International Journal on Artificial Intelligence Tools
Vol. 14, No. 3 (2005) 507–543


World Scientific
www.worldscientific.com

# THE HUGIN TOOL FOR PROBABILISTIC GRAPHICAL MODELS

ANDERS L. MADSEN*, FRANK JENSEN†, UFFE B. KJÆRULFF‡ and MICHAEL LANG§

*Hugin Expert A/S, Gasværksvej 5, DK-9000 Aalborg, Denmark*
**anders@hugin.com*
†*fj@hugin.com*
‡*uk@hugin.com*
§*michael@hugin.com*

Received 25 January 2004
Accepted 22 November 2004

As the framework of probabilistic graphical models becomes increasingly popular for knowledge representation and inference, the need for efficient tools for its support is increasing. The Hugin Tool is a general purpose tool for construction, maintenance, and deployment of Bayesian networks and influence diagrams. This paper surveys the key functionality of the Hugin Tool and reports on new advances of the tool. Furthermore, an empirical analysis reports on the efficiency of the Hugin Tool on common inference and learning tasks.

*Keywords*: Probabilistic reasoning; decision making; (mixed) Bayesian networks; influence diagrams.

## 1. Introduction

The framework of probabilistic graphical models such as Bayesian networks and influence diagrams is an increasingly popular paradigm for reasoning and decision making under uncertainty.[1,2,3,4,5] The need for efficient tools for its support is therefore increasing. The Hugin Tool is a general purpose tool for construction, maintenance, and deployment of Bayesian networks and influence diagrams.[6,7,8]

The framework of probabilistic graphical models provides a range of methods for knowledge representation in structured domains with inherent uncertainty. This framework is particularly useful when the properties of the problem domain can be modeled using structural relationships that encode dependence and independence assumptions among entities in the domain. These relationships, encoding the qualitative domain knowledge, can describe causal (directed) interactions or symmetric (undirected) interactions, and they can be derived from domain experts' experience, from data, or from both of these sources.

The range of knowledge representation methods provided by the Hugin Tool includes Bayesian networks (with purely discrete variables, purely continuous variables, or a mixture of discrete and continuous variables), influence diagrams, and object-oriented Bayesian networks and influence diagrams.

The Hugin Tool offers a variety of functionality for constructing, solving, and analyzing the aforementioned types of probabilistic graphical models. The functionality includes manual construction of models, object-oriented models, learning models from a database of cases, inference and solution of models, and data conflict analysis to name a few.

The Hugin Tool consists of the Hugin Decision Engine and the Hugin Graphical User Interface. The Hugin Decision Engine is the inference engine — the heart of the Hugin Tool — and it provides fundamental functionalities for model construction, learning, and analysis. The Hugin Graphical User Interface provides the user with an easy-to-use interface to the advanced Hugin Decision Engine.

The original Hugin Tool (Hugin Shell) dates back to the late 1980s.[6] A recent survey describes the general functionality of the Hugin Tool.[7] Another paper describes the learning functionality in more detail.[8] The present paper describes in detail all current functionalities of the Hugin Tool, and illustrates the performance of its main functionalities.

The purpose of this paper is to survey the key functionality of the Hugin Tool, to share knowledge on implementation aspects of the tool, and to report on the new advances of the tool. The experiments of the paper reports on the efficiency of the Hugin Tool on common inference and learning tasks. A second purpose of the paper is to offer a point of reference for both researchers and professionals in the field of artificial intelligence tools for probabilistic graphical models.

The Hugin Tool is one of the most important and popular tools for probabilistic graphical model. It is used by both researchers and professionals in the field of probabilistic graphical models and many other fields due to its reliability, stability, and efficiency. For this reason the Hugin Tool provides crucial support in the evolution of science and technology within the field of probabilistic graphical models.

This paper is organized as follows: Section 2 describes the knowledge representation and specifications supported by the Hugin Tool, Section 3 describes how inferences are made in the light of observations. Section 4 describes the method supporting data conflict analysis, i.e. the activity of detecting, tracing, and explaining possible conflicts among observations. Section 5 describes how the Hugin Tool supports construction of Bayesian networks by fusion of data and expert knowledge. Section 6 presents the results of a performance analysis. Finally, Section 7 concludes the paper with some general remarks on the Hugin Tool.

## 2. Knowledge Representation and Specification

A probabilistic graphical model $\mathcal{N}$ be it a (mixed) Bayesian network or an influence diagram over a set of variables consists of a qualitative part and a quantitative part. The qualitative part encodes dependence relations between chance variables in the case of Bayesian networks and encodes dependence relations between chance variables conditional on decision variables, information ordering, and preference relations in the case of influence diagrams.

The qualitative knowledge is specified in the form of an acyclic, directed graph (DAG) $G = (V, E)$ with nodes corresponding to chance variables, decision variables, and additive terms of a utility function. Each node of the graph $G$ representing either a decision variable or a chance variable corresponds one-to-one with a variable of $\mathcal{N}$. (We use the terms *node* and *variable* interchangeably.)

The structure of a Bayesian network or an influence diagram is encoded in the DAG, $G = (V, E)$, over the set of domain quantities or variables, $V$, where each variable, $X \in V$, can have zero or more parent variables, $\pi(X)$ (i.e. variables connected to $X$ through a directed edge, $e = (Y, X) \in E$ where $Y \in \pi(X)$, emanating from each of these variables). We denote variables by uppercase letters $X, Y, \ldots$, sets of variables by uppercase letters $S, S_{XY}, \ldots$, and states or values of variables by lowercase letters $x, y, \ldots$.

The dependence and independence relations among the variables depend on the observations available (i.e., observations of actual states or values of the variables). Simple rules are available for reading these relations from a DAG.[9,10] We use the notation that the relation $X \perp\!\!\!\perp_P Y \mid S$ states conditional independence between $X$ and $Y$ given $S$ under the probability distribution $P$ whereas $X \perp_G Y \mid S$ states conditional independence between $X$ and $Y$ given $S$ in the DAG $G$ (i.e. $d$-separation).

The quantitative part specifies the strengths of dependence relations and the value of preference relations. The quantitative domain knowledge is provided through local conditional probability distributions, density functions, and utility functions as described below.

### 2.1. *Bayesian networks*

A Bayesian network over purely discrete variables $\mathcal{N} = (G, \mathcal{P})$ consists of a DAG $G = (V, E)$ and a set of probability distributions $\mathcal{P}$. For each variable $X \in V$ there is a conditional probability distribution $P(X \mid \pi(X)) \in \mathcal{P}$. Through the conditional probability distributions (i.e. the parameters of $\mathcal{N}$), it is possible to specify unconstrained probabilistic dependence relations between a variable and its parents.

A Bayesian network $\mathcal{N} = (G, \mathcal{P})$ is an efficient representation of a joint probability distribution $P(V)$ over $V$ when $G = (V, E)$ is not dense. The joint probability distribution $P(V)$ factorizes according to the structure of $G$ as:

$$P(V) = \prod_{X \in V} P(X \mid \pi(X)). \tag{1}$$

This is referred to as the chain rule of Bayesian networks.

The above factorization of the joint probability distribution $P$ induced by $\mathcal{N}$ is equivalent to the conditional independence assumptions $X \perp\!\!\!\perp \mathrm{nd}(X) \mid \pi(X)$, for all $X \in V$. That is, variable $X$ is conditionally independent of its non-descendants $\mathrm{nd}(X)$ given its parents $\pi(X)$.[4] This is known as the local directed Markov property of Bayesian networks. Notice, that the specification and representation of a joint probability distribution is simplified significantly by the local directed Markov property.

A famous toy example of a Bayesian network is the Chest Clinic example.[11,2]

**Example 2.1.** Shortness-of-breath (dyspnoea) ($D$) may be due to tuberculosis ($T$), lung cancer ($L$) or bronchitis ($B$), or none of them, or more than one of them. A recent visit to Asia ($A$) increases the chances of tuberculosis, while smoking ($S$) is known to be a risk factor for both lung cancer and bronchitis. The result of a single chest X-ray ($X$) does not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea.

The qualitative knowledge of this diagnostic problem can be captured by the Bayesian network shown in Fig. 1 where ovals represent discrete binary chance variables with states yes and no. The network contains a mediating variable ($E$) representing the disjunction of tuberculosis and lung cancer.

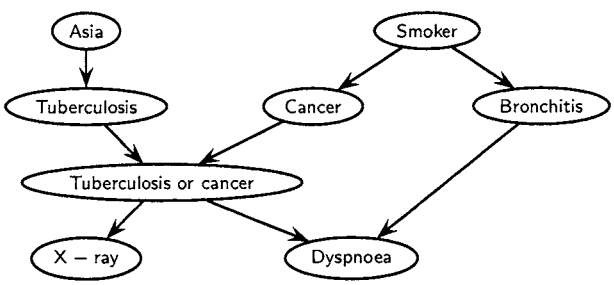


Fig. 1. The Chest Clinic example.

The quantitative knowledge of the Bayesian network is as follows: $P(A) = (0.01, 0.99)$, $P(S) = (0.5, 0.5)$, and $P(E \mid T, L) = T \vee L$. See Table 1 and Table 2 for specification of the remaining conditional probability distributions.

Table 1. The quantification of the conditional probability distributions $P(L \mid S)$, $P(B \mid S)$, $P(T \mid A)$, and $P(X \mid E)$.

| $P(L \mid S)$ | $S =$ yes | $S =$ no | $P(B \mid S)$ | $S =$ yes | $S =$ no |
|---|---|---|---|---|---|
| $L =$ yes | 0.1 | 0.01 | $B =$ yes | 0.6 | 0.3 |
| $L =$ no | 0.9 | 0.99 | $B =$ no | 0.4 | 0.7 |

| $P(T \mid A)$ | $A =$ yes | $A =$ no | $P(X \mid E)$ | $E =$ yes | $E =$ no |
|---|---|---|---|---|---|
| $T =$ yes | 0.05 | 0.01 | $X =$ yes | 0.98 | 0.05 |
| $T =$ no | 0.95 | 0 99 | $X =$ no | 0.02 | 0.95 |

The Bayesian network encodes the joint probability distribution:

$$P(S, A, T, L, B, E, X, D) = P(S)P(A)P(T \mid A)P(L \mid S)P(B \mid S)P(E \mid L, T) \\ P(X \mid E)P(D \mid B, E). \quad (2)$$

Table 2. The quantification of the conditional probability distribution $P(D \mid B, E)$.

| | $B$ = yes | | $B$ = no | |
|---|---|---|---|---|
| | $E$ = yes | $E$ = no | $E$ = yes | $E$ = no |
| $D$ = yes | 0.9 | 0.8 | 0.7 | 0.1 |
| $D$ = no | 0.1 | 0.2 | 0.3 | 0.9 |

## 2.2. *Mixed Bayesian networks*

A mixed Bayesian network $\mathcal{N} = (G, \mathcal{P})$ over a mixture of continuous and discrete variables consists of a DAG $G$ and a set of conditional probability distributions and density functions $\mathcal{P}$.[12,13] The variables of $\mathcal{N}$ are partitioned into the set of continuous variables $V_\Gamma$ and the set of discrete variables $V_\Delta$. For each variable $X \in V_\Delta$ there is a conditional probability distribution $P(X \mid \pi(X))$, whereas for each variable $X \in V_\Gamma$ there is a conditional density function $f(X \mid \pi(X))$.

A mixed Bayesian network of continuous and discrete variables is an efficient representation of a conditional Gaussian distribution. The joint probability distribution over the continuous variables given the discrete variables is a multivariate Gaussian such that:

$$\mathcal{L}(V_\Gamma \mid V_\Delta = i) = \mathbb{N}_{|V_\Gamma|}(\xi(i), \Sigma(i)) \text{ whenever } P(V_\Delta = i) > 0, \tag{3}$$

where $|V_\Gamma|$ is the cardinality of $V_\Gamma$, $\xi(i)$ is a table of mean vectors, $\Sigma(i)$ is a table of positive semidefinite covariance matrices, and $V_\Delta = i$ is an instantiation of the discrete variables $V_\Delta$.[13] The variables $V = V_\Gamma \cup V_\Delta$ are said to follow a CG distribution.

Let $X$ be a continuous variable with continuous parents $Z = \pi(X) \cap V_\Gamma = \{Z_1, \ldots, Z_n\}$ and discrete parents $I = \pi(X) \cap V_\Delta = \{I_1, \ldots, I_m\}$. Then the conditional Gaussian distribution of $X$ given $Z$ and $I$ is:

$$\mathcal{L}(X \mid Z = z, I = i) = \mathbb{N}\left(\alpha(i) + \sum_{k=1}^{n} \beta_k(i) z_k, \gamma(i)\right), \tag{4}$$

where $\alpha(i), \beta_k(i) \in \mathbb{R}$ and $\gamma(i) \geq 0$ for each pair of constants $(i, k)$. The mean value of $X$ is a multi-linear function in the values of its continuous parents conditional on its discrete parents. Hence, we specify one linear regression for each configuration of the discrete parents of $X$. Each constant $\beta_k(i)$ is the coefficient of a particular continuous parent for a given configuration of the discrete parents, while each $\alpha_k(i)$ is the constant contribution to the mean, and each $\gamma_k(i)$ is the conditional variance.

