of probabilistic inference to logical inference is currently the most effective technique for exploiting certain types of parametric structure, including determinism and parameter equality. It also provides a very effective framework for exploiting evidence computationally and for accommodating general types evidence [25, 24, 147, 27].

Table 11.2. Illustrating the models and corresponding weights of a CNF encoding a Bayesian network

Network instantiation	CNF model	ω_i sets these CNF vars to true and all others to false	Model weight
$a_1b_1c_1$	ω_0	$\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\theta_{a_1}\theta_{b_1\|a_1}\theta_{c_1\|a_1}$	$0.1 \cdot 0.1 \cdot 0.1 = 0.001$
$a_1b_1c_2$	ω_1	$\lambda_{a_1}\lambda_{b_1}\lambda_{c_2}\theta_{a_1}\theta_{b_1\|a_1}\theta_{c_2\|a_1}$	$0.1 \cdot 0.1 \cdot 0.9 = 0.009$
$a_1b_2c_1$	ω_2	$\lambda_{a_1}\lambda_{b_2}\lambda_{c_1}\theta_{a_1}\theta_{b_2\|a_1}\theta_{c_1\|a_1}$	$0.1 \cdot 0.9 \cdot 0.1 = 0.009$
$a_1b_2c_2$	ω_3	$\lambda_{a_1}\lambda_{b_2}\lambda_{c_2}\theta_{a_1}\theta_{b_2\|a_1}\theta_{c_2\|a_1}$	$0.1 \cdot 0.9 \cdot 0.9 = 0.081$
$a_2b_1c_1$	ω_4	$\lambda_{a_2}\lambda_{b_1}\lambda_{c_1}\theta_{a_2}\theta_{b_1\|a_1}\theta_{c_1\|a_2}$	$0.9 \cdot 0.2 \cdot 0.2 = 0.036$
$a_2b_1c_2$	ω_5	$\lambda_{a_2}\lambda_{b_1}\lambda_{c_2}\theta_{a_2}\theta_{b_1\|a_1}\theta_{c_2\|a_2}$	$0.9 \cdot 0.2 \cdot 0.8 = 0.144$
$a_2b_2c_1$	ω_6	$\lambda_{a_2}\lambda_{b_2}\lambda_{c_1}\theta_{a_2}\theta_{b_2\|a_1}\theta_{c_1\|a_2}$	$0.9 \cdot 0.8 \cdot 0.2 = 0.144$
$a_2b_2c_2$	ω_7	$\lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\theta_{a_2}\theta_{b_2\|a_1}\theta_{c_2\|a_2}$	$0.9 \cdot 0.8 \cdot 0.8 = 0.576$

11.3.6 Additional Inference Techniques

We discuss in this section some additional inference techniques which can be crucial in certain circumstances.

First, all of the methods discussed earlier are immediately applicable to DBNs. However, the specific, recurrent structure of these networks calls for some special attention. For example, PR queries can be further refined depending on the location of evidence and query variables within the network structure, leading to specialized queries, such as *monitoring*. Here, the evidence is restricted to network slices $t = 0, \ldots, t = i$ and the query variables are restricted to slice $t = i$. In such a case, and by using restricted elimination orders, one can perform inference in space which is better than linear in the network size [13, 97, 12]. This is important for DBNs as a linear space complexity can be unpractical if we have too many slices.

Second, depending on the given evidence and query variables, a network can potentially be pruned before inference is performed. In particular, one can always remove edges outgoing from evidence variables [156]. One can also remove leaf nodes in the network as long as they do not correspond to evidence or query variables [155]. This process of node removal can be repeated, possibly simplifying the network structure considerably. More sophisticated pruning techniques are also possible [107].

Third, we have so far considered only simple evidence corresponding to the instantiation $\mathbf{e}$ of some variables $\mathbf{E}$. If evidence corresponds to a general event α, we can add an auxiliary node X_α to the network, making it a child of all variables $\mathbf{U}$ appearing in α, setting the CPT $\Theta_{X_\alpha|\mathbf{U}}$ based on α, and asserting evidence on X_α [130]. A more effective solution to this problem can be achieved in the context of approaches that reduce the problem to logical inference. Here, we can simply add the event α to the encoded CNF before we apply logical inference [147, 24]. Another type of evidence we did not consider is *soft evidence*. This can be specified in two forms. We can declare that the evidence changes the probability of some variable X from $\Pr(X)$ to $\Pr'(X)$. Or we can assert that the new evidence on X changes its odds by a given factor k, known as the Bayes factor: $\mathrm{O}'(X)/\mathrm{O}(X) = k$. Both types of evidence can be handled by adding an auxiliary child $X_\mathbf{e}$ for node X, setting its CPT $\Theta_{X_\mathbf{e}|X}$ depending on the strength of soft evidence, and finally simulating the soft evidence by hard evidence on $X_\mathbf{e}$ [130, 22].

11.4 Approximate Inference

All exact inference algorithms we have discussed for PR have a complexity which is exponential in the network treewidth. Approximate inference algorithms are generally not sensitive to treewidth, however, and can be quite efficient regardless of the network topology. The issue with these methods is related to the quality of answers the compute, which for some algorithms is quite related to the amount of time budgeted by the algorithm. We discuss two major classes of approximate inference algorithms in this section. The first and more classical class is based on sampling. The second and more recent class of methods can be understood in terms of a reduction to optimization problems. We note, however, that none of these algorithms offer general guarantees on the quality of approximations they produce, which is not surprising since the problem of approximating inference to any desired precision is known to be NP-hard [36].

11.4.1 Inference by Stochastic Sampling

Sampling from a probability distribution $\Pr(\mathbf{X})$ is a process of generating complete instantiations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of variables $\mathbf{X}$. A key property of a sampling process is its *consistency:* generating samples $\mathbf{x}$ with a frequency that converges to their probability $\Pr(\mathbf{x})$ as the number of samples approaches infinity. By generating such consistent samples, one can approximate the probability of some event α, $\Pr(\alpha)$, in terms of the fractions of samples that satisfy α, $\widehat{\Pr}(\alpha)$. This approximated probability will then converge to the true probability as the number of samples reaches infinity. Hence, the precision of sampling methods will generally increase with the number of samples, where the complexity of generating a sample is linear in the size of the network, and is usually only weakly dependent on its topology.

Indeed, one can easily generate consistent samples from a distribution Pr that is induced by a Bayesian network (G, Θ), using time that is linear in the network size to generate each sample. This can be done by visiting the network nodes in topological order, parents before children, choosing a value for each node X by sampling from the distribution $\Pr(X|\mathbf{u}) = \Theta_{X|\mathbf{u}}$, where $\mathbf{u}$ is the chosen values for X's parents $\mathbf{U}$. The key question with sampling methods is therefore related to the speed of convergence (as opposed to the speed of generating samples), which is usually affected by two major factors: the query at hand (whether it has a low probability) and the specific network parameters (whether they are extreme).

Consider, for example, approximating the query $\Pr(\alpha|\mathbf{e})$ by approximating $\Pr(\alpha, \mathbf{e})$ and $\Pr(\mathbf{e})$ and then computing $\widehat{\Pr}(\alpha|\mathbf{e}) = \widehat{\Pr}(\alpha, \mathbf{e})/\widehat{\Pr}(\mathbf{e})$ according to the above sampling method, known as *logic sampling* [76]. If the evidence $\mathbf{e}$ has a low probability, the fraction of samples that satisfy $\mathbf{e}$ (and α, $\mathbf{e}$ for that matter) will be small, decreasing exponentially in the number of variables instantiated by evidence $\mathbf{e}$, and correspondingly increasing the convergence time. The fundamental problem here is that we are generating samples based on the original distribution $\Pr(\mathbf{X})$, where we ideally want to generate samples based on the posterior distribution $\Pr(\mathbf{X}|\mathbf{e})$, which can be shown to be the optimal choice in a precise sense [28]. The problem, however, is that $\Pr(\mathbf{X}|\mathbf{e})$ is not readily available to sample from. Hence, more sophisticated approaches for sampling attempt to sample from distributions that are meant to be close to $\Pr(\mathbf{X}|\mathbf{e})$, possibly changing the sampling distribution (also known as an importance function) as the sampling process proceeds and more information is gained. This includes the

methods of *likelihood weighting* [154, 63], *self-importance sampling* [154], *heuristic importance* [154], *adaptive importance sampling* [28], and *evidence pre-propagation importance sampling* (EPIS-BN) algorithm [179]. Likelihood weighing is perhaps the simplest of these methods. It works by generating samples that are guaranteed to be consistent with evidence $\mathbf{e}$, by avoiding to sample values for variables $\mathbf{E}$, always setting them to $\mathbf{e}$ instead. It also assigns a weight of $\prod_{\theta_{e|\mathbf{u}}:e\mathbf{u}\sim\mathbf{x}} \theta_{e|\mathbf{u}}$ to each sample $\mathbf{x}$. Likelihood weighing will then use these weighted samples for approximating the probabilities of events. The current state of the art for sampling in Bayesian networks is probably the EPIS-BN algorithm, which estimates the optimal importance function using belief propagation (see Section 11.4.2) and then proceeds with sampling.

