TECHNICAL RESEARCH REPORT

An Introduction to Belief Networks

by Hongjun Li

CSHCN T.R. 99-31
(ISR T.R. 99-58)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

An Introduction to Belief Networks *

Hongjun Li
Institute for Systems Research and
Center for Satellite and Hybrid Communication Networks
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742

October 16, 1998

Abstract

Belief networks, also called Bayesian networks or probabilistic causal networks, were developed in the late 1970's to model the distributed processing in reading comprehension. Since then they have attracted much attention and have become popular within the AI probability and uncertainty community. As a natural and efficient model for human's inferential reasoning, belief networks have emerged as the general knowledge representation scheme under uncertainty [14][15][16][17].

In this report, we first introduce belief networks in the light of knowledge representation under uncertainty, then in the following sections we give the descriptions of the semantics, inference mechanisms and some issues related to learning belief networks, respectively. This report is not intended to be a tutorial for beginners. Rather it aims to point out some important aspects of belief networks and summarize some important algorithms. For those who are interested in a tutorial, we refer to [4].

*This work was supported by the Center for Satellite and Hybrid Communication Networks, under NASA cooperative agreement NCC3-528.

1 Representing Knowledge under Uncertainty

The attempts to model human's inferential reasoning, namely the mechanism by which people integrate data from multiple sources and come up with a coherent interpretation of the data, have motivated much research in various areas within the artificial intelligence discipline. One of the most popular approaches to AI involves constructing an *"intelligent agent"* that functions as a narrowly focused expert.

While the past decades have seen some important contributions of expert systems in medical diagnosis, financial analysis and engineering applications, problematic expert system design issues still remain. Dealing with uncertainty is among the most important since uncertainty is the rule, not the exception, in most practical applications. This is based on two observations:

- The concrete knowledge, or the observed evidence from which reasoning will begin, is not accurate.
- The abstract knowledge, namely the knowledge stored in the expert systems as the model of human reasoning, is probabilistic rather than deterministic.

Therefore a natural starting point would be to cast the reasoning process in the framework of probability theory and statistics. However, cautions must be taken if this casting is interpreted in a textbook view of probability theory [13]. For example, if we assume that human knowledge is represented by a joint probability distribution (JPD), $p(x_1, ..., x_n)$, on a set of random variables (propositions), $x_1, ..., x_n$, then the task of reasoning given evidence $e_1, ..., e_k$ is nothing but computing the probability of a small set of hypotheses $p(H_1, ..., H_m|e_1, ..., e_k)$—the **belief** of the hypotheses given the set of evidence. So one may conclude that given JPD, such kind of computing is merely arithmetic labor.

Though it is true that JPD suffices to answer all kinds of queries on $x_1, ..., x_n$, this view turns out to be a rather distorted picture of human reasoning and computing queries in this way is cumbersome at least, if not intractable at all. For example, if we are to encode explicitly for binary variables $x_1, ..., x_n$ an arbitrary JPD $p(x_1, ..., x_n)$ on a computer, we will have to build up a table with 2^n entries—an unthinkably large number. Even if there is some economical way to compact this table, there still remains the problem of manipulating it to obtain queries on propositions of interest. For example, to compute $p(\mathbf{H}|\mathbf{e})$ (where $\mathbf{H}$ and $\mathbf{e}$ are the sets of hypotheses and evidence, respectively), according to Bayes' rule,

$$p(\mathbf{H}|\mathbf{e}) = \frac{\mathrm{p}(\mathbf{e}|\mathbf{H})\mathrm{p}(\mathbf{H})}{\mathrm{p}(\mathbf{e})} \tag{1}$$

we need to compute the marginal probabilities $p(\mathbf{e})$, $p(\mathbf{H})$ and the likelihood $p(\mathbf{e}|\mathbf{H})$, which incurs enormously large number of calculations.

Human reasoning, on the contrary, acts differently in that probabilistic inference on a small set of propositions is executed swiftly and reliably while judging the likelihood of the conjunction of a large number of propositions turns out to be difficult. This suggests that the elementary building blocks of human knowledge are not the entries of a JPD table but, rather, the low-order marginal

probabilities and conditional probabilities defined "locally" over some small set of propositions. It is further observed that an expert will feel more at ease to identify the dependence relationship between propositions than to give the numerical estimate of the conditional probability. This suggests that the dependence structure is more essential to human reasoning than the actual value. Noting also that the nature of dependence relationships between propositions resemble in many aspects that of connectivity in graphs, we can naturally represent such kind of relationship via more explicit graph approaches, which leads to belief networks.

Definition *A belief network is a Directed Acyclic Graph (DAG) in which:*

- *The nodes represent variables of interest (propositions), which maybe be discrete, assuming values from finite or countable states, or may be continuous.*
- *The set of directed links or arrows represent the causal influence among the variables and the parents of a node are all those nodes with arrows pointing to it.*
- *The strength of an influence is represented by conditional probabilities attached to each cluster of parent-child nodes in the network.*

As an example, consider the following situation. Suppose you have a new burglar alarm installed at home. It is fairly reliable in detecting a burglary, but also responds on occasion to minor earthquakes. You also have two neighbors, John and Mary, who have promised to call you at work if they hear the alarm. John always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, rather likes loud music and sometimes misses the alarm altogether. Given the evidence of who has or has not called, we would like to estimate the probability of a burglary. This simple scenario is illustrated in figure 1 [18]. The letters B, E, A, J, and M stand for Burglary, Earthquake, Alarm, JohnCall, and MaryCall, respectively.