Notice that the mean of $X$ depends linearly on the values of the continuous parent variables $\{Z_1, \ldots, Z_n\}$, while the covariance is independent of the continuous parent variables. Notice also that the variance may be zero making it possible to model deterministic linear relations between continuous variables.

A mixed Bayesian network represents a conditional Gaussian distribution:

$$P(i) * \mathbb{N}_{|V_\Gamma|}(\mu(i), \Sigma(i)), \tag{5}$$

where $P(i) = \prod_{X \in V_\Delta} P(X = i_X \mid \pi(X) = i_{\pi(X)})$ and $i_X, i_{\pi(X)}$ are the values of $X, \pi(X)$ in the instantiation $i$ of $V_\Delta$, respectively.

The notion of a CG potential $\phi = [p, A, B, C](V_H \mid V_T)$ where $V_H$ and $V_T$ denotes a partitioning of the continuous variables of the domain of $\phi$ into head and tail is introduced to represent CG regressions: $P(i) \propto p(i)$, $\mathcal{L}(X \mid Z = z, I = i) = \mathbb{N}(A(i) + B(i)z, C(i))$. Using the notion of CG potentials, the joint distribution of a mixed Bayesian network (Eq. (5)), which is a multivariate linear Gaussian distribution conditional on the discrete variables, can be written as:

$$\phi(V) = \bigotimes_{X \in V} \phi(X \mid \pi(X)), \tag{6}$$

where $\otimes$ is the combination operator. A conditional probability distribution $P(X \mid \pi(X))$ of a discrete variable $X$ given its parents $\pi(X)$ is represented as $[P, -, -, -](-, -)$.[13] Eq. (5) is referred to as the chain rule of mixed Bayesian networks.

The local directed Markov property of Bayesian networks is valid for mixed Bayesian networks.

The Waste Incinerator example is an example of a mixed Bayesian network.[12,2]

**Example 2.2.** The emissions from a waste incinerator differ because of compositional differences in incoming waste ($W$). Another important factor is the waste burning regime ($B$), which can be monitored by measuring the concentration of $CO_2$ in the emissions ($C$). The filter efficiency ($E$) depends on the technical state of the electro-filter ($F$) and the amount and composition of the waste ($W$). The emission of metal ($M_o$) depends on both the concentration of metals in the incoming waste ($M_i$) and the emission of dust particulates in general. The emission of dust ($D$) is monitored through measuring the penetrability of light ($L$).

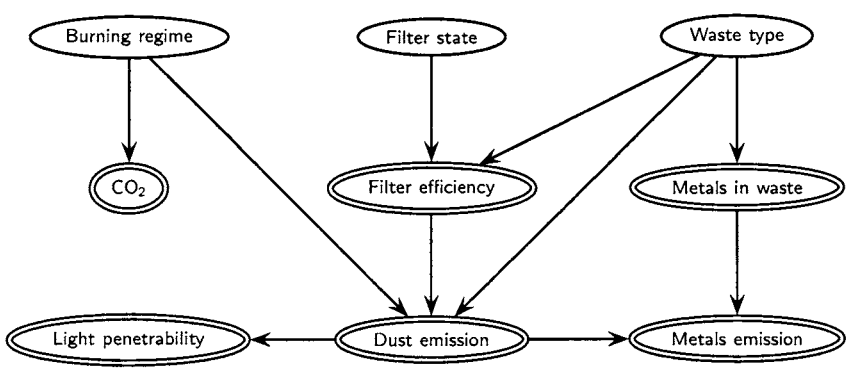


Fig. 2. The Waste Incinerator example.

The qualitative knowledge of this example can be captured by the mixed Bayesian network shown in Fig. 2 where double lined ovals represent continuous

chance variables. The quantitative knowledge is as follows. The (conditional) probability distributions of the example are $P(B = \text{stable}) = 1 - P(B = \text{unstable}) = 0.85$, $P(F = \text{intact}) = 1 - P(F = \text{defective}) = 0.95$, and $P(W = \text{industrial}) = 1 - P(W = \text{household}) = 2/7$. The distributional forms of the continuous variables are (we use a logarithmic scale):

$$\mathcal{L}(M_o \mid d, m_i) = \mathbb{N}(d + m_i, 0.002). \tag{7}$$

$$\mathcal{L}(M_i \mid \text{industrial}) = \mathbb{N}(0.5, 0.01). \tag{8}$$
$$\mathcal{L}(M_i \mid \text{household}) = \mathbb{N}(-0.5, 0.005). \tag{9}$$

$$\mathcal{L}(L \mid d) = \mathbb{N}(3.0 - 0.5d, 0.25). \tag{10}$$

$$\mathcal{L}(C \mid \text{stable}) = \mathbb{N}(-2.0, 0.1). \tag{11}$$
$$\mathcal{L}(C \mid \text{unstable}) = \mathbb{N}(-1.0, 0.3). \tag{12}$$

$$\mathcal{L}(D \mid \text{stable}, \text{industrial}, e) = \mathbb{N}(6.5 + e, 0.03). \tag{13}$$
$$\mathcal{L}(D \mid \text{stable}, \text{household}, e) = \mathbb{N}(6.0 + e, 0.04). \tag{14}$$
$$\mathcal{L}(D \mid \text{unstable}, \text{industrial}, e) = \mathbb{N}(7.5 + e, 0.1). \tag{15}$$
$$\mathcal{L}(D \mid \text{unstable}, \text{household}, e) = \mathbb{N}(7.0 + e, 0.1). \tag{16}$$

$$\mathcal{L}(E \mid \text{intact}, \text{industrial}) = \mathbb{N}(-3.9, 0.00002). \tag{17}$$
$$\mathcal{L}(E \mid \text{intact}, \text{household}) = \mathbb{N}(-3.2, 0.00002). \tag{18}$$
$$\mathcal{L}(E \mid \text{defective}, \text{industrial}) = \mathbb{N}(-0.4, 0.0001). \tag{19}$$
$$\mathcal{L}(E \mid \text{defective}, \text{household}) = \mathbb{N}(-0.5, 0.0001). \tag{20}$$

The mixed Bayesian network encodes a CG distribution using the following factorization into CG potentials:

$$\begin{aligned}\phi(B, F, W, C, E, M_i, D, L, M_o) = {} & \phi(B) \otimes \phi(F) \otimes \phi(W) \otimes \phi(C \mid B) \\ & \otimes \phi(E \mid F, W) \otimes \phi(M_i \mid W) \otimes \phi(D \mid B, W, E) \\ & \otimes \phi(L \mid D) \otimes \phi(M_o \mid M_i).\end{aligned} \tag{21}$$

### 2.3. *Influence diagrams*

An influence diagram is a Bayesian network augmented with decision facilities, i.e. augmented with decision alternatives (decision variables), preference relations (utility functions), and information constraints. An influence diagram $\mathcal{N} = (G, \mathcal{P}, \mathcal{U})$ consists of a DAG $G$, a set of conditional probability distributions $\mathcal{P}$, and a set of (local) utility functions $\mathcal{U}$. The variables of $\mathcal{N}$ are partitioned into the set of chance variables $V_C$ and the set of decision variables $V_D$. The utility functions of $\mathcal{N}$ are represented as utility nodes $V_U$ in $G$. For each variable $X \in V_C$ there is a conditional probability distribution $P(X \mid \pi(X))$ and for each utility node $U \in V_U$ there is a

local utility function $U(\pi(U))$, whereas a decision variable $D \in V_D$ does not have any associated function or distribution. The task of solving an influence diagram is to find an optimal strategy $\hat{\Delta} = \{\delta_1, \ldots, \delta_n\}$ consisting of one policy for each decision $D \in V_D$. A policy is a mapping from (relevant) past observations and decisions to current decision alternatives.

An influence diagram supports the representation and solution of sequential decision problems with an additively decomposing utility function under the no-forgetting assumption (i.e., we assume perfect recall of all observations and decisions made in the past).

The order of observation and decision making is specified using informational edges (or knowledge edges), which are edges into decision variables. An informational edge specifies that the value of a certain variable is observed prior to a decision or that one decision is made prior to another decision. In this way, informational edges are used to indicate the order in which observations and decisions are made. The informational edges must enforce a total order on the decisions (due to the no-forgetting assumption, some informational edges may be left implicit in the structure). This implies that the set of chance variables and decision variables are subject to a partial ordering $\prec$:

$$\mathcal{I}_0 \prec D_1 \prec \mathcal{I}_1 \prec \cdots \prec \mathcal{I}_{i-1} \prec D_i \prec \mathcal{I}_i \prec \cdots \prec D_n \prec \mathcal{I}_n, \tag{22}$$

where $\mathcal{I}_i$ is the set of chance variables observed after making decision $D_i$ and before making decision $D_{i+1}$, for all $i = 1, \ldots, n-1$, $\mathcal{I}_0$ is the set of chance variables observed initially, and $\mathcal{I}_n$ is the set of chance variables never observed or observed after the last decision has been made. If the influence diagram is not constructed or used according to this constraint, the strategy may not be optimal and the computed expected utilities will (of course) not be correct.

The utility function over the set of chance and decision variables decomposes into terms $\mathcal{U}$ whereas the conditional probability distribution of the chance variables given the decision variables decomposes into factors $\mathcal{P}$. This decomposition produces the chain rule of influence diagrams:

$$\mathrm{EU}(V) = \prod_{X \in V_C} P(X \,|\, \pi(X)) \sum_{U \in V_U} u(\pi(U)). \tag{23}$$

The local directed Markov property of Bayesian networks is also valid for the distribution induced by $\mathcal{P}$, i.e. the conditional probability distribution of the chance variables given the decision variables.

The Oil Wildcatter example illustrates the use of an influence diagram.[14]

**Example 2.3.** An oil wildcatter must decide either to drill or not to drill ($D$). He is uncertain whether the hole is dry, wet, or soaking ($O$). The wildcatter could take seismic soundings that will help determine the geological structure of the site ($T$). The soundings ($S$) will give a closed reflection pattern (indication of much oil), an open pattern (indication of some oil), or a diffuse pattern (almost no hope of oil).

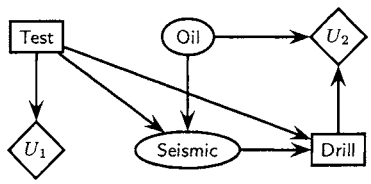


Fig. 3. The Oil Wildcatter example.

The qualitative knowledge of this example can be captured by the influence diagram shown in Fig. 3 where boxes represent decision variables and diamonds represent (local) utility functions.

The probabilistic quantification of the model is $P(O) = (0.5, 0.3, 0.2)$ and $P(S \mid T, O)$ is shown in Table 3.

Table 3. The quantification of $P(S \mid O, T)$.

| | $T$ = no | | | $T$ = yes | | |
|---|---|---|---|---|---|---|
| | $O$ = dry | $O$ = wet | $O$ = soaking | $O$ = dry | $O$ = wet | $O$ = soaking |
| $S$ = diffuse | 1/3 | 1/3 | 1/3 | 0.6 | 0.3 | 0.1 |
| $S$ = open | 1/3 | 1/3 | 1/3 | 0.3 | 0.4 | 0.4 |
| $S$ = close | 1/3 | 1/3 | 1/3 | 0.1 | 0.3 | 0.5 |

The cost of testing is 10 units of utility, whereas the cost of drilling is 70. The utility of drilling is 0, 120, and 270 for a soaking, wet, and dry hole, respectively. That is, $U(T) = (0, -10)$ and $U(D = yes, O) = (-70, 50, 200)$.

The influence diagram encodes the joint expected utility function:

$$EU(T, O, S, D) = P(S \mid T, O)P(O)(U(T) + U(D, O)). \tag{24}$$

## 2.4. *Object-oriented modeling*

The Hugin Tool supports object-oriented specification of (mixed) Bayesian networks and influence diagrams, which makes it simple to reuse models, to encapsulate submodels (providing a means for hierarchical model specification), and to perform model construction in a top-down fashion, a bottom-up fashion, or a mixture of the two (allowing repeated changes of level of abstraction).[15] Specifying a model in a hierarchical fashion often makes the model less cluttered, and thus provides a better means of communicating ideas among knowledge engineers, domain experts, and users.