Another class of sampling methods is based on *Markov Chain Monte Carlo* (*MCMC*) simulation [23, 128]. Procedurally, samples in MCMC are generated by first starting with a random sample $\mathbf{x}_0$ that is consistent with evidence $\mathbf{e}$. A sample $\mathbf{x}_i$ is then generated based on sample $\mathbf{x}_{i-1}$ by choosing a new value of some nonevidence variable X by sampling from the distribution $\Pr(X|\mathbf{x}_i - X)$. This means that samples $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ will disagree on at most one variable. It also means that the sampling distribution is potentially changed after each sample is generated. MCMC approximations will converge to the true probabilities if the network parameters are strictly positive, yet the algorithm is known to suffer from convergence problems in case the network parameters are extreme. Moreover, the sampling distribution of MCMC will convergence to the optimal one if the network parameters satisfy some (ergodic) properties [178].

One specialized class of sampling methods, known as *particle filtering*, deserves particular attention at it applies to DBNs [93]. In this class, one generates *particles* instead of *samples*, where a particle is an instantiation of the variables at a given time slice t. One starts by a set of n particles for the initial time slice $t = 0$, and then moves forward generating particles $\mathbf{x}^t$ for time t based on the particles $\mathbf{x}^{t-1}$ generated for time $t - 1$. In particular, for each particle $\mathbf{x}^{t-1}$, we sample a particle $\mathbf{x}^t$ based on the distributions $\Pr(X^t|\mathbf{x}^{t-1})$, in a fashion similar to logic sampling. The particles for time t can then be used to approximate the probabilities of events corresponding to that slice. As with other sampling algorithms, particle filtering needs to deal with the problem of unlikely evidence, a problem that is more exaggerated in the context of DBNs as the evidence pertaining to slices $t > i$ is generally not available when we generate particles for times $t \leqslant i$. One simple approach for addressing this problem is to *resample* the particles for time t based on the extent to which they are compatible with the evidence $\mathbf{e}^t$ at time t. In particular, we regenerate n particles for time t from the original set based on the weight $\Pr(\mathbf{e}^t|\mathbf{x}^t)$ assigned to each particle $\mathbf{x}^t$. The family of particle filtering algorithms include other proposals for addressing this problem.

11.4.2 Inference as Optimization

The second class of approximate inference algorithms for PR can be understood in terms of reducing the problem of inference to one of optimization. This class includes *belief propagation* (e.g., [130, 117, 56, 176]) and *variational* methods (e.g., [92, 85]).

Given a Bayesian network which induces a distribution Pr, variational methods work by formulating approximate inference as an optimization problem. For example, say we are interested in searching for an approximate distribution $\widehat{\Pr}$ which is more

well behaved computationally than Pr. In particular, if Pr is induced by a Bayesian network $\mathcal{N}$ which has a high treewidth, then $\widehat{\Pr}$ could possibly be induced by another network $\widehat{\mathcal{N}}$ which has a manageable treewidth. Typically, one starts by choosing the structure of network $\widehat{\mathcal{N}}$ to meet certain computational constraints and then search for a parametrization of $\widehat{\mathcal{N}}$ that minimizes the KL-divergence between the original distribution Pr and the approximate one $\widehat{\Pr}$ [100]:

$$KL\big(\widehat{\Pr}(.|\mathbf{e}), \Pr(.|\mathbf{e})\big) = \sum_{w} \widehat{\Pr}(w|\mathbf{e}) \log \frac{\widehat{\Pr}(w|\mathbf{e})}{\Pr(w|\mathbf{e})}.$$

Ideally, we want parameters of network $\widehat{\mathcal{N}}$ that minimize this KL-divergence, while possibly satisfying additional constraints. Often, we can simply set to zero the partial derivatives of $KL(\widehat{\Pr}(.|\mathbf{e}), \Pr(.|\mathbf{e}))$ with respect to the parameters, and perform an iterative search for parameters that solve the resulting system of equations. Note that the KL-divergence is not symmetric. In fact, one would probably want to minimize $KL(\Pr(.|\mathbf{e}), \widehat{\Pr}(.|\mathbf{e}))$ instead, but this is not typically done due to computational considerations (see [57, 114] for approaches using this divergence, based on local optimizations).

One of the simplest variational approaches is to choose a completely disconnected network $\widehat{\mathcal{N}}$, leading to what is known as a *mean-field* approximation [72]. Other variational approaches typically assume a particular structure of the approximate model, such as chains [67], trees [57, 114], disconnected subnetworks [149, 72, 175], or just tractable substructures in general [173, 65]. These methods are typically phrased in the more general setting of graphical models (which includes other representational schemes, such as Markov Networks), but can typically be adapted to Bayesian networks as well. We should note here that the choice of approximate network $\widehat{\mathcal{N}}$ should at least permit one to evaluate the KL-divergence between $\widehat{\mathcal{N}}$ and the original network $\mathcal{N}$ efficiently. As mentioned earlier, such approaches seek minima of the KL-divergence, but typically search for parameters where the partial derivatives of the KL-divergence are zero, i.e., parameters that are stationary points of the KL-divergence. In this sense, variational approaches can reduce the problem of inference to one of optimization. Note that methods identifying stationary points, while convenient, only approximate the optimization problem since stationary points do not necessarily represent minima of the KL-divergence, and even when they do, they do not necessarily represent global minima.

Methods based on belief propagation [130, 117, 56] are similar in the sense that they also can be understood as solving an optimization problem. However, this understanding is more recent and comes as an after fact of having discovered the first belief propagation algorithm, known as loopy belief propagation or iterative belief propagation (IBP). In IBP, the approximate distribution $\widehat{\Pr}$ is assumed to have a particular factored form:

$$\widehat{\Pr}(\mathbf{X}|\mathbf{e}) = \prod_{X \in \mathbf{X}} \frac{\widehat{\Pr}(X\mathbf{U}|\mathbf{e})}{\prod_{U \in \mathbf{U}} \widehat{\Pr}(U|\mathbf{e})}, \tag{11.5}$$

where $U \in \mathbf{U}$ are parents of the node X in the original Bayesian network $\mathcal{N}$. This form allows one to decompose the KL-divergence between the original and approximate

distributions as follows:

$$KL\big(\widehat{\Pr}(.|\mathbf{e}), \Pr(.|\mathbf{e})\big) = \sum_{x\mathbf{u}} \widehat{\Pr}(x\mathbf{u}|\mathbf{e}) \log \frac{\widehat{\Pr}(x\mathbf{u}|\mathbf{e})}{\prod_{u\sim\mathbf{u}} \widehat{\Pr}(u|\mathbf{e})} - \sum_{x\mathbf{u}} \widehat{\Pr}(x\mathbf{u}|\mathbf{e}) \log \theta_{x|\mathbf{u}} + \log \Pr(\mathbf{e}).$$

This decomposition of the KL-divergence has important properties. First, the term $\Pr(\mathbf{e})$ does not depend on the approximate distribution and can be ignored in the optimization process. Second, all other terms are expressed as a function of the approximate marginals $\widehat{\Pr}(x\mathbf{u}|\mathbf{e})$ and $\widehat{\Pr}(u|\mathbf{e})$, in addition to the original network parameters $\theta_{x|\mathbf{u}}$. In fact, IBP can be interpreted as searching for values of these approximate marginals that correspond to stationary points of the KL-divergence: ones that set to zero the partial derivatives of the divergence with respect to these marginals (under certain constraints). There is a key difference between the variational approaches based on searching for parameters of approximate networks and those based on searching for approximate marginals: The computed marginals may not actually correspond to any particular distribution as the optimization problem solved does not include enough constraints to ensure the global coherence of these marginals (only node marginals are consistent, e.g., $\widehat{\Pr}(x|\mathbf{e}) = \sum_{\mathbf{u}} \widehat{\Pr}(x\mathbf{u}|\mathbf{e})$).

The quality of approximations found by IBP depends on the extent to which the original distribution can indeed be expressed as given in (11.5). If the original network $\mathcal{N}$ has a polytree structure, the original distribution can be expressed as given in (11.5) and the stationary point obtained by IBP corresponds to exact marginals. In fact, the form given in (11.5) is not the only one that allows one to set up an optimization problem as given above. In particular, any factored form that has the structure:

$$\widehat{\Pr}(.|\mathbf{e}) = \frac{\prod_{\mathbf{C}} \widehat{\Pr}(\mathbf{C}|\mathbf{e})}{\prod_{\mathbf{S}} \widehat{\Pr}(\mathbf{S}|\mathbf{e})}, \tag{11.6}$$

where $\mathbf{C}$ and $\mathbf{S}$ are sets of variables, will permit a similar decomposition of the KL-divergence in terms of marginals $\widehat{\Pr}(\mathbf{C}|\mathbf{e})$ and $\widehat{\Pr}(\mathbf{S}|\mathbf{e})$. This leads to a more general framework for approximate inference, known as *generalized belief propagation* [176]. Note, however, that this more general optimization problem is exponential in the sizes of sets $\mathbf{C}$ and $\mathbf{S}$. In fact, any distribution induced by a Bayesian network $\mathcal{N}$ can be expressed in the above form, if the sets $\mathbf{C}$ and $\mathbf{S}$ correspond to the clusters and separators of a jointree for network $\mathcal{N}$ [130]. In that case, the stationary point of the optimization problem will correspond to exact marginals, yet the size of the optimization problem will be at least exponential in the network treewidth. The form in (11.6) can therefore be viewed as allowing one to trade the complexity of approximate inference with the quality of computed approximations, with IBP and jointree factorizations being two extreme cases on this spectrum. Methods for exploring this spectrum include joingraphs (which generalize jointrees) [1, 48], region graphs [176, 169, 170], and partially ordered sets (or posets) [111], which are structured methods for generating factorizations with interesting properties.

The above optimization perspective on belief propagation algorithms is only meant to expose the semantics behind these methods. In general, belief propagation algorithms do not set up an explicit optimization problem as discussed above. Instead,

they operate by passing messages in a Bayesian network (as is done by IBP), a joingraph, or some other structure such as a region graph. For example, in a Bayesian network, the message sent from a node X to its neighbor Y is based on the messages that node X receives from its other neighbors $Z \neq Y$. Messages are typically initialized according to some fixed strategy, and then propagated according to some message passing schedule. For example, one may update messages in parallel or sequentially [168, 164]. Additional techniques are used to fine tune the propagation method, including message dampening [117, 78]. When message propagation converges (if it does), the computed marginals are known to correspond to stationary points of the KL-divergence as discussed above [176, 79]. There are methods that seek to optimize the divergence directly, but they may be slow to converge [180, 171, 94, 174].

Statistical physics happens to be the source of inspiration for many of these methods and perspectives. In particular, we can reformulate the optimization of the KL-divergence in terms of optimizing a *variational free energy* that approximates a free energy (e.g., in thermodynamics). The free energy approximation corresponding to IBP and Eq. (11.5) is often referred to as the *Bethe free energy* [176]. Other free energy approximations in physics that improve on, or generalize, the Bethe free energy have indeed lent themselves to generalizing belief propagation. Among them is the *Kikuchi free energy* [177], which led to *region-based free energy* approximations for generalized belief propagation algorithms [176].

11.5 Constructing Bayesian Networks

Bayesian networks can be constructed in a variety of methods. Traditionally, Bayesian networks have been constructed by knowledge engineers in collaboration with domain experts, mostly in the domain of medical diagnosis. In more recent applications, Bayesian networks are typically synthesized from high level specifications, or learned from data. We will review each of these approaches in the following sections.

11.5.1 Knowledge Engineering

The construction of Bayesian networks using traditional knowledge engineering techniques has been most prevalent in medical reasoning, which also constitute some of the first significant applications of Bayesian networks to real-world problems. Some of the notable examples in this regard include: The Quick Medical Reference (QMR) model [113] which was later reformulated as a Bayesian network model [159] that covers more than 600 diseases and 4000 symptoms; the CPCS-PM network [137, 125], which simulates patient scenarios in the medical field of hepatobiliary disease; and the MUNIN model for diagnosing neuromuscular disorders from data acquired by electromyographic (EMG) examinations [7, 5, 6], which covers 8 nerves and 6 muscles.

The construction of Bayesian networks using traditional knowledge engineering techniques has been recently made more effective through progress on the subject of *sensitivity analysis*: a form of analysis which focuses on understanding the relationship between local network parameters and global conclusions drawn from the network [102, 18, 90, 98, 19–21]. These results have lead to the creation of efficient sensitivity analysis tools which allow experts to assess the significance of network parameters, and to easily isolate problematic parameters when obtaining counterintuitive results to posed queries.

11.5.2 High-Level Specifications

The manual construction of large Bayesian networks can be laborious and error-prone. In many domains, however, these networks tend to exhibit regular and repetitive structures, with the regularities manifesting themselves both at the level of individual CPTs and at the level of network structure. We have already seen in Section 11.2.4 how regularities in a CPT can reduce the specification of a large CPT to the specification of a few parameters. A similar situation can arise in the specification of a whole Bayesian network, allowing one to synthesize a large Bayesian network automatically from a compact, high-level specification that encodes probabilistic dependencies among network nodes, in addition to network parameters.

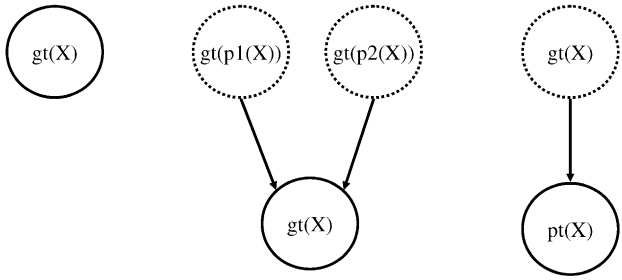
This general *knowledge-based model construction* paradigm [172] has given rise to many concrete high-level specification frameworks, with a variety of representation styles. All of these frameworks afford a certain degree of modularity, thus facilitating the adaptation of existing specifications to changing domains. A further benefit of high-level specifications lies in the fact that the smaller number of parameters they contain can often be learned from empirical data with higher accuracy than the larger number of parameters found in the full Bayesian network [59, 96]. We next describe some fundamental paradigms for high-level representation languages, where we distinguish between two main paradigms: template-based and programming-based. It must be acknowledged, however, that this simple distinction is hardly adequate to account for the whole variety of existing representation languages.

Template-based representations

The prototypical example of template-based representations is the dynamic Bayesian network described in Section 11.2.6. In this case, one specifies a DBN having an arbitrary number of slices using only two templates: one for the initial time slice, and one for all subsequent slices. By further specifying the number of required slices t, a Bayesian network of arbitrary size can be compiled from the given templates and temporal horizon t.

One can similarly specify other types of large Bayesian networks that are composed of identical, recurring segments. In general, the template-based approach requires two components for specifying a Bayesian network: a set of network templates whose instantiation leads to network segments, and a specification of which segments to generate and how to connect them together. Fig. 11.8 depicts three templates from the domain of genetics, involving two classes of variables: genotypes (*gt*) and phenotypes (*pt*). Each template contains nodes of two kinds: nodes representing random variables that are created by instantiating the template (solid circles, annotated with CPTs), and nodes for input variables (dashed circles). Given these templates, together with a pedigree which enumerates particular individuals with their parental relationships, one can then generate a concrete Bayesian network by instantiating one genotype template and one phenotype template for each individual, and then connecting the resulting segments depending on the pedigree structure. The particular genotype template instantiated for an individual will depend on whether the individual is a founder (has no parents) in the pedigree.