In addition to the usual nodes, the graph of an object-oriented probabilistic graphical model contains instance nodes representing instances of other networks. An instance node represents an instantiation of a network class within another network class. A network class is a named and self-contained description of a probabilistic graphical model, which is characterized by its name, interface, and hidden

part. As instance nodes can be nested, an object-oriented network can be viewed as a hierarchical description of a problem domain. Thus, an instance node $M$ is the realization of a network class $C_M$ within another network class $C_N$, see Fig. 4. A network class $C$ is a DAG over three pairwise disjoint sets of nodes $I(C)$, $H(C)$, $O(C)$ where $I(C)$ are the input nodes, $H(C)$ are the hidden nodes, and $O(C)$ are the output nodes of $C$. The set $I \cup O$ is the interface of $C$. Interface nodes may represent either decision or chance variables, whereas hidden nodes may be instances of network classes, decision variables, chance variables, and utility functions.

An input node $X$ of an instance $M$ is a placeholder for a node (the parent of $X$) in the encapsulating class of $M$. For this reason an input node has at most one parent. An output node $Y$ of instance $M$ enlarges the visibility of $Y$ to include the encapsulating network class of $M$.

Let $M$ be an instance of network class $C$. Each input node $I \in I(C)$ has no parent in $C$, no children outside $C$, and its corresponding node in $M$ has at most one parent in the encapsulating class of $M$. Each output node $O \in O(C)$ may only have parents in $I(C) \cup H(C)$. The children and parents of $H \in H(C)$ are subsets of $V(C)$

An input node $I$ of an instance $M$ of network class $C$ is bound if it has a parent $X$ in the network class encapsulating $M$. Each chance input node $I$ of a class $C$ is assigned a default prior probability distribution, which becomes the probability distribution of the node $I$ in all instances of $C$ where $I$ is an unbound input node.

**Example 2.4.** Fig. 4 shows an instance node $M$ of a network class $C_M$ instantiated within network class $C_N$. Network class $C_N$ has input node $C_1$, hidden nodes $C_3$ and $M$, and output node $C_2$. The network class $C_M$ has input nodes $C_1$ and $C_2$, output node $C_3$, and unknown hidden nodes. The input node $C_1$ of instance $M$ is bound to $C_1$ of $C_N$ whereas $C_2$ is unbound.

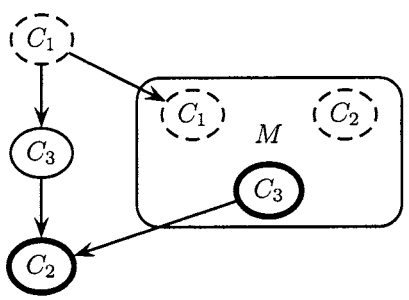


Fig. 4. $M$ is an instance of a network class $C_M$ within another network class $C_N$.

An instance node $M$ encapsulates a conditional probability distribution over the variables of its network class $C$ given its bound input nodes. An object-oriented

network $\mathcal{N}$ has an equivalent flat or unfolded network model representation. The unfolded network model of an object-oriented network $\mathcal{N}$ is obtained by recursively unfolding the instance nodes of $\mathcal{N}$. The unfolded network representation of a network class is important as it is the structure used for inference.

The object-oriented Bayesian network represents a hierarchical decomposition of a joint probability distribution, whereas the object-oriented influence diagram represents a hierarchical decomposition of a joint expected utility function.

As systems often are composed of collections of identical or similar components, models of systems often contain repetitive patterns. The notion of instance nodes makes it simple to construct multiple identical instances of a network class as the following example shows (this example is an extension of the Stud Farm example [3]).

**Example 2.5.** The probability of a given horse carrying a hereditary disease can often be determined from the probability of each parent carrying the disease. This can be represented as a Bayesian network. However, if the model for each horse gets just a little complicated, it can be tiresome to create all the models, and the resulting model may be rather cluttered. Both issues are solved when using object-oriented Bayesian networks.

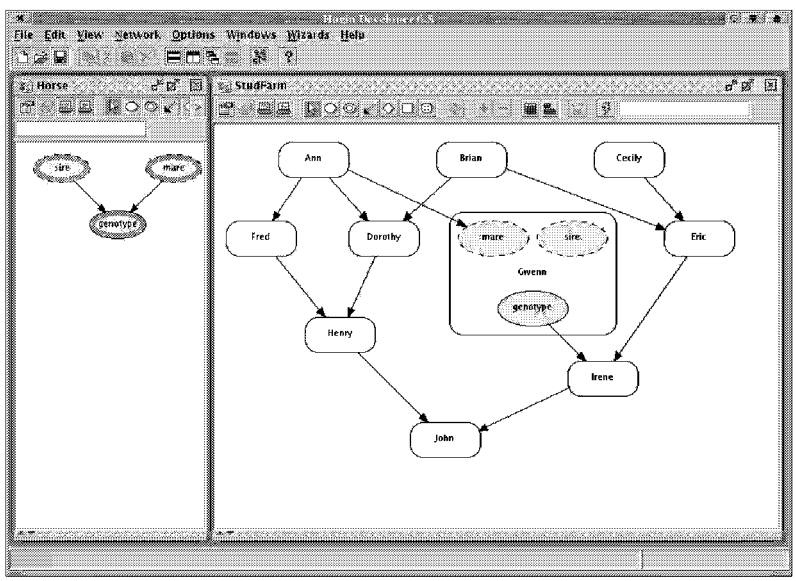


Fig. 5. The Stud Farm network.

Consider a stud farm with ten horses: Cecily has unknown mare and sire, John has mare Irene and sire Henry, Henry has mare Dorothy and sire Fred, Irene has mare Gwenn and sire Eric, Gwenn has mare Ann and unknown sire, Eric has mare

Cecily and sire Brian, Fred has mare Ann and unknown sire, Brian has unknown mare and sire, Dorothy has mare Ann and sire Brian, and Ann has unknown mare and sire.

Given that John is sick, the task is to compute the probability of each of the other horses being sick or carriers. The possible genotypes of each horse are aa, aA, and AA. A sick horse has genotype aa, a carrier of the disease has genotype aA, and a non-carrier has genotype AA. The prior distribution is $P(\mathsf{aa}, \mathsf{aA}, \mathsf{AA}) = (0.1, 0.2, 0.7)$.

Fig. 5 shows a model specifying the pedigree of the ten horses. Each horse is simply represented as an instance of the Horse network class shown in the left part of Fig. 5. Hence, the resulting model is quite simple to build and interpret.

Another feature of object-oriented Bayesian networks is modularity. If e.g., one of the horses is a special breed it can easily be represented using a different model, as long as the interfaces of the the object classes are identical (the interface of the Horse network class can be seen on the expanded node in Fig. 5).

### 2.5. *Table generator*

The specification of a conditional probability distribution $P(X \mid \pi(X))$ or a utility function $u(\pi(U))$ can be a labor intensive task when the cardinality of the parent set is large. As probability distributions and utility functions are represented as tables, we use the term table to refer to both.

In many cases, however, $P(X \mid \pi(X))$ or $u(\pi(U))$ are known to follow (at least approximately) certain functional or distributional forms. The Hugin Tool provides a powerful Table Generator mechanism for compact specifications of such conditional probability and utility tables. The Table Generator provides a large set of standard mathematical functions for specifying table models and expressions.

An expression is built using standard statistical distributions (e.g., Normal, Binomial, Beta, Gamma, etc.), arithmetic operators, standard mathematical functions (e.g., logarithmic, exponential, trigonometric, and hyperbolic functions), logical operators (e.g., and, or, if-then-else), and relations (e.g., less-than, equals).

The different operators used in an expression have different return types and different type requirements for their arguments. Thus, in order to provide a rich language for specifying expressions, it is convenient to have a classification of the discrete chance and decision variables into different groups of variables. The Hugin Decision Engine accepts four different types of discrete variables: *labelled*, *Boolean*, *numbered*, and *interval* variables.

The ability to specify expressions instead of entire conditional probability distributions and utility functions is powerful and can save a lot of time as shown by the examples given below.

**Example 2.6.** Consider the problem of computing the probability of getting $n$ sixes in $n$ rolls with a die, which is either fair or fake. A random variable, $X$, denoting the number of sixes obtained in $n$ rolls with a fair die is Binomially distributed with parameters $(n, 1/6)$. Thus, the probability of getting $k$ sixes in $n$ rolls with a fair

die is $P(X = k)$, where $P$ is Binomial$(n, 1/6)$. Assuming that for a fake die the probability of getting six eyes in one roll is $1/5$, the probability of getting $k$ sixes in $n$ rolls with a fake die is $Q(X = k)$, where $Q$ is Binomial$(n, 1/5)$.

A Bayesian-network model of this problem contains a node n6s with parents nRolls and fakeDie. Now, if we let nRolls be a numbered node with states $1, \ldots, n$, let fakeDie be a Boolean node, and let n6s be a numbered node with states $0, \ldots, n$, then $P(\text{n6s} \mid \text{nRolls}, \text{fakeDie})$ can be specified elegantly through the expression:

Binomial(nRolls, if (fakeDie, 1/5, 1/6)).

Table 4 shows the conditional probability table of n6s when $n = 5$.

Table 4. The conditional probability distribution $P(\text{n6s} \mid \text{nRolls}, \text{fakeDie})$.

| nRolls | 1 | | 2 | | 3 | | 4 | | 5 | |
|---|---|---|---|---|---|---|---|---|---|---|
| fakeDie | false | true | false | true | false | true | false | true | false | true |
| n6s | | | | | | | | | | |
| 0 | 0.8333 | 0.8 | 0.6944 | 0.64 | 0.5787 | 0.512 | 0.4823 | 0.4096 | 0.4019 | 0.3277 |
| 1 | 0.1667 | 0.2 | 0.2778 | 0.32 | 0.3472 | 0.384 | 0.3858 | 0.4096 | 0.4019 | 0.4096 |
| 2 | 0 | 0 | 0.0278 | 0.04 | 0.0694 | 0.096 | 0.1157 | 0.1536 | 0.1608 | 0.2048 |
| 3 | 0 | 0 | 0 | 0 | 0.0046 | 0.008 | 0.0154 | 0.0256 | 0.0322 | 0.0512 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0.0008 | 0.0016 | 0.0032 | 0.0064 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.0001 | 0.0003 |

**Example 2.7.** Assume that $P(X \mid Y)$ can be approximated by a Normal distribution with mean given by $Y$ and with variance one, where $Y$ is a numbered or an interval variable. Now, a suitable discretization of $X$ is determined (e.g., given by the intervals $]-\infty, -5]$, $]-5, -2]$, $]-2, 0]$, $]0, 2]$, $]2, 5]$, $]5, \infty[$) and an interval variable, say $X'$, is defined with states corresponding to the discretization intervals. Then we can specify $P(X' \mid Y)$ simply as Normal$(Y, 1)$ and the Hugin Tool will generate the table of corresponding conditional probability distribution.

**Example 2.8.** If, e.g., the interaction model between a variable and its three parents $X$, $Y$, and $Z$ can be modeled as a Noisy-OR interaction model, this can be specified as [1]:

NoisyOR (X, 0.3, Y, 0.5, Z, 0.2, true, 0.93).

This expression specifies a Noisy-OR model where the inhibitor probabilities are 0.3, 0.5, and 0.2 for $X$, $Y$, and $Z$, respectively, with a leak probability of 0.07.

The Table Generator contains a lot of other features, e.g., specification of probabilities using bars (and dragging) instead of numbers, exporting tables to various formats (e.g., LaTeX), auto normalization, etc.

## 3. Inference

In this section, we consider the task of performing probabilistic inference in Bayesian networks and mixed Bayesian networks in the light of evidence and solving an influence diagram representation of a decision problem.

The Hugin Tool supports both hard and soft evidence. Hard evidence is an instantiation of a discrete chance variable or an assignment of a value to a continuous variable. Soft evidence is a likelihood finding on a discrete chance variable. A set of evidence $\epsilon = \{\epsilon_1, \ldots, \epsilon_n\}$ consists of $n$ pieces of evidence.

### 3.1. *Bayesian networks*

For Bayesian networks we define probabilistic inference as the task of computing all posterior marginals of non-evidence variables given the evidence $\epsilon$, i.e., $P(X \mid \epsilon)$ for each non-evidence variable $X$.

Evidence $\epsilon_i$ — whether hard or soft — on a discrete variable $X$ can be specified using a finding potential $f(\epsilon_i)$ over $X$ assigning appropriate values reflecting the evidence to each state of $X$. Hence, the task of computing the posterior marginal distribution of $Y$ given evidence $\epsilon$ in a discrete Bayesian network is to compute:

$$P(Y \mid \epsilon) \propto \sum_{X \in V \setminus \{Y\}} \prod_{X \in V} P(X \mid \pi(X)) \prod_{\epsilon_i \in \epsilon} f(\epsilon_i), \tag{25}$$

where $f(\epsilon_i)$ is a finding potential enforcing $\epsilon_i$.