The most basic type of template-based representations, such as the one in Fig. 11.8, is quite rigid as all generated segments will have exactly the same structure. More



$gt(X)$		
AA	Aa	aa
0.49	0.42	0.09

		$gt(X)$		
$gt(p1(X))$	$gt(p2(X))$	AA	Aa	aa
AA	AA	1.0	0.0	0.0
AA	Aa	0.5	0.5	0.0
...	...	...	...	...
aa	aa	0.0	0.0	1.0

	$pt(X)$	
$gt(X)$	affected	not affected
AA	0.0	1.0
Aa	0.0	1.0
aa	1.0	0.0

Figure 11.8: Templates for specifying a Bayesian network in the domain of genetics. The templates assume three possible genotypes (AA, Aa, aa) and two possible phenotypes (affected, not affected).

sophisticated template-based representations add flexibility to the specification in various ways. *Network fragments* [103] allow nodes in a template to have an unspecified number of parents. The CPT for such nodes must then be specified by generic rules. *Object oriented Bayesian networks* [99] introduce abstract classes of network templates that are defined by their interface with other templates. *Probabilistic relational models* enhance the template approach with elements of relational database concepts [59, 66], by allowing one to define probabilities conditional on aggregates of the values of an unspecified number of parents. For example, one might include nodes *life_expectancy*(X) and *age_at_death*(X) into a template for individuals X, and condition the distribution of *life_expectancy*(X) on the average value of the nodes *age_at_death*(Y) for all ancestors Y of X.

Programming-based representations

Frameworks in this group contain some of the earliest high-level representation languages. They use procedural or declarative specifications, which are not as directly connected to graphical representations as template-based representations. Many are based on logic programming languages [17, 132, 71, 118, 96]; others resemble functional programming [86] or deductive database [69] languages. Compared to template-based approaches, programming-based representations can sometimes allow more modular and intuitive representations of high-level probabilistic knowledge. On the other hand, the compilation of the Bayesian network from the high-level specification

Table 11.3. A probabilistic Horn clause specification

alarm(X)	$\leftarrow$	*burglary*(X): 0.95
alarm(X)	$\leftarrow$	*quake*(Y), *lives_in*(X, Y): 0.8
call(X, Z)	$\leftarrow$	*alarm*(X), *neighbor*(X, Z): 0.7
call(X, Z)	$\leftarrow$	*prankster*(Z), *neighbor*(X, Z): 0.1
comb(*alarm*): noisy-or		
comb(*call*): noisy-or		

Table 11.4. CPT for ground atom *alarm*(Holmes)

burglary(Holmes)	*quake*(LA)	*lives_in*(Holmes, LA)	*quake*(SF)	*lives_in*(Holmes, SF)	*alarm*(Holmes)
t	t	t	t	t	0.998
t	t	t	t	f	0.99
f	t	t	f	f	0.8
t	f	f	f	f	0.95
...	...	...	...	...	...

is usually not as straightforward, and part of the semantics of the specification can be hidden in the details of the compilation process.

Table 11.3 shows a basic version of a representation based on probabilistic Horn clauses [71]. The logical atoms *alarm*(X), *burglary*(X), ... represent generic random variables. Ground instances of these atoms, e.g., *alarm*(Holmes), *alarm*(Watson), become the actual nodes in the constructed Bayesian network. Each clause in the probabilistic rule base is a partial specification of the CPT for (ground instances of) the atom in the head of the clause. The second clause in Table 11.3, for example, stipulates that *alarm*(X) depends on variables *quake*(Y) and *lives_in*(X, Y). The parameters associated with the clauses, together with the *combination rules* associated with each relation determine how a full CPT is to be constructed for a ground atom. Table 11.4 depicts part of the CPT constructed for *alarm*(Holmes) when Table 11.3 is instantiated over a domain containing an individual Holmes and two cities LA and SF. The basic probabilistic Horn clause paradigm illustrated in Table 11.3 can be extended and modified in many ways; see for example *Context-sensitive probabilistic knowledge bases* [118] and *Relational Bayesian networks* [86].

Specifications such as the one in Table 11.3 need not necessarily be seen as high-level specifications of Bayesian networks. Provided the representation language is equipped with a well-defined probabilistic semantics that is not defined procedurally in terms of the compilation process, such high-level specifications are also stand-alone probabilistic knowledge representation languages. It is not surprising, therefore, that some closely related representation languages have been developed which were not intended as high-level Bayesian network specifications [148, 116, 135, 140].

Inference

Inference on Bayesian networks generated from high-level specifications can be performed using standard inference algorithms discussed earlier. Note, however, that the

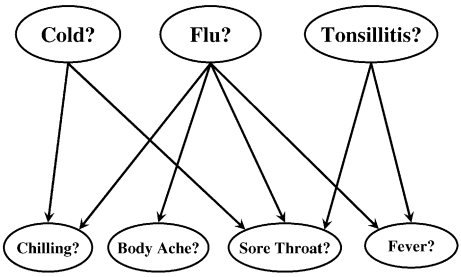


Figure 11.9: A Bayesian network structure for medical diagnosis.

Table 11.5. A data set for learning the structure in Fig. 11.9

Case	*Cold*?	*Flu*?	*Tonsillitis*?	*Chilling*?	*Body ache*?	*Sore throat*?	*Fever*?
1	true	false	?	true	false	false	false
2	false	true	false	true	true	false	true
3	?	?	true	false	?	true	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

generated networks can be very large and very connected (large treewidth), and therefore often pose particular challenges to inference algorithms. As an example, observe that the size of the CPT for *alarm*(Holmes) in Table 11.4 grows exponentially in the number of cities in the domain. Approximate inference techniques, as described in Section 11.4, are therefore particularly important for Bayesian networks generated from high-level specifications. One can also optimize some of these algorithms, such as sampling methods, for Bayesian networks compiled from these specifications [126]. It should also be noted that such Bayesian networks can sometimes be rich with local structure, allowing exact inference even when the network treewidth is quite high [27].

Exact inference algorithms that operate directly on high-level specifications have also been investigated. Theoretical complexity results show that in the worst case one cannot hope to obtain more efficient algorithms than standard exact inference on the compiled network [87]. This does not, however, preclude the possibility that high-level inference methods can be developed that are more efficient for particular applications and particular queries [133, 43].

11.5.3 Learning Bayesian Networks

A Bayesian network over variables $X_1, \ldots, X_n$ can be learned from a data set over these variables, which is a table with each row representing a partial instantiation of variables $X_1, \ldots, X_n$. Table 11.5 depicts an example data set for the network in Fig. 11.9.

Each row in the above table represents a medical case of a particular patient, where ? indicates the unavailability of corresponding data for that patient. It is typically assumed that when variables have missing values, one cannot conclude anything from

that fact that the values are missing (e.g., a patient did not take an X-ray because the X-ray happened to be unavailable that day) [108].

There are two orthogonal dimensions that define the process of learning a Bayesian network from data: the task for which the Bayesian network will be used, and the amount of information available to the learning process. The first dimension decides the criteria by which we judge the quality of a learned network, that is, it decides the objective function that the learning process will need to optimize. This dimension calls for distinguishing between learning *generative* versus *discriminative* Bayesian networks. To make this distinction more concrete, consider again the data set shown in Table 11.5. A good *generative* Bayesian network is one that correctly models all of the correlations among the variables. This model could be used to accurately answer any query, such as the correlations between *Chilling*? and *BodyAche*?, as well as whether a patient has *Tonsilitis* given any other (partial) description of that patient. On the other hand, a discriminative Bayesian network is one that is intended to be used only as a classifier: determining the value of one particular variable, called the *class variable*, given the values of some other variables, called the *attributes* or *features*. When learning a discriminative network, we will therefore optimize the classification power of the learned network, without necessarily insisting on the global quality of the distribution it induces. Hence, the answers that the network may generate for other types of queries may not be meaningful. This section will focus on generative learning of networks; for information on discriminative learning of networks, see [84, 70].

The second dimension calls for distinguishing between four cases:

1. *Known network structure, complete data*. Here, the goal is only to learn the parameters Θ of a Bayesian network as the structure G is given as input to the learning process. Moreover, the given data is complete in the sense that each row in the data set provides a value for each network variable.

2. *Known network structure, incomplete data*. This is similar to the above case, but some of the rows may not have values for some of the network variables; see Table 11.5.

3. *Unknown network structure, complete data*. The goal here is to learn both the network structure and parameters, from complete data.

4. *Unknown network structure, incomplete data*. This is similar to Case 3 above, but where the data is incomplete.

In the following discussion, we will only consider the learning of Bayesian networks in which CPTs have tabular representations, but see [60] for results on learning networks with structured CPT representations.