### 3.2. *Mixed Bayesian networks*

For mixed Bayesian networks we define probabilistic inference as the task of computing all posterior marginals of non-evidence variables given the evidence $\epsilon$. For each discrete non-evidence variable $X$, we compute $P(X \mid \epsilon)$. For each continuous non-evidence variable $Y$, the posterior marginal is a mixture of Gaussian distributions. The correct mean and variance of the mixture is computed for each $Y$ with the option of computing an exact representation of the mixture.

Evidence $\epsilon$ on a continuous variable $X$ is an instantiation of $X$ to a value $x$. Special care needs to be taken when inserting evidence on continuous variables and calculating posterior distributions given the evidence in a mixed Bayesian network. As mentioned above, the posterior of a discrete variable $X$ given the evidence $\epsilon$ has the form of a conditional probability distribution $P(X \mid \epsilon)$, whereas the posterior marginal of a continuous variable $Y$ given the evidence $\epsilon$ has the form of a Gaussian discrete mixture, i.e.,

$$\mathcal{L}(Y \mid \epsilon) = \sum_i P(i) * \mathbb{N}(A(i), C(i)), \tag{26}$$

where the mixture components and weights $P(i)$ are determined by the inference process. In principle, the marginal of a variable $Y$ is computed by marginalization of $\bigotimes_{X \in V} \phi(X \mid \pi(X))$ onto $Y$; see the literature for details.[13]

For networks containing CG variables, the normal output of a propagation consists of the mean and variance of each CG variable given the evidence. However, it is also possible to get an exact representation of the distribution of a CG variable. As seen above, this distribution is a mixture of Gaussian distributions, and an exact representation of the distribution contains a list of the parameters describing each component of the mixture.[13]

### 3.3. *Influence diagrams*

For influence diagrams we define inference as the task of determining an optimal strategy $\hat{\Delta} = \{\delta_1, \ldots, \delta_n\}$ consisting of exactly one policy for each decision variable of the influence diagram. The policy for decision variable $D$ prescribes what action the decision maker should take for $D$ given knowledge of previous observations and decisions, i.e. it is a function from (a subset of) past observations and decisions to current decision alternatives. A strategy is said to be optimal when it maximizes the expected utility.

The optimal strategy for each decision variable and the maximum expected utility of adhering to the optimal strategy can be determined by variable elimination (in the reverse order of observation):

$$\mathrm{MEU}(\hat{\Delta}) = \sum_{\mathcal{I}_0} \max_{D_1} \sum_{\mathcal{I}_1} \cdots \max_{D_n} \sum_{\mathcal{I}_n} \prod_{X \in V_C} P(X \mid \pi(X)) \sum_{U \in V_U} u(\pi(U)). \tag{27}$$

Notice, that once an influence diagram has been solved further inference is unnecessary, as an optimal strategy has been identified.

### 3.4. *Message passing*

For reasons of efficiency, inference is performed by message passing in a secondary computational structure. Hence, before inference can take place, the Hugin Decision Engine first compiles the Bayesian network or influence diagram model into a structure known as a (strong) junction tree. The nodes of the junction tree are sets of variables of the original network. These sets are sometimes called cliques because they correspond to cliques (maximal complete sets) in some triangulated graph obtained from the original network.

The complexity of the junction tree is determined by the complexity of the triangulation of the DAG of the model. The choice of triangulation method can have a huge impact on the size of the junction tree corresponding to the triangulated graph, where size is the total sum of the clique state spaces. The state space size of a clique $C$ is $\prod_{X \in C} ||X||$ where $||X||$ is the state space size of $X$.

The Hugin Tool supports five different triangulation methods *clique-size*, *clique-weight*, *fill-in-size*, *fill-in-weight*, and *total-weight*. The first four methods are heuristic methods based on node elimination whereas the fifth method is a combined heuristic/exact method capable of producing optimal triangulations based on graph

decomposition by minimal separators.[16,17,18] All methods are applied to the moral graph of the DAG of the model.

Once the junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ is constructed, a probability potential (and a utility potential for influence diagrams) is associated with each clique $C \in \mathcal{C}$ and each separator $S \in \mathcal{S}$ between two adjacent cliques $C_i$ and $C_j$ where $S = C_i \cap C_j$, see Fig. 6. A probability potential $\phi_W$ for a set of variables $W$ is a function from $W$ to the set of non-negative real values (a utility potential for a set of variables $W$ is a function from $W$ to the set of real values).

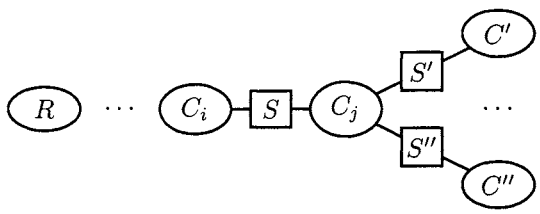


Fig. 6. When $C_j$ has absorbed evidence from its other neighbors, $C_i$ can absorb from $C_j$.

Inference in mixed Bayesian networks and influence diagrams can be considered as special cases of probabilistic inference in discrete Bayesian networks requiring special data types and operations. Hence, we focus on inference in discrete Bayesian networks and refer the reader to the literature on the inference process in the aforementioned types of models.

Inference is performed using a message passing algorithm on the junction tree. This process is known as propagation. Inference involves the following steps:

(i) Each item of evidence must be incorporated into the junction tree potentials. For each item of evidence, some potential, containing the variable in question, is modified to reflect the evidence.
(ii) Some clique $R \in \mathcal{C}$ of $\mathcal{T}$ is selected. This clique is referred to as the root of the propagation.
(iii) Then messages are passed through the separators of the junction tree (i.e., along the edges of the tree) towards the selected root. These messages cause the potentials of the receiving cliques and separators to be updated. This phase is known as CollectInformation.
(iv) Now messages are passed in the opposite direction (i.e., from the root towards the leaves of the junction tree). This phase is known as DistributeInformation.
(v) At this point, the junction tree is said to be in equilibrium: The probability $P(X \mid \epsilon)$ can be computed from any clique or separator containing $X$—the result will be independent of the chosen clique or separator.

Prior to the initial round of message passing, for each variable $X$ of the Bayesian

network $\mathcal{N}$ we assign the conditional probability distribution $P(X \mid \pi(X))$ to a clique $C$ such that $\{X\} \cup \pi(X) \subseteq C$. Once all conditional probability distributions have been assigned to cliques, the distributions assigned to each clique are combined to form the initial clique potential.

The basic inference algorithm is as follows. Each separator holds a single potential over the separator variables, which initially is a unity potential. During propagation of evidence the separator and clique potentials are updated. Consider two adjacent cliques $C_i$ and $C_j$ as shown in Fig. 6. When a message is passed from $C_j$ to $C_i$ either during COLLECTINFORMATION or DISTRIBUTEINFORMATION, $C_i$ absorbs information from $C_j$. Absorption of information involves performing the following calculations:

(i) Calculate the updated separator potential:

$$\phi_S^* = \sum_{C_j \setminus S} \phi_{C_j}. \tag{28}$$

(ii) Update the clique potential of $C_i$:

$$\phi_{C_i}^* \leftarrow \phi_{C_i} \frac{\phi_S^*}{\phi_S}. \tag{29}$$

(iii) Associate the updated potential with the separator:

$$\phi_S \leftarrow \phi_S^*. \tag{30}$$

After a full round of message passing the potential associated with any clique (separator) is the joint probability distribution (up to the same normalization constant) over the variables in the clique (separator) and the evidence. This algorithm is known as the Hugin algorithm. Details on the inference process can be found in the literature.[11,6,19,20,21,13]

As mentioned above, inference in a mixed Bayesian network and solution of an influence diagram require special data types and operations.

Inference in mixed Bayesian networks is performed using an algorithm, which is a combination of the Hugin and the Lauritzen-Spiegelhalter algorithms working on a strong junction tree.[11,19,13] The main issue is that a strong triangulation is required as the conditional Gaussian distribution is not closed under marginalization.[13]

An influence diagram is solved using an algorithm, which is an extension of the Hugin algorithm to cope with decision variables and utility functions performing only a COLLECTINFORMATION on the root of the strong junction tree.[21] Introducing decision variables into the algorithm requires that the message passing is performed in a strong junction tree (unless an algorithm based on the LImited Memory Influence Diagram (LIMID) representation is used.[22,23])

**Example 3.1.** Fig. 7 shows the posterior marginal of each variable of the Chest Clinic example with observations on Dyspnoea (yes), Positive X-ray result (no), and Smoker (yes).

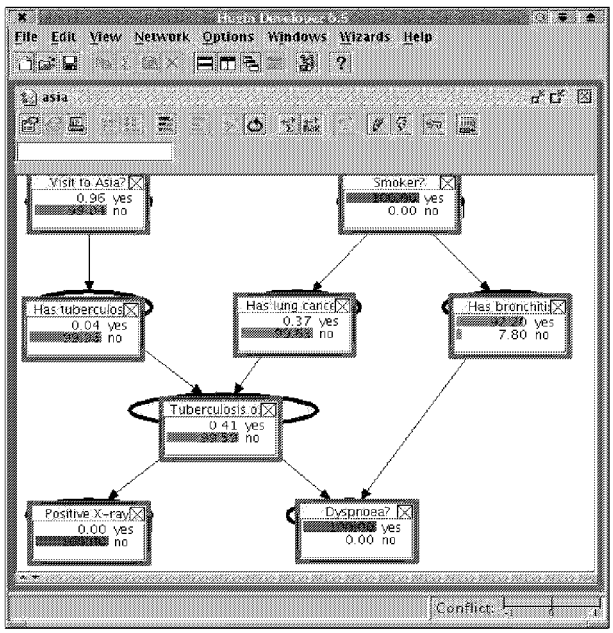


Fig. 7. The posterior marginals given evidence $\epsilon = \{D = \text{yes}, S = \text{yes}, X = \text{no}\}$ of the Chest Clinic example.

Once evidence has been incorporated, the Bayesian network represents the joint distribution over its variables and the evidence.

**Example 3.2.** Fig. 8 shows the prior marginal of each variable of the Waste Incinerator example.

### 3.5. *Common inference tasks and special features*

The junction tree structure is also suitable for many other inference tasks:

(i) The probability of the evidence $P(\epsilon)$ is computed as part of a propagation. It is the normalization constant of the probability potential of the root after CollectInformation has been completed.

(ii) The most probable explanation is the configuration of all variables that has the highest probability given the evidence. This configuration can be identified by a so-called max-propagation, which differs from a normal propagation only by using maximization instead of summation.[20]

(iii) The joint probability distribution over a set of variables can be computed by combining potentials in a subtree of the junction tree containing all the variables in question.

(iv) A variant propagation algorithm, known as fast retraction, computes, for each

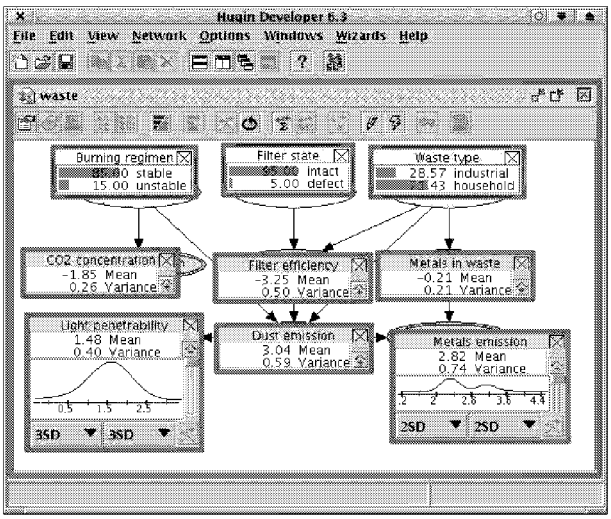


Fig. 8. The marginals of the Waste Incinerator example.

evidence variable $X$, the conditional probability given the evidence on all variables except $X$.[24] Fast retraction is useful for identifying spurious findings.

In addition to the aforementioned operations, the Hugin Decision Engine provides the user with the following special features:

(i) An approximation method that changes very small probabilities to zeros is supported. Zeros in the probability potentials allow the potentials to be represented using sparse tables. This can result in huge speed-ups.[25]

(ii) Parallel processing for inference on multi-processor systems is supported. The parallel processing is implemented through parallel table operations.[26]

(iii) Sampling using either the Bayesian network or the junction tree structure is supported. Sampling of discrete, CG, and mixtures of CG and discrete variables is supported. A configuration of the variables can be sampled according to the distribution determined by the evidence. Sampling in an uncompiled Bayesian network is possible when the set of nodes with evidence form an ancestral set of instantiated nodes. Similarly, sampling in influence diagrams is supported when all decisions are instantiated.