Learning network parameters

We will now consider the task of learning Bayesian networks whose structure is already known and then discuss the case of unknown structure. Suppose that we have a complete data set $\mathcal{D}$ over variables $\mathbf{X} = X_1, \ldots, X_n$. The first observation here is to view this data set as defining a probability distribution $\widehat{\Pr}$ over these variables, where $\widehat{\Pr}(\mathbf{x}) = count(\mathbf{x}, \mathcal{D})/|\mathcal{D}|$ is simply the percentage of rows in $\mathcal{D}$ that contain the instantiation $\mathbf{x}$. Suppose now that we have a Bayesian network structure G and our goal

is to learn the parameters Θ of this network given the data set $\mathcal{D}$. This is done by choosing parameters Θ so that the network (G, Θ) will induce a distribution $\Pr_\Theta$ that is as close to $\widehat{\Pr}$ as possible, according to the KL-divergence. That is, the goal is to minimize:

$$KL(\widehat{\Pr}, \Pr_\Theta) = \sum_{\mathbf{x}} \widehat{\Pr}(\mathbf{x}) \log \frac{\widehat{\Pr}(\mathbf{x})}{\Pr_\Theta(\mathbf{x})}$$
$$= \sum_{\mathbf{x}} \widehat{\Pr}(\mathbf{x}) \log \widehat{\Pr}(\mathbf{x}) - \sum_{\mathbf{x}} \widehat{\Pr}(\mathbf{x}) \log \Pr_\Theta(\mathbf{x}).$$

Since the term $\sum_{\mathbf{x}} \widehat{\Pr}(\mathbf{x}) \log \widehat{\Pr}(\mathbf{x})$ does not depend on the choice of parameters Θ, this corresponds to maximizing $\sum_{\mathbf{x}} \widehat{\Pr}(\mathbf{x}) \log \Pr_\Theta(\mathbf{x})$, which can be shown to equal[6]:

$$g(\Theta) = \sum_{\mathbf{x}} \widehat{\Pr}(\mathbf{x}) \log \Pr_\Theta(\mathbf{x}) = \frac{1}{|\mathcal{D}|} \log \prod_{d \in \mathcal{D}} \Pr_\Theta(d). \tag{11.7}$$

Note that parameters which maximize the above quantity will also maximize the probability of data, $\prod_{d \in \mathcal{D}} \Pr_\Theta(d)$ and are known as *maximum likelihood parameters*. A number of observations are in order about this method of learning. First, there is a unique set of parameters $\Theta = \{\theta_{x|\mathbf{u}}\}$ that satisfy the above property, defined as follows: $\theta_{x|\mathbf{u}} = count(x\mathbf{u}, \mathcal{D})/count(\mathbf{u}, \mathcal{D})$ (e.g., see [115]). Second, this method may have problems when the data set does not contain enough cases, leading possibly to $count(\mathbf{u}, \mathcal{D}) = 0$ and a division by zero. This is usually handled by using (something like) a Laplacian correction; using, say

$$\theta_{x|\mathbf{u}} = \frac{1 + count(x, \mathbf{u}, \mathcal{D})}{|X| + count(\mathbf{u}, \mathcal{D})}, \tag{11.8}$$

where $|X|$ is the number of values for variable X. We will refer to these parameters as $\hat{\Theta}(G, \mathcal{D})$ from now on.

When the data is incomplete, the situation is not as simple for a number of reasons. First, we may have multiple sets of maximum likelihood parameters. Second, the two most commonly used methods that search for such parameters are not optimal, and both can be computationally intensive. Both methods are based on observing, from Eq. (11.7), that we are trying to optimize a function $g(\Theta)$ of the network parameters. The first method tries to optimize this function using standard gradient ascent techniques [146]. That is, we first compute the gradient which happens to have the following form:

$$\frac{\partial g}{\partial \theta_{x|\mathbf{u}}}(\Theta) = \sum_{d \in \mathcal{D}} \frac{\Pr_\Theta(x\mathbf{u}|d)}{\theta_{x|\mathbf{u}}}, \tag{11.9}$$

and then use it to drive a gradient ascent procedure that attempts to find a local maxima of the function g. This method will start with some initial parameter Θ^0, leading to an initial Bayesian network (G, Θ^0) with distribution $\Pr^0_\Theta$. It will then use Eq. (11.9) to compute the gradient $\partial g/\partial \theta_{x|\mathbf{u}}(\Theta^0)$, which is then used to find the next set of parameters Θ^1, with corresponding network (G, Θ^1) and distribution $\Pr^1$. The process

[6] We are treating a data set as a multi-set, which can include repeated elements.

continues, computing a new set of parameters Θ^i based on the previous set Θ^{i-1}, until some convergence criteria is satisfied. Standard techniques of gradient ascent all are applicable in this case, including conjugate gradient, line search and random restarts [14].

A more commonly used method in this case is the *expectation maximization* (*EM*) algorithm [104, 112], which works as follows. The method starts with some initial parameters Θ^0, leading to an initial distribution $\Pr^0_\Theta$. It then uses the distribution to complete the data set $\mathcal{D}$ as follows. If d is a row in $\mathcal{D}$ for which some variable values are missing, the algorithm will (conceptually) consider every completion d' of this row and assign it a weight of $\Pr^0_\Theta(d'|d)$. The algorithm will then pretend as if it had a complete (but weighted) data set, and use the method for complete data to compute a new set of parameters Θ^1, leading to a new distribution $\Pr^1_\Theta$. This process continues, computing a new set of parameters Θ^i based on the previous set Θ^{i-1}, until some convergence criteria is satisfied. This method has a number of interesting properties. First, the value of parameters at iteration i have the following closed from:

$$\theta^i_{x|\mathbf{u}} = \frac{\sum_{d\in\mathcal{D}} \Pr^{i-1}_\Theta(x\mathbf{u}|d)}{\sum_{d\in\mathcal{D}} \Pr^{i-1}_\Theta(\mathbf{u}|d)},$$

which has the same complexity as the gradient ascent method (see Eq. (11.9)). Second, the probability of the data set is guaranteed to never decrease after each iteration of the method. There are many techniques to make this algorithm even more efficient; see [112].

Learning network structure

We now turn to the problem of learning a network *structure* (as well as the associated parameters), given complete data. As this task is NP-hard in general [30], the main algorithms are iterative, starting with a single structure (perhaps the empty graph), and incrementally modifying this structure, until reaching some termination condition. There are two main classes of algorithms, score-based and independence-based.

As the name suggests, the algorithms based on independence will basically run a set of independence tests, between perhaps every pair of currently-unconnected nodes in the current graph, to see if the data set supports the claim that they are independent given the rest of the graph structure; see [68, 127].

Score-based algorithms will typically employ local search, although systematic search has been used in some cases too (e.g., [165]). Local search algorithms will evaluate the current structure, as well as every structure formed by some simple modification—such as adding one addition arc, or deleting one existing arc, or changing the direction of one arc [29]—and climb to the new structure with the highest score. One plausible score is based on favoring structures that lead to higher probability of the data:

$$g_\mathcal{D}(G) = \max_\Theta \log \prod_{d\in\mathcal{D}} \Pr_{G,\Theta}(d). \tag{11.10}$$

Unfortunately, this does not always work. To understand why, consider the simpler problem of fitting a polynomial to some pairs of real numbers. If we do not fix the degree of the polynomial, we would probably end up fitting the data perfectly by

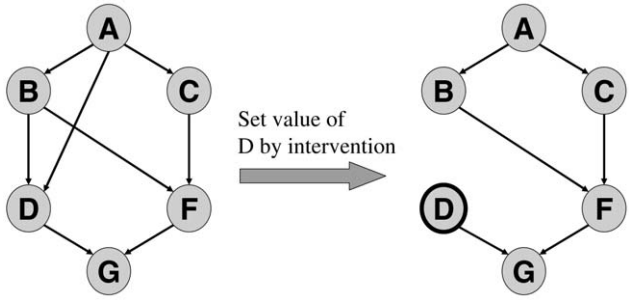


Figure 11.10: Modeling intervention on causal networks.

selecting a high degree polynomial. Even though this may lead to a perfect fit over the given data points, the learned polynomial may not generalize the data well, and so do poorly at labeling other novel data points. The same phenomena, called *overfitting* [141], shows up in learning Bayesian networks, as it means we would favor a fully connected network, as clearly this complete graph would maximize the probability of data due to its large set of parameters (maximal degrees of freedom). To deal with this overfitting problem, other scoring functions are used, many explicitly including a penalty term for complex structure. This includes the Minimum Description Length (MDL) score [142, 62, 101, 163], the Akaike Information Criterion (AIC) score [16], and the "Bayesian Dirichlet, equal" (BDe) [33, 75, 74]. For example, the MDL score is given by:

$$MDL_{\mathcal{D}}(G) = g_{\mathcal{D}}(G) - \frac{\log m}{2} k(G),$$

where m is the size of data set $\mathcal{D}$ and $k(G)$ is the number of independent network parameters (this also corresponds to the Bayesian Information Criterion (BIC) [151]). Each of these scores is asymptotically correct in that it will identify the correct structures in the limit as the data increases.