(iv) Computations are usually performed using a single precision floating point representation, but double precision is also an option. Double precision is useful for mixed Bayesian networks and when performing sequential learning.

(v) Special *save-to-memory* functionality is provided to support efficient initialization (i.e., establishing the initial distribution, with no evidence incorporated, on the junction tree).

## 4. Data Conflict Analysis

Data conflict analysis is the activity of detecting, tracing, and explaining possible conflicts among observations of variable values (i.e., evidence or data) given a discrete Bayesian network model $\mathcal{N}$. Inconsistencies according to $\mathcal{N}$ among the elements of $\epsilon$ are easily detected as $P(\epsilon) = 0$, but also flawed findings should be detected and traced. For example, in a diagnostic situation a single flawed test result may take the investigation in a completely wrong direction.

The Hugin Tool implements the data conflict analysis method based on evaluating observations relative to a pure independence model.[27]

Given the pure independence model, a conflict measure can be defined as:

$$\text{conf}(\epsilon) = \log\left(\frac{\prod_{i=1}^{n} P(\epsilon_i)}{P(\epsilon)}\right). \tag{31}$$

If $\text{conf}(\epsilon) > 0$, then this is an indication of a possible conflict as we assume individual pieces of evidence to be positively correlated under our current model. The conflict is easily computed as part of a propagation.[27]

Notice that a positive conflict measure indicates negatively correlated observations (i.e., a possible conflict) and that a negative conflict measure indicates positively correlated observations.

Once an indication of a possible conflict in $\epsilon$ has been detected, the source of the conflict should be traced. The conflict measure associated with each clique of the junction tree enables the user to identify possible sources of the conflict. For each clique $C$, the conflict measure is computed on the evidence inserted in the subtree of the junction tree rooted at $C$ relative to the root of the tree.

Typical data from a very rare case of evidence $\epsilon$ (according to the model) may indicate a possible conflict, i.e. $\text{conf}(\epsilon) > 0$. To identify a possible hypothesis, which could explain away the conflict, we compare $P(h)$ and $P(h \mid \epsilon)$. If the change in probability is sufficient to eliminate the conflict (i.e. $\text{conf}(\epsilon \cup \{h\}) \leq 0$), then $h$ is presented to the user as a possible explanation of the conflict. The change is sufficient, if the logarithm of the normalized likelihood is larger than the conflict, i.e. $\text{conf}(\epsilon) \leq \log \frac{P(h \mid \epsilon)}{P(h)}$.

**Example 4.1.** Consider the case of a smoking patient with a positive X-ray result and no shortness-of-breath in the Chest Clinic example, i.e. $\epsilon = \{S = \text{yes}, D = \text{no}, X = \text{yes}\}$, see Fig. 9. In this case the value of the conflict measure is $\text{conf}(\epsilon) = 0.425$, which is an indication of a possible conflict. From our knowledge of the model, we would argue that the observations that the patient has a positive X-ray result, but no shortness-of-breath are conflicting (indeed $\text{conf}(\{X = \text{yes}\}, \{D = \text{no}\}) = 0.451$).

From our knowledge of the model, a possible explanation of the conflict could be that the patient has tuberculosis, i.e. $T = \text{yes}$. The value of the conflict measure with hypothesis $h_{T=\text{yes}}$ is $\text{conf}(\epsilon \cup h_{T=\text{yes}}) = -1.04$. Hence, the hypothesis $h_{T=\text{yes}}$ may resolve the conflict as a rare case. Notice the change in belief of $T = \text{yes}$: the prior

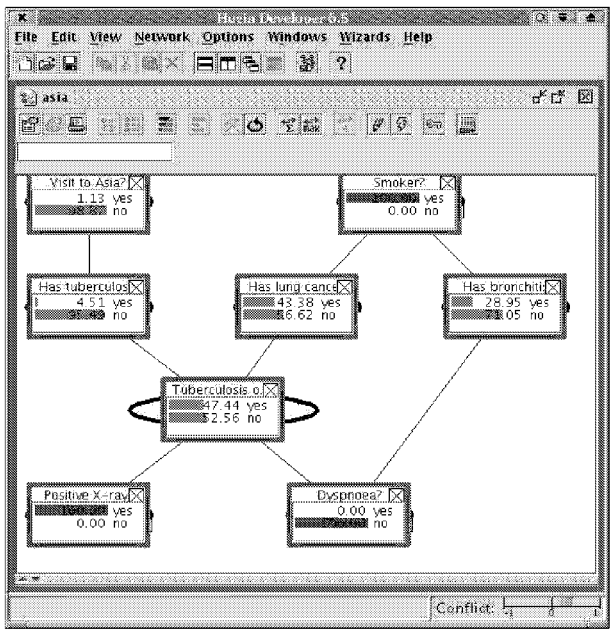


Fig. 9. The evidence $\epsilon = \{S = \text{yes}, D = \text{no}, X = \text{yes}\}$ produces a positive conflict measure indicating a possible conflict in $\epsilon$.

is $P(T = \text{yes}) = 0.01$ while the posterior is $P(T = \text{yes} \mid \epsilon) = 0.0451$. Similarly, $h_{L=\text{yes}}$ and $h_{E=\text{yes}}$ are hypotheses, which may explain away the conflict as a rare case. The Hugin Tool will present all three hypotheses to the user for her inspection.

Consider a scenario of a non-smoking patient with shortness-of-breath and a positive X-ray result who has recently been to Asia, i.e. $\epsilon = \{S = \text{no}, D = \text{yes}, X = \text{yes}, A = \text{yes}\}$.

Even though there is an indication of local conflict of $\text{conf}(\{D = \text{yes}\}, \{S = \text{no}\}) = 0.312$ at clique $BEL$, the global conflict measure is $\text{conf}(\epsilon) = \text{conf}(\{D = \text{yes}\}, \{S = \text{no}\}) + \text{conf}(\{D = \text{no}, S = \text{yes}\}, \{X = \text{yes}\}) = -0.203$ indicating no conflict in the entire set of evidence, see Fig. 10.

## 5. Learning a Bayesian Network

In this section, we consider the task of inducing a Bayesian network by fusion of a database of cases and domain expert knowledge. The DAG $G$ of a Bayesian network $\mathcal{N} = (G, \mathcal{P})$ induces a set of (conditional) dependence and independence relations (CIDRs) $\mathcal{M}_G$, which can be read off $G$ using the $d$-separation criterion.[9,10] Two DAGs representing the same set of CIDRs are equivalent in the sense that they can capture the same set of probability distributions. The equivalence class of a DAG $G$ is the set of DAGs with the same set of $d$-separation relations as $G$. A

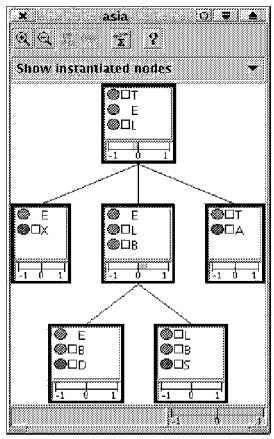


Fig. 10. Shows the local conflict at each clique.

PDAG — an acyclic, partially directed graph, i.e. an acyclic graph with some edges undirected (also known as a pattern or essential graph) — can be used to represent the equivalence class of a set of DAG structures, i.e. a maximal set of DAGs with the same set of $d$-separation relations.[28] Two DAGs $G_1$ and $G_2$ are equivalent if they have the same skeleton (i.e. undirected graph obtained by replacing directed edges with undirected edges) and the same set of colliders (i.e. $X \rightarrow Y \leftarrow Z$-structures).[28]

In the remainder of this section, we will assume that $P_0$ is a DAG faithful probability distribution with underlying DAG $G_0$. The faithfulness assumption (also known as stability) says that the distribution $P$ over $V$ induced by $(G, \mathcal{P})$ satisfies no independence relations beyond those implied by the structure of $G$.[29,28] We consider learning a Bayesian network as the task of identifying a DAG structure $G$ and a set of conditional probability distributions $\mathcal{P}$ with parameters $\Theta$ from a sample of independent and identically distributed data cases $D = \{c^1, \ldots, c^N\}$ drawn at random from $P_0$ and possibly some domain expert background knowledge. Each case $c^i = \{x_1^i, \ldots, x_n^i\}$ contains an assignment of a value $x_j^i$ to each variable $X_j$ of $\mathcal{N}$. Some values may be missing, but missing values are assumed to be missing-at-random or missing-completely-at-random (i.e., the missing data mechanism is uninformative and can be ignored).

Based on data alone, we can at most hope to identify a PDAG representing the equivalence class of the generating distribution $P_0$.

### 5.1. *Structure learning*

Structure learning in the Hugin Tool is supported through a constraint-based approach.[30,31,29] In the constraint-based approach, the graph $G$ of a Bayesian network $\mathcal{N}$ is considered as an encoding of a set of CIDRs $\mathcal{M}$. Structure learning is

then the task of identifying a DAG structure from a set of CIDRs derived from the data by statistical tests.

Two algorithms for structure learning are supported. The PC algorithm (which is similar to the IC algorithm) and its extension, the NPC algorithm.[32,29,31,28,33] The main steps of the PC algorithm are:

(i) Test for (conditional) independence between each pair of variables.
(ii) Identify the skeleton of the graph induced by the derived CIDRs.
(iii) Identify colliders.
(iv) Identify derived directions.

The PC algorithm produces a PDAG. In step i, the hypothesis is that $X$ and $Y$ are independent given $S_{XY}$. This hypothesis is tested by statistical tests using conditioning sets $S_{XY}$ of cardinality $0, 1, 2, 3$. If the hypothesis $X \perp\!\!\!\perp Y \mid S_{XY}$ cannot be rejected based on some significance level $\alpha$, then the search for an independence relation between $X$ and $Y$ is terminated.

Various heuristic speed-up improvements of the straightforward incremental testing scheme have been implemented.[29] These improvements are related to maintaining an undirected graph describing the current set of neighbors of each node and only performing independence tests conditional on subsets of the neighbors of $X$ and $Y$. This makes the algorithm a variant of the PC* algorithm.[29] The order in which we try out the possible conditioning sets of a fixed cardinality is according to how likely they are to cause independence for the edge under consideration. We use the heuristic rule that the variables of the conditioning set should be strongly correlated with both endpoints of the edge being tested. The neighbor graph is updated after each independence test not rejecting the hypothesis.

Due to the nature of the testing scheme, the conditioning set $S_{XY}$ for an identified independence relation $X \perp\!\!\!\perp Y \mid S_{XY}$ is minimal in the sense that no proper subset of $S_{XY}$ makes $X$ and $Y$ independent. An undirected edge is added between each pair of variables $X, Y$ whenever no conditional independence relation has been identified between $X$ and $Y$. This produces the skeleton of the graph.

Once the skeleton has been identified, colliders are identified. If $X$ and $Y$ are neighbors, $Z$ and $Y$ are neighbors, $X$ and $Z$ are not neighbors, and $Y \notin S_{XZ}$ for any $S_{XZ}$ satisfying $X \perp\!\!\!\perp Z \mid S_{XZ}$, then a collider is created at $Y$.

Starting with any PDAG $G$, a maximally directed PDAG can be obtained following four necessary and sufficient rules, see Fig. 11.[31,34] That is, by repeated application of these four rules all edges common to the equivalence class of $G$ are identified. The fourth rule is unnecessary, if the orientation of the initial PDAG is limited to colliders (i.e. no background knowledge). The four rules are necessary and sufficient for achieving maximal orientation (up to equivalence) of the PDAG returned by the PC algorithm. The first rule follows from the fact that no collider was identified, the remaining rules ensure that no directed cycle is created. The dashed lines used to illustrate the fourth rule indicate that $X$ and $V$ are connected by an edge.

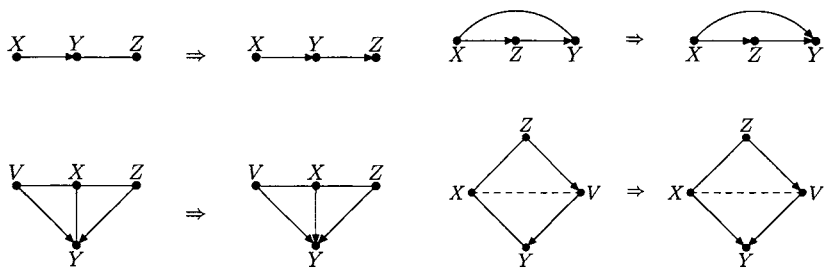


Fig. 11. Four rules for orientation of edges.

Correctness of the PC algorithm has been proved under the assumption of infinite data sets. In real-life, data sets are finite. When dealing with finite data sets, the faithfulness assumption is often violated. Hence, when the derived set of CIDRs is induced by statistical tests on finite data sets we cannot — in general — expect a DAG (or PDAG) representing all CIDRs to exist. Often too many conditional independence relations are derived due to the limited data set. This suggests to represent all conditional dependence relations, but not all conditional independence relations in the induced DAGs. Applying the principle of Occam's Razor, we choose the simplest model among equally good models.

As mentioned above, the NPC algorithm is an extension of the PC algorithm. The new feature of the NPC learning algorithm is the introduction of the notion of a Necessary Path Condition.[33] Informally, the necessary path condition says that in order for two variables $X$ and $Y$ to be independent (in a DAG faithful data set) conditional on a set $S_{XY}$ and no subset $S' \subset S_{XY}$, there must exist a path between $X$ and every $Z \in S$ (not crossing $Y$) and between $Y$ and every $Z \in S_{XY}$ (not crossing $X$). Otherwise, the inclusion of each $Z$ in $S_{XY}$ is unexplained. Thus, in order for an independence relation to be valid, a number of edges (or paths) are required to be present in the graph.

An edge $(X, Y)$ is an ambiguous edge, if the absence of $(X, Y)$ depends on the presence of an edge $(X', Y')$, and vice versa. A maximal set of interdependent ambiguous edges is an ambiguous region. An ambiguous region indicates inconsistency in the set of independence relations derived by the statistical tests.

In order to increase reliability and stability of the NPC algorithm, the iteration step for a fixed cardinality of the conditioning set is completed even if an independence statement is found. Thus, multiple independence relations may be found for each pair of variables. If one of these independence relations satisfy the necessary path condition, then the independence hypothesis is not rejected. Otherwise, an ambiguous edge is created. Once all ambiguous regions are identified, each ambiguous region is simplified to include only a minimal number of edges. An ambiguous region is resolved by including a minimal number of ambiguous edges in order to

satisfy a maximal number of independence relations.

Prior to the testing phase, background knowledge in the form of constraints on the structure of the DAG can be specified. It is possible to specify the presence and absence of edges, the orientation of edges, and a combination. At the moment, background knowledge is assumed to be consistent with the underlying DAG $G_0$ of the generating distribution $P_0$. Therefore, user specified constraints are not tested. In practice, this has produced some unwanted behavior of the edge orientation algorithm. This implies that background knowledge should be used with caution.

**Example 5.1.** Fig. 12 shows the PDAG generated by applying the NPC algorithm a random sample of 10,000 cases independently drawn from the Chest Clinic network using a significance level of $\alpha = 0.05$.

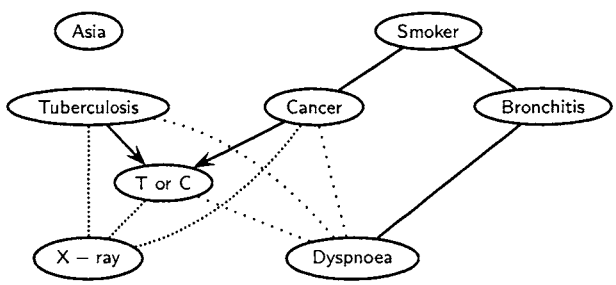


Fig. 12 The result of NPC learning on data generated from the Chest Clinic example.

The sets of edges $\{(T,X),(E,X),(L,X)\}$ and $\{(T,D),(E,D),(L,D)\}$ are the two ambiguous regions of Fig. 12. The two ambiguous regions are due to the deterministic relation between $E$ and $L,T$ (i.e. $E = L \vee T$). This produces according to the necessary path condition two sets of inconsistent independence statements $\{(X \perp E \,|\, T,L),(X \perp T \,|\, E),(X \perp L \,|\, E)\}$ and $\{(D \perp E \,|\, T,L),(D \perp T \,|\, E),(D \perp L \,|\, E)\}$. The simplest resolution of each ambiguous region is to include edge $(E,X)$ and edge $(E,D)$, respectively. Notice that some certain edges are directed and some are undirected.

Based on domain expert knowledge, the user will resolve the ambiguous regions by selecting which edges to include and exclude. The user will also select an appropriate orientation of each undirected edge in the graph to obtain a DAG.

### 5.2. *Batch parameter learning*

Parameter estimation in a Bayesian network $\mathcal{N} = (G,\mathcal{P})$ is the task of estimating the values of parameters $\Theta$ corresponding to DAG structure $G$ and distributions $\mathcal{P}$ from a database of cases $D = \{c^1,\ldots,c^N\}$. Parameter estimation in the Hugin Tool is supported through the Expectation-Maximization (EM) algorithm.[35] The EM

algorithm is well-suited for calculating maximum likelihood (ML) and maximum a posteriori (MAP) estimates in the case of missing data. The EM algorithm proceeds by iterating two steps: the E-step and the M-step.

Let $\mathcal{N} = (G, \mathcal{P})$ be a Bayesian network with parameters $\Theta = \{\Theta_i\}$ where $\Theta_i = \{\Theta_{ij}\}$ and $\Theta_{ij} = \{\theta_{ijk}\}$ such that $\theta_{ijk} = P(X_i = k \mid \pi(X_i) = j)$ for each $i, j, k$. The EM algorithm is based on computing the expected value of the log-likelihood function:

$$Q(\Theta^* \mid \Theta) = \mathbb{E}_\Theta\{\log P(X \mid \Theta^*) \mid D\}, \tag{32}$$

where $P$ is the density function for $X$, and $D$ is the observed data $D = g(X)$.[35] Given an initial assignment of values to the parameters $\Theta$, the E-step is to compute the current expected value of $Q$ with respect to $\Theta$, while the subsequent M-step is to maximize $Q$ in $\Theta^*$. These two steps are alternated iteratively until a stopping criterion is satisfied.

In the case of missing data, the log-likelihood function is a linear function in the sufficient marginals.[35] The log-likelihood function $l(\Theta)$ of the parameters $\Theta$ given the data $D$ and DAG $G$ is:

$$l(\Theta) = \sum_{l=1}^{N} \log P(c^l \mid \Theta) \tag{33}$$

$$= \sum_{l=1}^{N} \sum_{i=1}^{|V|} \log P(X_i = x_i^l \mid \pi(X_i) = x_{\pi(X_i)}^l, \Theta_i, c^l) \tag{34}$$

$$= \sum_{i=1}^{|V|} l(\Theta_i), \tag{35}$$

where $l(\Theta_i) = \sum_{l=1}^{N} \log P(X_i = x_i^l \mid \pi(X_i) = x_{\pi(X_i)}^l, \Theta_i, c^l)$ assuming the parameters $\Theta_i$ to be independent and $x_i^l$, $x_{\pi(X_i)}^l$ are the values of $X_i$, $\pi(X_i)$ in the $l$th (possibly incomplete) case of $D$.

For Bayesian networks, the E-step of the EM algorithm is to compute expected counts (expected sufficient statistics for a complete database), where expectation is taken with respect to the joint distribution over $V$ under the current parameter values $\Theta$ and observed data $D$:

$$\mathbb{E}_\Theta(N_{ijk}) = \sum_{l=1}^{N} P(X_i = k, \pi(X_i) = j \mid c^l, \Theta_i, G), \tag{36}$$

where $N_{ijk}$ is the count for $(X_i, \pi(X_i)) = (k, j)$ and $c^l$ is the $l$th case of $D$. Next, the M-step computes new estimates $\theta_{ijk}^*$ of $\theta_{ijk}$ interpreting the expected sufficient statistics as actual sufficient statistics from a complete database of cases:

$$\theta_{ijk}^* = \frac{\mathbb{E}_\Theta(N_{ijk})}{\sum_{k=1}^{||X_i||} \mathbb{E}_\Theta(N_{ijk})}. \tag{37}$$

The E-step and M-step are iterated until convergence of $l(\Theta)$ (or until a limit on the number of iterations is reached). In the Hugin Tool convergence is achieved when the difference between the log-likelihoods of two consecutive iterations is less than or equal to the numerical value of a log-likelihood threshold times the log-likelihood. Alternatively, the user can specify an upper limit on the number of iterations to ensure that the procedure terminates.

When both data and domain expert knowledge are available, these two sources of knowledge can be fused. We specify domain expert knowledge in the form of experience. Experience is the quantitative knowledge related to a probability distribution based on quantitative expert knowledge.[36] Expert knowledge on the parameters is specified as Dirichlet distributions. For each variable $X_i$, the distribution $P(X_i \mid \pi(X_i)) = \{P(X_i = k \mid \pi(X_i) = j)\}$ and the experience counts $\alpha_{i1}, \ldots, \alpha_{im}$ where $m = \prod_{Y \in \pi(X_i)} ||Y||$ associated with $X_i$ are used to specify the prior expert knowledge. Hence, the experience table of a variable $X_i$ indicates the experience related to the child distribution for each configuration of the parents. In the case of expert knowledge, the E-step does not change whereas the M-step becomes:

$$\theta^*_{ijk} = \frac{\alpha_{ijk} + \mathbb{E}_\Theta(N_{ijk})}{\sum_{k=1}^{||X_i||}(\alpha_{ijk} + \mathbb{E}_\Theta(N_{ijk}))}, \tag{38}$$

where $\alpha_{ijk} = P(X_i = k \,|\, \pi(X_i) = j)\alpha_{ij}$ is the initial count for $(X_i, \pi(X_i)) = (k, j)$.

The quality of the model is expressed in the value of $l(\Theta)$ computed after each iteration. It should be noticed that $l(\Theta)$ as a quality measure does not incorporate the complexity of the model. For comparison of models with different complexity other measures such as BIC or AIC could be used.

As described above, the experience counts for the prior beliefs in the conditional probability distribution of variable $X_i$ given its parents $\pi(X_i)$ are specified in a separate table including one experience count for each configuration of $\pi(X_i)$. After the termination of the EM algorithm, the expected counts are stored as the experience counts. If a variable $X$ does not have an experience count table before invoking EM, then the conditional probability distribution of $X$ is left unaltered. Similarly, as can be seen from Eq. (38), if an entry $\alpha_{ij}$ of the experience count table is zero, then an ML estimation is performed. Otherwise, a MAP estimation is performed.

The E-step is performed using probabilistic inference. Each case is inserted and propagated as evidence. Separate tables are used to store and accumulate the expected counts computed for each case. Once all cases have been processed, the M-step is performed. If a stopping criterion is met, the algorithm terminates. Otherwise, a new iteration through all cases is performed. As the efficiency of the EM algorithm relies heavily on the efficiency of probabilistic inference, performance of probabilistic inference is important. In the following section, we report on the efficiency of inference using the Hugin Tool.

**Example 5.2.** Assume that the qualitative knowledge of the Chest Clinic example is as shown in Fig. 1 and that a database of cases $D = \{c^1, \ldots, c^N\}$ with $N = 10,000$

is available. From the qualitative knowledge, we know that $E = L \vee T$. This is specified in $P(E \mid L, T)$ and no experience table is allocated to $E$ in order to avoid estimation of this table from the data whereas all other variables have an experience count table consisting of zeros indicating no expert knowledge on the distributions.

The EM algorithm will produce a maximum likelihood estimate of the parameters of the model under the constraint that $P(E \mid L, T)$ encodes disjunction. If we had expert knowledge on $P(S)$, for instance, we would specify this in $P(S)$ and encode the second order uncertainty in the experience table of $S$.

Table 5. The distribution $P(S)$ and its experience count.

| $S$ | |
|---|---|
| yes | 0.50486 |
| no | 0.49514 |
| Experience | 10000 |

Table 5 shows the resulting probability distribution $P(S)$ and corresponding experience counts for this table.

### 5.3. *Sequential parameter updating*

Sequential parameter updating or parameter adaptation is the task of sequentially updating the conditional probability distributions of a Bayesian network when the structure and an initial specification of the conditional probability distributions are given in advance. Sequential parameter updating in the Hugin Tool is supported through a Bayesian approach.[36,2] In sequential learning, experience is extended to include both quantitative expert knowledge and past cases (e.g. from EM learning). Thus, the result of EM learning could be used as the input for sequential learning.

Let $X_i$ be a variable with $n$ states, then the prior belief in the parameter vector $\Theta_{ij} = (\theta_{ij1}, \ldots, \theta_{ijn})$, i.e. the conditional probability distribution of a variable $X_i$ given its parents $\pi(X_i) = j$, is specified as an $n$-dimensional Dirichlet distribution $\mathcal{D}(\alpha_{ij1}, \ldots, \alpha_{ijn})$. This distribution is represented using a single experience count $\alpha_{ij}$ (equivalent sample size) and the initial content of $P(X_i \mid \pi(X_i) = j)$. The experience count $\alpha_{ijk}$ for a particular state $k$ of $X_i$ given $\pi(X_i) = j$ is $\alpha_{ijk} = \alpha_{ij} P(X_i = k \mid \pi(X_i) = j)$. This setting is similar to the setting of the EM algorithm.