The above discussion has focused on learning arbitrary network structures. There are also efficient algorithms for computing the optimal structures, for some restricted class of structures, including trees [31] and polytrees [131].

If the data is incomplete, learning structures becomes much more complicated as we have two nested optimization problems: one for choosing the structure, which can again be accomplished by either greedy or optimal search, and one for choosing the parameters for a given structure, which can be accomplished using methods like EM [75]. One can improve the double search problem by using techniques such as *structural EM* [58], which uses particular data structures that allow computational results to be used across the different iterations of the algorithm.

11.6 Causality and Intervention

The directed nature of Bayesian networks can be used to provide causal semantics for these networks, based on the notion of *intervention* [127], leading to models that not

only represent probability distributions, but also permit one to induce new probability distributions that result from intervention. In particular, a *causal network*, intuitively speaking, is a Bayesian network with the added property that the parents of each node are its direct causes. For example, $Cold \rightarrow HeadAche$ is a causal network whereas $HeadAche \rightarrow Cold$ is not, even though both networks are equally capable of representing any joint distribution on the two variables. Causal networks can be used to compute the result of intervention as illustrated in Fig. 11.10. In this example, we want to compute the probability distribution that results from having set the value of variable D by *intervention*, as opposed to having observed the value of D. This can be done by deactivating the current causal mechanism for D—by disconnecting D from its direct causes A and B—and then conditioning the modified causal model on the set value of D. Note how different this process is from the classical operation of *Bayes conditioning* (Eq. (11.1)), which is appropriate for modeling observations but not immediately for intervention. For example, intervening on variable D in Fig. 11.10 would have no effect on the probability associated with F, while measurements taken on variable D would affect the probability associated with F.[7] Causal networks are more properly defined, then, as Bayesian networks in which each parents–child family represents a stable causal mechanism. These mechanisms may be reconfigured locally by interventions, but remain invariant to other observations and manipulations.

Causal networks and their semantics based on intervention can then be used to answer additional types of queries that are beyond the scope of general Bayesian networks. This includes determining the truth of counterfactual sentences of the form $\alpha \rightarrow \beta \mid \gamma$, which read: "Given that we have observed γ, if α were true, then β would have been true". The counterfactual antecedent α consists of a conjunction of value assignments to variables that are forced to hold true by external intervention. Typically, to justify being called "counterfactual", α conflicts with γ. The truth (or probability) of a counterfactual conditional $\alpha \rightarrow \beta \mid \gamma$ requires a causal model. For example, the probability that "the patient would be alive had he not taken the drug" cannot be computed from the information provided in a Bayesian network, but requires a functional causal networks, where each variable is functionally determined by its parents (plus noise factors). This more refined specification allows one to assign unique probabilities to all counterfactual statements. Other types of queries that have been formulated with respect to functional causal networks include ones for distinguishing between direct and indirect causes and for determining the sufficiency and necessity aspects of causation [127].

Acknowledgements

Marek Druzdzel contributed to Section 11.4.1, Arthur Choi contributed to Section 11.4.2, Manfred Jaeger contributed to Section 11.5.2, Russ Greiner contributed to Section 11.5.3, and Judea Pearl contributed to Section 11.6. Mark Chavira, Arthur Choi, Rina Dechter, and David Poole provided valuable comments on different versions of this chapter.

[7] For a simple distinction between observing and intervening, note that observing D leads us to increase our belief in its direct causes, A and B. Yet, our beliefs will not undergo this increase when intervening to set D.

Bibliography

[1] S.M. Aji and R.J. McEliece. The generalized distributive law and free energy minimization. In *Proceedings of the 39th Allerton Conference on Communication, Control and Computing*, pages 672–681, 2001.

[2] D. Allen and A. Darwiche. New advances in inference by recursive conditioning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 2–10, 2003.

[3] D. Allen and A. Darwiche. Optimal time–space tradeoff in probabilistic inference. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 969–975, 2003.

[4] D. Allen and A. Darwiche. Advances in Bayesian networks. In *Studies in Fuzziness and Soft Computing*, vol. 146, pages 39–55. Springer-Verlag, New York, 2004 (chapter *Optimal Time–Space Tradeoff in Probabilistic Inference*).

[5] S. Andreassen, F.V. Jensen, S.K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A.R. Sorensen, A. Rosenfalck, and F. Jensen. MUNIN—an expert EMG assistant. In J.E. Desmedt, editor. *Computer-Aided Electromyography and Expert Systems*. Elsevier Science Publishers, Amsterdam, 1989 (Chapter 21).

[6] S. Andreassen, M. Suojanen, B. Falck, and K.G. Olesen. Improving the diagnostic performance of MUNIN by remodelling of the diseases. In *Proceedings of the 8th Conference on AI in Medicine in Europe*, pages 167–176. Springer-Verlag, 2001.

[7] S. Andreassen, M. Woldbye, B. Falck, and S.K. Andersen. Munin—a causal probabilistic network for interpretation of electromyographic findings. In J. McDermott, editor. *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 366–372. Morgan Kaufmann Publishers, 1987.

[8] S.A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey. *BIT*, 25:2–23, 1985.

[9] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic and Discrete Methods*, 8:277–284, 1987.

[10] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian inference via backtracking search. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 20–28. Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[11] A. Becker, R. Bar-Yehuda, and D. Geiger. Random algorithms for the loop cutset problem. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.

[12] J. Bilmes and C. Bartels. Triangulating dynamic graphical models. In *Uncertainty in Artificial Intelligence: Proceedings of the Nineteenth Conference*, pages 47–56, 2003.

[13] J. Binder, K. Murphy, and S. Russell. Space-efficient inference in dynamic probabilistic networks. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[14] C. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, Oxford, 1998.

[15] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 115–123, 1996.

[16] H. Bozdogan. Model selection and Akaike's Information Criterion (AIC): the general theory and its analytical extensions. *Psychometrica*, 52:345–370, 1987.

[17] J.S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 8(4):624–647, 1992.

[18] E. Castillo, J.M. Gutiérrez, and A.S. Hadi. Sensitivity analysis in discrete Bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 27:412–423, 1997.

[19] H. Chan and A. Darwiche. When do numbers really matter? *Journal of Artificial Intelligence Research*, 17:265–287, 2002.

[20] H. Chan and A. Darwiche. Sensitivity analysis in Bayesian networks: From single to multiple parameters. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 67–75. AUAI Press, Arlington, VA, 2004.

[21] H. Chan and A. Darwiche. A distance measure for bounding probabilistic belief change. *International Journal of Approximate Reasoning*, 38:149–174, 2005.

[22] H. Chan and A. Darwiche. On the revision of probabilistic beliefs using uncertain evidence. *Artificial Intelligence*, 163:67–90, 2005.

[23] M.R. Chavez and G.F. Cooper. A randomized approximation algorithm for probabilistic inference on Bayesian belief networks. *Networks*, 20(5):661–685, 1990.

[24] M. Chavira, D. Allen, and A. Darwiche. Exploiting evidence in probabilistic inference. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 112–119, 2005.

[25] M. Chavira and A. Darwiche. Compiling Bayesian networks with local structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1306–1312, 2005.

[26] M. Chavira and A. Darwiche. Compiling Bayesian networks using variable elimination. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2443–2449, 2007.

[27] M. Chavira, A. Darwiche, and M. Jaeger. Compiling relational Bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1–2):4–20, May 2006.

[28] J. Cheng and M.J. Druzdzel. BN-AIS: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13:155–188, 2000.

[29] D.M. Chickering. Optimal structure identification with greedy search. *JMLR*, 2002.

[30] D.M. Chickering and D. Heckerman. Large-sample learning of Bayesian networks is NP-hard. *JMLR*, 2004.

[31] C.K. Chow and C.N. Lui. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[32] F.G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2–3):393–405, 1990.