Parameter adaptation proceeds by updating the experience associated with the parameters and subsequently updating the parameters to reflect the new experience. The process of updating the experience associated with a distribution is referred to as retrieval of experience. Dissemination of experience is the process of calculating prior conditional probability distributions for the variables in the Bayesian network given the experience, and it proceeds by setting the value of each parameter equal to the mean of the corresponding updated Dirichlet distribution, i.e. $\theta^*_{ijk}$, as shown

below. See Fig. 13 for a graphical representation of dissemination and retrieval of experience.

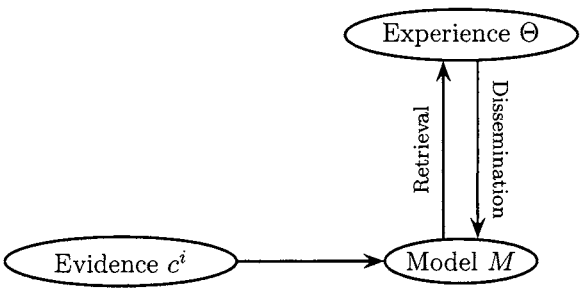


Fig. 13. Retrieval and dissemination of experience.

After a complete observation $(X_i, \pi(X_i)) = (k, j)$, the posterior belief in the distribution is updated as $\alpha^*_{ijk} = \alpha_{ijk} + 1$ and $\alpha^*_{ijl} = \alpha_{ijl}$ for $l \neq k$. After an incomplete observation, the weighted sum of Dirichlet distributions is approximated with a single Dirichlet distribution with the same means and sum of variances as the mixture. The approximation is used in order to avoid the combinatorial explosion, which would otherwise occur when subsequent incomplete observations are made. For each $i, j, k$, the updated value $\theta^*_{ijk}$ of each parameter $\theta_{ijk}$ is:

$$\theta^*_{ijk} = \frac{\alpha_{ijk} + P(X_i = k, \pi(X_i) = j \mid \epsilon) + \theta_{ijk}(1 - P(\pi(X_i) = j \mid \epsilon))}{\alpha_{ij} + 1}. \quad (39)$$

The updated $\alpha^*_{ij}$ is a function of the means and the sum of variances. Notice, that $P(X_i = k, \pi(X_i) = j \mid \epsilon)$ and $P(\pi(X_i) = j \mid \epsilon))$ are readily available after a propagation of evidence.

In order to reduce the influence of past and possibly outdated information, an optional feature of fading is provided. Fading proceeds by reducing the experience count before the retrieval of experience takes place. The experience count $\alpha_{ij}$ is faded by a factor of $0 < \lambda_{ij} \leq 1$ typically close to one according to $p_{ij} = P(\pi(X_i) = j)$ such that $\alpha^*_{ij} = \alpha_{ij}((1 - p_{ij}) + \lambda_{ij} p_{ij})$. Notice, that experience counts corresponding to parent configurations, which are inconsistent with the evidence are unchanged. The fading factors of a variable $X_i$ are specified in a separate table including one fading factor for each configuration of $\pi(X_i)$.

The Hugin Tool supports sequential updating of the parameters of conditional probability distributions in mixed Bayesian networks and in influence diagrams when all decision have been instantiated.

**Example 5.3.** Assume we have evidence $\epsilon = \{S = \mathsf{no}, A = \mathsf{yes}, D = \mathsf{yes}\}$ on a patient, i.e. a non-smoking patient with dyspnoea who has recently been to Asia.

The evidence is entered and propagated followed by an adaptation of parameters. Table 6 shows the experience counts for $L$, $B$, and $S$ before (i.e. after EM learning using 10,000 randomly generated cases) and after the adaptation with fading factor of 0.999 for each distribution.

Notice, that since $S$ is an observed variable without parents, the experience count $\alpha_S$ for $P(S)$ will converge to $\frac{1}{\lambda} = 1001$ if $S = \text{no}$ is observed multiple times.

Table 6. Experience counts for $B$, $L$, and $S$ before and after adaptation.

| | $\alpha_S$ | $\alpha_{L\|S=\text{no}}$ | $\alpha_{L\|S=\text{yes}}$ | $\alpha_{B\|S=\text{no}}$ | $\alpha_{B\|S=\text{yes}}$ |
|---|---|---|---|---|---|
| Before | 10,000 | 4970.88 | 5029.12 | 4970.88 | 5029.12 |
| After | 9,001 | 4472.71 | 5029.12 | 4473.73 | 5029.12 |

### 5.4. *Learning wizard and EM learning wizard*

Structure and parameter learning of a Bayesian network is in the Hugin Graphical User Interface supported through two wizards: the Learning Wizard and the EM Learning Wizard. The wizards guide the user through the necessary steps of learning a Bayesian network from a database of cases and domain expert knowledge.

A full cycle of learning a Bayesian network, as performed by the Learning Wizard, consists of three main steps: Data acquisition, structure learning, and parameter estimation. Each of these steps consists of a number of sub-steps guiding the user in the process of learning a Bayesian network from data and expert knowledge.

The functionality of the EM Learning Wizard is similar to that of the Learning Wizard except that the EM Learning Wizard is dedicated to parameter learning.

The data acquisition step serves two purposes: Read data from a data source and preprocess the data. In the first step, the user can read in data from various data sources, including databases (e.g. through an ODBC interface) and data files. In the second step, the user can preprocess the data, e.g. discretize a variable or select to use only a subset of the variables. It is also possible for the user to use his own preprocessor (must be implemented in the Java programming language though), if the existing preprocessor does not suffice.

The structure learning step contains two sub-steps: Structure learning and data analysis. In the structure learning phase, the user needs to choose the algorithm for learning the structure, i.e. either the PC or NPC algorithm. Common for both algorithms is the specification of a significance-level parameter and constraints on the structure of the DAG before the learning takes place. The constraints on the structure provide the user with a way to force known dependence and independence relations onto the learning algorithm. As it can be a tiresome task to specify these constraints for complex networks, the wizard facilitates the saving and loading of network information, including constraints, node positions, node labels, etc.

When the NPC algorithm is used for learning the structure, the user has the

possibility of resolving ambiguous regions and specifying the directions of undirected edges, see e.g. Fig. 12.

In the data analysis phase, the strength of marginal dependences between pairs of variables and the complexity of the learned network is indicated.

The parameter learning phase allows the user to specify initial values for the parameters of the Bayesian network and the parameters of the EM algorithm. The parameters of the Bayesian network may be assigned a random value in order to escape from a local maximum.

## 6. Performance Analysis

In this section, we report on the results of a performance analysis on the Hugin Tool. We focus on the performance of triangulation, probabilistic inference in discrete Bayesian networks, and learning discrete Bayesian networks from data. The solution of an influence diagram and inference in a mixed Bayesian network are both similar to probabilistic inference in a discrete Bayesian network. The difference is in the construction of the junction tree representation.

### 6.1. *Probabilistic inference*

Probabilistic inference in a Bayesian network is defined as the task of computing all posterior marginals given a subset of evidence. Here, the average run-times in seconds of inserting and propagating one state of one variable and computing all posterior marginals are reported (this corresponds to the average run-time of performing a DISTRIBUTEINFORMATION). Eleven real-world Bayesian networks have been used in the analysis.

The performance of inference using the Hugin algorithm is dominated by the size of the largest clique of the junction tree in which inference is performed. Tables 7 and 8 show the results of the analysis of inference performance using best known triangulations (some of which are optimal) found by the Hugin Decision Engine. The tables also show the previously best known triangulations, which in some cases is an order of magnitude worse than the optimal triangulation reported here.

Table 7. Information on optimal triangulations and inference times in seconds for four real-world Bayesian networks.

| | #nodes | optimal | previously best known | inference time |
|---|---|---|---|---|
| Barley | 48 | 17,140,796 | 17,338,949 | 0.96 |
| Mildew | 35 | 3,400,464 | 4,179,856 | 0.14 |
| Water | 32 | 3,028,305 | 8,035,356 | 0.20 |
| Ship-ship | 50 | 24,258,572 | 217,904,474 | 1.05 |

The optimality criterion for a triangulation is the total clique state space size of a corresponding junction tree $T = (\mathcal{C}, \mathcal{S})$, i.e. we try to minimize the score $\sum_{C \in \mathcal{C}} \prod_{X \in C} ||X||$.

Table 8. Information on best known triangulations and inference times in seconds for seven real-world Bayesian networks.

| | #nodes | best known | triangulation time | previously best known | inference time |
|---|---|---|---|---|---|
| Diabetes | 413 | 9,825,960 | 25 | 10,415,407 | 0.38 |
| Link | 724 | 23,983,962 | 116 | 26,179,418 | 1.43 |
| Munin1 | 189 | 83,735,758 | 41 | 188,387,156 | 6.17 |
| Munin2 | 1003 | 2,049,942 | 325 | 2,762,938 | 0.10 |
| Munin3 | 1044 | 3,077,688 | 29 | 3,241,805 | 0.21 |
| Munin4 | 1041 | 8,860,074 | 81 | 16,416,346 | 0.57 |
| Treat | 1422 | 463,717 | 441 | 6,767,084 | 0.03 |

Execution times are for a 1.4GHz AMD Athlon PC running Solaris 8. Notice that for all networks except three the posterior marginals given the evidence are computed in less than one second even though some networks contain more than a thousand variables.

## 6.2. *Learning*

In the performance analysis of the learning functionality, we have used the ALARM network, which has become a standard benchmark for structure learning.[37] The ALARM network consists of 37 variables and 46 edges. Each variable has between two and four states with an average of 1.2 parents of each variable, see Fig. 14.

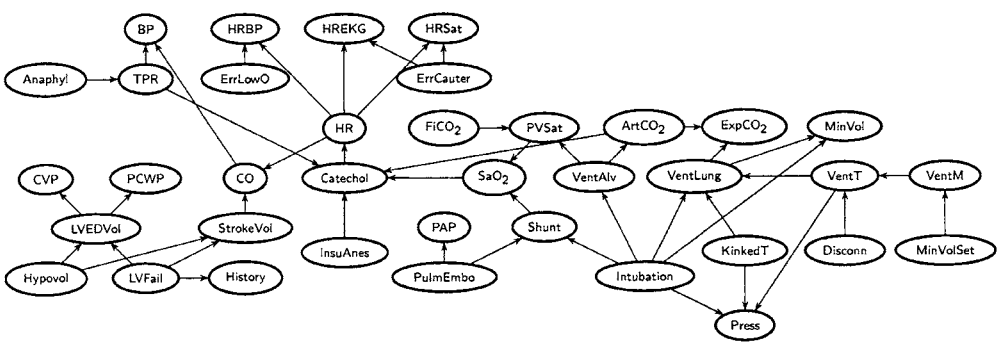


Fig. 14. The ALARM network.

The PDAG shown in Fig. 15, which is the result of NPC learning on a sample of 10,000 cases generated from the ALARM network with a significance level of $\alpha = 0.01$, contains three ambiguous regions $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$:

$$\mathcal{R}_1 = \{(\mathsf{HR}, \mathsf{ArtCO2}), (\mathsf{ArtCO2}, \mathsf{Catechol}), (\mathsf{Catechol}, \mathsf{ExpCO2})\}, \tag{40}$$

$$\mathcal{R}_2 = \{(\mathsf{VentAlv}, \mathsf{KinkedTube}), (\mathsf{VentLung}, \mathsf{KinkedTube}), (\mathsf{MinVol}, \mathsf{KinkedTube})\}, \tag{41}$$

$$\mathcal{R}_3 = \{(\mathsf{CVP}, \mathsf{LVFailure}), (\mathsf{LVEDVolume}, \mathsf{LVFailure}), (\mathsf{PCWP}, \mathsf{LVFailure}), (\mathsf{LVEDVolume}, \mathsf{History})\}, \tag{42}$$

The three ambiguous regions will be resolved by selecting the *correct* edges ((ArtCO2, Catechol), (VentLung, KinkedTube), and (LVEDVolume, LVFailure)). Subsequently, adjacent edges are directed correctly. A few edges cannot be directed based on the data alone, a wrong collider is present at Intubation, no other edge is directed incorrectly, no extra edges are present, and one edge (MinVol, Intubation) is missing. The result of applying the PC algorithm to the same set of data produced a DAG with two incorrect colliders, the two edges (TPR, Anaphylaxis) and (LVFailure, History) missing, and a few incorrect directions on edges. At the moment, the PC algorithm generates a DAG structure where some edges have been directed at random. The NPC algorithm with $\alpha = 0.05$ produce no missing edges, but an additional collider at Intubation.