[33] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *MLJ*, 9:309–347, 1992.
[34] R. Cowell, A. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.
[35] T. Verma, D. Geiger, and J. Pearl. d-separation: from theorems to algorithms. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 139–148, 1990.
[36] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.
[37] A. Darwiche. Conditioning algorithms for exact and approximate inference in causal networks. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 99–107, 1995.
[38] A. Darwiche. Any-space probabilistic inference. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 133–142, 2000.
[39] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1–2):5–41, 2001.
[40] A. Darwiche. A logical approach to factoring belief networks. In *Proceedings of KR*, pages 409–420, 2002.
[41] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.
[42] A. Darwiche and M. Hopkins. Using recursive decomposition to construct elimination orders, jointrees and dtrees. In *Trends in Artificial Intelligence*, *Lecture Notes in Artificial Intelligence*, vol. 2143, pages 180–191. Springer-Verlag, 2001.
[43] R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Proceedings of the Nineteenth Int. Joint Conf. on Artificial Intelligence (IJCAI-05)*, pages 1319–1325, 2005.
[44] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
[45] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.
[46] R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks and their and/or search space. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, pages 120–129, 2004.
[47] R. Dechter and Y. El Fattah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, 125(1–2):93–118, 2001.
[48] R. Dechter, K. Kask, and R. Mateescu. Iterative join-graph propagation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 128–136, 2002.
[49] R. Dechter and D. Larkin. Hybrid processing of beliefs and constraints. In *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001)*, pages 112–119. Morgan Kaufmann Publishers, San Francisco, CA, 2001.
[50] R. Dechter and D. Larkin. Bayesian inference in the presence of determinism. In C.M. Bishop and B.J. Frey, editors, In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, Key West, FL, 2003.

[51] F.J. Díez. Parameter adjustment in Bayesian networks: the generalized noisy-or gate. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence (UAI)*, 1993.

[52] F.J. Díez. Local conditioning in Bayesian networks. *Artificial Intelligence*, 87(1):1–20, 1996.

[53] F.J. Díez and S.F. Galán. An efficient factorization for the noisy MAX. *International Journal of Intelligent Systems*, 18:165–177, 2003.

[54] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 18(1):189–198, 2002.

[55] M. Fishelson and D. Geiger. Optimizing exact genetic linkage computations. In *RECOMB'03*, 2003.

[56] B.J. Frey and D.J.C. MacKay. A revolution: Belief propagation in graphs with cycles. In *NIPS*, pages 479–485, 1997.

[57] B.J. Frey, R. Patrascu, T. Jaakkola, and J. Moran. Sequentially fitting "inclusive" trees for inference in noisy-or networks. In *NIPS*, pages 493–499, 2000.

[58] N. Friedman. The Bayesian structural EM algorithm. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998.

[59] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

[60] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1996.

[61] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 252–262, 1996.

[62] N. Friedman and Z. Yakhini. On the sample complexity of learning Bayesian networks. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1996.

[63] R. Fung and K.-C. Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In M. Henrion, R.D. Shachter, L.N. Kanal, and J.F. Lemmer, editors. *Uncertainty in Artificial Intelligence*, vol. 5, pages 209–219. Elsevier Science Publishing Company, Inc., New York, NY, 1989.

[64] D. Geiger, T. Verma, and J. Pearl. Identifying independence in Bayesian networks. *Networks*:507–534, 1990.

[65] D. Geiger and C. Meek. Structured variational inference procedures and their realizations. In *Proceedings of Tenth International Workshop on Artificial Intelligence and Statistics*. The Society for Artificial Intelligence and Statistics, The Barbados, January 2005.

[66] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *Proceedings of the 18th International Conference on Machine Learning*, pages 170–177, 2001.

[67] Z. Ghahramani and M.I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29(2–3):245–273, 1997.

[68] C. Glymour, R. Scheines, P. Spirtes, and K. Kelly. *Discovering Causal Structure*. Academic Press, Inc., London, 1987.

[69] R.P. Goldman and E. Charniak. Dynamic construction of belief networks. In P.P. Bonissone, M. Henrion, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, vol. 6, pages 171–184, Elsevier Science, 1991.

[70] Y. Guo and R. Greiner. Discriminative model selection for belief net structures. In *Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 770–776, Pittsburgh, July 2005.

[71] P. Haddawy. Generating Bayesian networks from probability logic knowledge bases. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 262–269, 1994.

[72] M. Haft, R. Hofmann, and V. Tresp. Model-independent mean-field theory as a local method for approximate propagation of information. *Network: Computation in Neural Systems*, 10:93–105, 1999.

[73] D. Heckerman. Causal independence for knowledge acquisition and inference. In D. Heckerman and A. Mamdani, editors, *Proc. of the Ninth Conf. on Uncertainty in AI*, pages 122–127, 1993.

[74] D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[75] D.E. Heckerman. A tutorial on learning with Bayesian networks. In M.I. Jordan, editor. *Learning in Graphical Models*. MIT Press, 1998.

[76] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence*, vol. 2, pages 149–163. Elsevier Science Publishing Company, Inc., New York, NY, 1988.

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

[78] T. Heskes. Stable fixed points of loopy belief propagation are local minima of the Bethe free energy. In *NIPS*, pages 343–350, 2002.

[79] T. Heskes. On the uniqueness of loopy belief propagation fixed points. *Neural Computation*, 16(11):2379–2413, 2004.

[80] J. Hoey, R. St-Aubin, A. Hu, and G. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 279–288, 1999.

[81] E.J. Horvitz, H.J. Suermondt, and G.F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of Conference on Uncertainty in Artificial Intelligence, Windsor, ON*, pages 182–193. Association for Uncertainty in Artificial Intelligence, Mountain View, CA, August 1989.

[82] J. Hastad. Tensor rank is NP-complete. *Journal of Algorithms*, 11:644–654, 1990.

[83] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[84] I. Inza, P. Larranaga, J. Lozano, and J. Pena. *Machine Learning Journal*, 59, June 2005 (Special Issue: Probabilistic Graphical Models for Classification).

[85] T. Jaakkola. *Advanced Mean Field Methods—Theory and Practice*. MIT Press, 2000 (chapter *Tutorial on Variational Approximation Methods*).

[86] M. Jaeger. Relational Bayesian networks. In D. Geiger and P.P. Shenoy, editors. *Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13)*, pages 266–273. Morgan Kaufmann, Providence, USA, 1997.

[87] M. Jaeger. On the complexity of inference about probabilistic relational models. *Artificial Intelligence*, 117:297–308, 2000.

[88] R. Jeffrey. *The Logic of Decision*. McGraw-Hill, New York, 1965.

[89] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[90] F.V. Jensen. Gradient descent training of Bayesian networks. In *Proceedings of the Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 5–9, 1999.

[91] F.V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, 2001.

[92] M.I. Jordan, Z. Ghahramani, T. Jaakkola, and L.K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[93] K. Kanazawa, D. Koller, and S.J. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, pages 346–351, 1995.

[94] H.J. Kappen and W. Wiegerinck. Novel iteration schemes for the cluster variation method. In *NIPS*, pages 415–422, 2001.

[95] K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 129:91–131, 2001.

[96] K. Kersting and L. De Raedt. Towards combining inductive logic programming and Bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP-2001)*, *Springer Lecture Notes in AI*, vol. 2157. Springer, 2001.

[97] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eight Conference*, pages 121–129, 1992.

[98] U. Kjaerulff and L.C. van der Gaag. Making sensitivity analysis computationally efficient. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

[99] D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 302–313. Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[100] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[101] W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computation Intelligence*, 10(4):269–293, 1994.

[102] K.B. Laskey. Sensitivity analysis for probability assessments in Bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 25:901–909, 1995.

[103] K.B. Laskey and S.M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. In *Proceedings of the Thirteenth Annual*

Conference on Uncertainty in Artificial Intelligence (UAI–97), pages 334–341. San Morgan Kaufmann Publishers, Francisco, CA, 1997.

[104] S.L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[105] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of Royal Statistics Society, Series B*, 50(2):157–224, 1988.

[106] V. Lepar and P.P. Shenoy. A comparison of Lauritzen–Spiegelhalter, Hugin, and Shenoy–Shafer architectures for computing marginals of probability distributions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 328–337. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[107] Y. Lin and M. Druzdzel. Computational advantages of relevance reasoning in Bayesian belief networks. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 342–350, 1997.

[108] J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data*. Wiley, New York, 1987.

[109] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, MD, 1983.