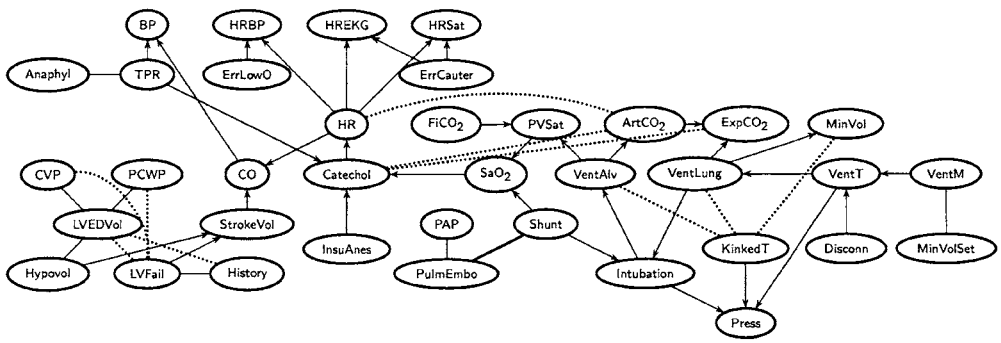


Fig. 15. NPC learning on a sample from ALARM with $\alpha = 0.01$.

To analyze the performance of the PC and NPC algorithms as a function of the significance level $\alpha$ we have performed tests with values of $\alpha$ equal to 0.001, 0.01, 0.05, and 0.1. The results are shown in Table 9. The tests have been performed using samples generated from the ALARM network.

Table 9 Results from using different values of $\alpha$.

| | $\alpha$ | Edges | $A \rightarrow B$ | Time (sec) | Ambiguous Regions | Ambiguous edges |
|---|---|---|---|---|---|---|
| PC | 0 001 | 45.25(0.5) | 44.75 | 0.4 | | |
| PC | 0.01 | 45.5 (0.25) | 42.5 | 0.4 | | |
| PC | 0.05 | 44.25(0.0) | 41.75 | 0.4 | | |
| PC | 0.1 | 45.25(0.25) | 44.0 | 0.4 | | |
| NPC | 0.001 | 43.75(0.0) | 39.0 | 4.0 | 1.5 | 5.0 (1.25) |
| NPC | 0.01 | 44.0 (0.25) | 34.5 | 4.2 | 2.0 | 7.0 (2.0) |
| NPC | 0.05 | 43.0 (0.0) | 36.25 | 3.8 | 2.25 | 9.25(2.75) |
| NPC | 0.1 | 44.25(0.0) | 38.75 | 4.1 | 1.0 | 4.0 (1.0) |

The table shows the number of edges found including neighbors with the number of incorrect edges found in parentheses, the number of edges with correct orientation, and the time to perform the learning in seconds for both algorithms. Furthermore, for the NPC algorithm the table also shows the number of ambiguous regions and the number of ambiguous edges in each region with the number of missing edges, which are represented as an ambiguous edge in parentheses. The values are average values over 25 samples of 10,000 cases with 5% missing values (MCAR). Similarly, Fig. 16 shows the number of edges identified by the PC and NPC algorithms on data generated from the ALARM network as a function of the sample size. The PC algorithm removes too many edges for small sample size, whereas the NPC algorithm remedies this using the necessary path condition.

The results show that the average run-time of the PC algorithm is lower than that of the NPC algorithm. The PC algorithm is faster since fewer tests are performed and since there is no notion of ambiguous regions requiring additional computations. The PC algorithm is able to direct more edges then the NPC algorithm, but some of these are directed at random in order to obtain a DAG.

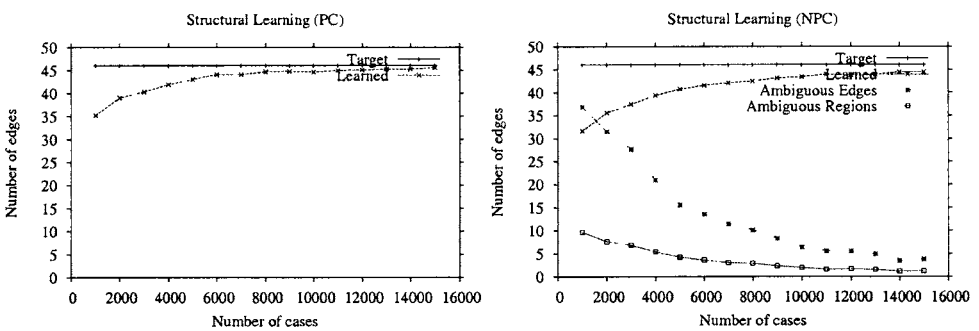


Fig. 16. Results of structure learning as a function of $N$.

The performances of the PC and NPC algorithms on large networks have also been analyzed using randomly generated networks. For a fixed size in terms of the number of variables, ten networks with random topology (zero to five parents) and distribution have been generated. Each variable has from two to five states. The results are shown in Table 10. All tests have been performed on a HP Omnibook

Table 10. Average run-time in seconds as a function of the number of nodes.

| | 25 | 50 | 75 | 100 | 150 | 200 |
|---|---|---|---|---|---|---|
| PC | 0.2 | 0 8 | 2 | 5 | 16 | 29 |
| NPC | 2 | 11 | 30 | 71 | 204 | 330 |

xe4500 with a 1.7GHz Pentium 4 processor and 256MB of RAM running Linux Redhat 8.

Table 11. On the efficiency of learning the structure of randomly generated Bayesian networks.

| | PC | | | NPC | | | |
|---|---|---|---|---|---|---|---|
| | time | edges | $A \rightarrow B$ | time | edges | $A \rightarrow B$ | regions |
| ALARM | 0.42 | 44.6 (1.4) | 42.5 | 3.8 | 43.4 (2.6) | 36.25 | 2.25 |
| net25 | 0.2 | 27.3 (1.8) | 14.1 | 2 | 27.0 (2.1) | 12.4 | 0.5 |
| net50 | 0.8 | 61.4 (4.6) | 30.9 | 11 | 60.4 (5.6) | 30.3 | 1.1 |
| net75 | 2 | 93.9 (7.5) | 48.1 | 30 | 92.8 (8.6) | 45.3 | 0.5 |
| net100 | 5 | 125.0 (12 2) | 67.1 | 71 | 123.0 (14.2) | 63.4 | 1.7 |
| net150 | 16 | 188.7 (14.9) | 89.7 | 204 | 186.6 (17.0) | 83.7 | 1.9 |
| net200 | 29 | 250.9 (23.3) | 128.1 | 330 | 247.8 (26.4) | 120.1 | 2.9 |

Table 11 shows average run-time in seconds, average number of correct (missing) edges, and average number of correctly orientated edges. Recall that the PC algorithm in its current implementation direct all edges and many at random.

The performance tests have been performed using 10,000 cases with 5% of the values missing completely-at-random (MCAR) drawn at random from the distribution of the network and a significance level of $\alpha = 0.05$.

## 7. Concluding Remarks

The Hugin Graphical User Interface provides the user with an easy-to-use front end to the Hugin Decision Engine. In addition, the Hugin Graphical User Interface supplies the user with functionality supporting efficient use of Bayesian networks and influence diagrams. For instance, the Hugin Graphical User Interface contains wizards that guide the user through the necessary steps for learning a Bayesian network from data. The Hugin Graphical User Interface is a tool for both experts and novices of probabilistic graphical models.

The core of the Hugin Decision Engine is implemented using the ISO C programming language. This implies that the Hugin Decision Engine is highly portable. Application programming interfaces to the Hugin Decision Engine are available for C, C++, and Java programming languages as well as for Visual Basic for Applications. The Hugin Graphical User Interface is implemented in the Java programming language using the Hugin Java API.

As the Hugin Tool is a commercial product, numerical stability and performance are major issues in the implementation of the Hugin Decision Engine. A lot of effort has been and will be put into achieving as high numerical stability and performance as possible. The performance of the Hugin Tool has been illustrated on both artificial and real-world Bayesian networks.

A free version of the Hugin Tool for demonstration purposes only can be downloaded from the web-site http://www.hugin.com.

## References


1. J. Pearl. *Probabilistic Reasoning in Intelligence Systems.* Series in Representation and Reasoning. Morgan Kaufmann Publishers, 1988.
2. R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer-Verlag, 1999.
3. F. V. Jensen. *Bayesian Networks and Decision Graphs.* Springer-Verlag, 2001.
4. R. E. Neapolitan. *Learning Bayesian Networks.* Prentice Hall, 2003.
5. R. A. Howard and J. E. Matheson. Influence Diagrams. In *The Principles and Applications of Decision Analysis*, volume 2, chapter 37, pages 721–762. 1981.
6. S. K. Andersen, K. G. Olesen, F. V. Jensen, and F. Jensen. HUGIN — a Shell for Building Bayesian Belief Universes for Expert Systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1080–1085, 1989.
7. F. Jensen, U. B. Kjærulff, M. Lang, and A. L. Madsen. HUGIN - The Tool for Bayesian Networks and Influence Diagrams. In *First European Workshop on Probabilistic Graphical Models*, pages 212–221, 2002.
8. A.L. Madsen, M. Lang, U. Kjærulff, and F. Jensen. The Hugin Tool for Learning Bayesian Networks. In *Proceedings of the 7th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 594–605, 2003.
9. S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H. G. Leimer. Independence properties of directed Markov fields. *Networks*, 20(5):491–505, 1990.
10. D. Geiger, T. Verma, and J. Pearl. Identifying independence in Bayesian networks. *Networks*, 20(5):507–534, 1990. Special Issue on Influence Diagrams.
11. S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, B.*, 50(2):157–224, 1988.
12. S. L. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87(420):1098–1108, 1992.
13. S. L. Lauritzen and F. Jensen. Stable Local Computation with Mixed Gaussian Distributions. *Statistics and Computing*, 11(2):191–203, 2001.
14. H. Raiffa. *Decision Analysis.* Addison-Wesley, Reading, MA, 1968.
15. Daphne Koller and Avi Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 302–313, 1997.
16. A. Berry, J.-P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, 11(3):397–403, 2000.
17. V. Bouchitté and I. Todinca. Treewidth and minimum fill-ins: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.
18. K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 27–31, 1997.
19. F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian Updating in Causal Probabilistic Networks by Local Computations. *Computational Statistics Quarterly*, 4:269–282, 1990.
20. A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.
21. F. Jensen, F. V. Jensen, and S. L. Dittmer. From Influence Diagrams to Junction Trees. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 367–373, 1994.
22. S. L. Lauritzen and D. Nilsson. Representing and solving decision problems with limited information. *Management Science*, 47:1238–1251, 2001.

23. A. L. Madsen and D. Nilsson. Solving Influence Diagrams using HUGIN, Shafer-Shenoy and Lazy Propagation. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 337–345, 2001.
24. R. G. Cowell and A. P. Dawid. Fast retraction of evidence in a probabilistic expert system. *Statistics and Computing*, 2:37–40, 1992.
25. F. Jensen and S. K. Andersen. Approximations in Bayesian Belief Universes for Knowledge Based Systems. In *Uncertainty in Artificial Intelligence 4*, pages 162–129, 1990.
26. A. L. Madsen and F. V. Jensen. Lazy Evaluation of Symmetric Bayesian Decision Problems. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 382–390, 1999.
27. F. V. Jensen, B. Chamberlain, T. Nordahl, and F. Jensen. Analysis in HUGIN of data conflict. In *Uncertainty in Artificial Intelligence*, pages 519–528, 1991.
28. Judea Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge University Press, Cambridge, UK, 2000.
29. P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search.* Adaptive Computation and Machine Learning. MIT Press, Cambridge, Massachusetts, second edition, 2000.
30. N. Wermuth and S. L. Lauritzen. Graphical and recursive models for contingency tables. *Biometrika*, 70:537–552, 1983.
31. T. Verma and J. Pearl. An Algorithm for Deciding if a Set of Observed Independencies Has a Causal Explanation. In *Proceedings of the 8th Conference on Uncertainty in Artificial Intelligence*, pages 323–330, 1992.
32. P. Spirtes and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computing Review*, 9(1):62–72, 1991.
33. H. Steck and V. Tresp. Bayesian belief networks for data mining. Proceedings of the 2nd Workshop "Data Mining und Data Warehousing als Grundlage moderner enscheidungsunterstuetzender Systeme", 1999.
34. C. Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 403–410, 1995.
35. S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics & Analysis*, 19:191–201, 1995.
36. D. Spiegelhalter and S. L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605, 1990.
37. I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, pages 247–256, 1989.