[110] R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[111] R.J. McEliece and M. Yildirim. Belief propagation on partially ordered sets. In J. Rosenthal and D.S. Gilliam, editors, *Mathematical Systems Theory in Biology, Communications, Computation and Finance*.

[112] G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. *Wiley Series in Probability and Statistics. Applied Probability and Statistics*. Wiley, New York, 1997.

[113] R.A. Miller, F.E. Fasarie, and J.D. Myers. Quick medical reference (QMR) for diagnostic assistance. *Medical Computing*, 3:34–48, 1986.

[114] T.P. Minka and Y.(A.) Qi. Tree-structured approximations by expectation propagation. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2003.

[115] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[116] S. Muggleton. Stochastic logic programs. In L. de Raedt, editor. *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

[117] K.P. Murphy, Y. Weiss, and M.I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[118] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.

[119] A. Nicholson and J.M. Brady. The data association problem when monitoring robot vehicles using dynamic belief networks. In *10th European Conference on Artificial Intelligence Proceedings*, pages 689–693, 1992.

[120] T. Nielsen, P. Wuillemin, F. Jensen, and U. Kjaerulff. Using ROBDDs for inference in Bayesian networks with troubleshooting as an example. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 426–435, 2000.

[121] J.D. Park and A. Darwiche. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI–03)*, pages 459–468. Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[122] J. Park and A. Darwiche. Morphing the Hugin and Shenoy–Shafer architectures. In *Trends in Artificial Intelligence*, *Lecture Notes in AI*, vol. 2711, pages 149–160. Springer-Verlag, 2003.

[123] J. Park and A. Darwiche. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research*, 21:101–133, 2004.

[124] J. Park and A. Darwiche. A differential semantics for jointree algorithms. *Artificial Intelligence*, 156:197–216, 2004.

[125] R.C. Parker and R.A. Miller. Using causal knowledge to create simulated patient cases: The CPCS project as an extension of Internist-1. In *Proceedings of the Eleventh Annual Symposium on Computer Applications in Medical Care*, pages 473–480. IEEE Comp. Soc. Press, 1987.

[126] H. Pasula and S. Russell. Approximate inference for first-order probabilistic languages. In *Proceedings of IJCAI-01*, pages 741–748, 2001.

[127] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, 2000.

[128] J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32(2):245–257, 1987.

[129] J. Pearl. Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986.

[130] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[131] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[132] D. Poole. Probabilistic horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[133] D. Poole. First-order probabilistic inference. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[134] D. Poole and N.L. Zhang. Exploiting contextual independence in probabilistic inference. *Journal of Artificial Intelligence*, 18:263–313, 2003.

[135] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1–2):7–56, 1997.

[136] D. Poole. Context-specific approximation in probabilistic inference. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 447–454, 1998.

[137] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference (UAI-94)*, pages 484–490. Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[138] A. Krogh, R. Durbin, S. Eddy, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[139] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *IEEE /ACM International Conference on CAD*, pages 188–191. IEEE Computer Society Press, Santa Clara, CA, 1993.

[140] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1–2):107–136, 2006.

[141] B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.

[142] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, 1989.

[143] S.L. Lauritzen, D.J. Spiegelhalter, R.G. Cowell, and A.P. Dawid. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[144] N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of treewidth. *J. Algorithms*, 7:309–322, 1986.

[145] D. Roth. *On the hardness of approximate reasoning*. *Artificial Intelligence*, 82(1–2):273–302, April 1996.

[146] S. Russell, J. Binder, D. Koller, and K. Kanazawa. Local learning in probabilistic networks with hidden variables. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 1146–1152, 1995.

[147] T. Sang, P. Beame, and H. Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, vol. 1, pages 475–482. AAAI Press, 2005.

[148] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*, pages 715–729, 1995.

[149] L.K. Saul and M.I. Jordan. Exploiting tractable substructures in intractable networks. In *NIPS*, pages 486–492, 1995.

[150] P. Savicky and J. Vomlel. Tensor rank-one decomposition of probability tables. In *Proceedings of the Eleventh Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU)*, pages 2292–2299, 2006.

[151] G. Schwartz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[152] R. Shachter, S.K. Andersen, and P. Szolovits. Global Conditioning for Probabilistic Inference in Belief Networks. In *Proc. Tenth Conference on Uncertainty in AI*, pages 514–522, Seattle WA, 1994.

[153] R. Shachter, B.D. D'Ambrosio, and B. del Favero. Symbolic probabilistic inference in belief networks. In *Proc. Conf. on Uncertainty in AI*, pages 126–131, 1990.

[154] R.D. Shachter and M.A. Peot. Simulation approaches to general probabilistic inference on belief networks. In M. Henrion, R.D. Shachter, L.N. Kanal, and J.F. Lemmer, editors. *Uncertainty in Artificial Intelligence*, vol. 5, pages 221–231. Elsevier Science Publishing Company, Inc., New York, NY, 1989.

[155] R. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, 1986.

[156] R. Shachter. Evidence absorption and propagation through evidence reversals. In M. Henrion, R.D. Shachter, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, vol. 5, pages 173–189, Elsvier Science, 1990.

[157] P.P. Shenoy and G. Shafer. Propagating belief functions with local computations. *IEEE Expert*, 1(3):43–52, 1986.

[158] S.E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.

[159] M. Shwe, B. Middleton, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base I. The probabilistic model and inference algorithms. *Methods of Information in Medicine*, 30:241–255, 1991.

[160] P. Smyth, D. Heckerman, and M.I. Jordan. Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9(2):227–269, 1997.

[161] S. Srinivas. A generalization of the noisy-or model. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence (UAI)*, 1993.

[162] H.J. Suermondt, G.F. Cooper, and D.E. Heckerman. A combination of cutset conditioning with clique-tree propagation in the Pathfinder system. In *Proceedings of the 6th Annual Conference on Uncertainty in Artificial Intelligence (UAI-91)*, pages 245–253. Elsvier Science, New York, NY, 1991.

[163] J. Suzuki. Learning Bayesian belief networks based on the MDL principle: An efficient algorithm using the branch and bound technique. *Annals of Statistics*, 6, 1978.

[164] M.F. Tappen and W.T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *ICCV*, pages 900–907, 2003.

[165] J. Tian. A branch-and-bound algorithm for MDL learning Bayesian networks. In C. Boutilier and M. Goldszmidt, editors, *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, Stanford, CA, pages 580–588, 2000.

[166] T. Verma and J. Pearl. Causal networks: Semantics and expressiveness. In *Proceedings of the 4th Workshop on Uncertainty in AI*, pages 352–359, Minneapolis, MN, 1988.

[167] J. Vomlel. Exploiting functional dependence in Bayesian network inference. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 528–535. Morgan Kaufmann Publishers, 2002.

[168] M.J. Wainwright, T. Jaakkola, and A.S. Willsky. Tree-based reparameterization for approximate inference on loopy graphs. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 1001–1008, 2001.

[169] M. Welling. On the choice of regions for generalized belief propagation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, page 585. AUAI Press, Arlington, VA, 2004.

[170] M. Welling, T.P. Minka, and Y.W. Teh. Structured region graphs: morphing EP into GBP. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

[171] M. Welling and Y.W. Teh. Belief optimization for binary networks: A stable alternative to loopy belief propagation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 554–561, 2001.

[172] M.P. Wellman, J.S. Breese, and R.P. Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(1):35–53, 1992.

[173] W. Wiegerinck. Variational approximations between mean field theory and the junction tree algorithm. In *UAI*, pages 626–633, 2000.
[174] W. Wiegerinck and T. Heskes. Fractional belief propagation. In *NIPS*, pages 438–445, 2002.
[175] E.P. Xing, M.I. Jordan, and S.J. Russell. A generalized mean field algorithm for variational inference in exponential families. In *UAI*, pages 583–591, 2003.
[176] J. Yedidia, W. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.
[177] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. Technical Report TR-2001-022, MERL, 2001. Available online at http://www.merl.com/publications/TR2001-022/.
[178] J. York. Use of the Gibbs sampler in expert systems. *Artificial Intelligence*, 56(1):115–130, 1992, http://dx.doi.org/10.1016/0004-3702(92)90066-7.
[179] C. Yuan and M.J. Druzdzel. An importance sampling algorithm based on evidence pre-propagation. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 624–631. Morgan Kaufmann Publishers, San Francisco, CA, 2003.
[180] A.L. Yuille. Cccp algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14(7):1691–1722, 2002.
[181] N.L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 171–178, 1994.
[182] N.L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.