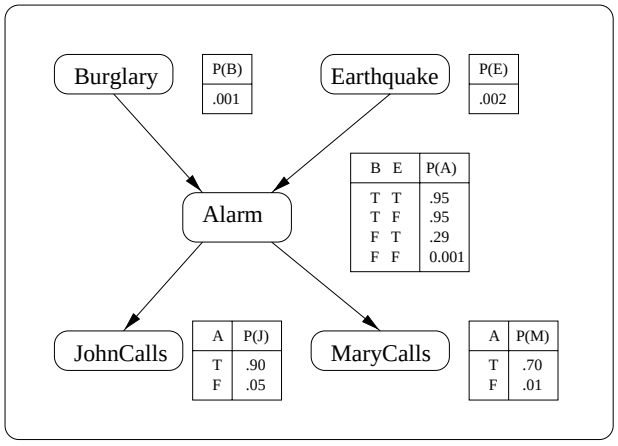

Figure 1: An example of Belief Network

As mentioned above, it is usually easy for a domain expert to decide what dependence relationships hold in the domain. Once the topology of the belief network is specified, we need only to specify conditional probabilities for the nodes that participate in direct influences, and use those

to compute any other probability values. To do this, we need to set up for each node a conditional probability table (CPT). When a node has many parents, however, specifying even its local distribution can also be quite onerous since we have to specify the probabilities of the node conditioned on **every** instance of its parents, which is not desirable even when there are only 6 or 7 parents. We can introduce more structure, such as noisy-OR model [14], into the belief network model to reduce this burden.

2 The Semantics of Belief Networks

There are two equivalent ways in which one can understand the semantics of belief networks. The first one is to treat the network as a representation of the JPD, which is helpful in understanding how to *construct* networks. The second is to view it as an encoding of a set of conditional independence statements, which turns out helpful in designing *inference* algorithms.

2.1 Representation of joint probabilities

A belief network for problem domain $\{x_1, ..., x_n\}$ represents the JPD over those random variables. The representation consists of a set of local conditional probability distributions, combined with a set of assertions of conditional independence that allow us to construct the global joint distribution from the local distributions.

Based on the chain rule of probability, we have

$$p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1}). \tag{2}$$

Given a DAG G and a JPD P over a set $\mathbf{x} = \{\mathrm{x}_1, ..., \mathrm{x}_\mathrm{n}\}$ of discrete variables, we say that G *represents* P if there is a one-to-one correspondence between the variables in $\mathbf{x}$ and the nodes of G, such that

$$p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i | \Pi_i). \tag{3}$$

If we order the nodes in such a way that the order of a node is larger than those of its parents and smaller than those of its children (the so-called *topological ordering*), we have

$$p(x_i | x_1, ..., x_{i-1}) = p(x_i | \Pi_i), \tag{4}$$

which means given its parent set $\Pi_i \subseteq \{x_1, ..., x_{i-1}\}$, the set of variables that render x_i, each variable x_i is conditionally independent of all its other predecessors $\{x_1, ..., x_{i-1}\} \backslash \Pi_i$.

Therefore, we can construct a belief network following the steps below [18]:

- Choose the set of random variables that describe the domain

- Give order numbers to the random variables using topological ordering
- While there are still variables left:
  - Pick a random variable and add a node representing it
  - Choose parents for it as the minimal set of nodes already in the network such that (4) is satisfied
  - Specify the CPT for it

Since each node is only connected to earlier nodes, this construction method guarantees that the network is acyclic. Furthermore, it is both complete and consistent [13].

We can also see that, in the context of using Bayes' rule and (4), conditional independence relationships among variables really simplify the computation of query results and greatly reduce the number of conditional probabilities that needs to be specified.

Suppose we have a simple belief network(a Markov Chain in this case) with structure $w \to x \to y \to z$, and we want to know $p(w|z)$ [9]. One way to get this is by

$$p(w|z) = \frac{p(w,z)}{p(z)} = \frac{\sum_{x,y} p(w,x,y,z)}{\sum_{w,x,y} p(w,x,y,z)}, \tag{5}$$

where $p(w,x,y,z)$ is the joint distribution determined from the belief network. In practice, this approach is not feasible since it entails summing over an exponential number of terms, as shown before. Fortunately, we can take advantage of the conditional independencies encoded in this belief network to make the computation more efficient, as shown below:

$$p(w|z) = \frac{\sum_{x,y} p(w,x,y,z)}{\sum_{w,x,y} p(w,x,y,z)} = \frac{p(w)\sum_x p(x|w)\sum_y p(y|x)p(z|y)}{\sum_x p(w)\sum_x p(x|w)\sum_y p(y|x)p(z|y)}, \tag{6}$$

where you can get each term above by simply drawing from the CPTs already specified for each node.

Thus, using conditional independence, we can reduce the dimensionality of the problem by rewriting the sums over multiple variables as the product of sums over a single variable or at least smaller numbers of variables.

2.2 Representation of conditional independence relations

We have described above the conditional independence of a node and its predecessors, given its parents. But, is this the only and general case of conditional independence? In other words, given a set of evidence nodes E, is it possible to "read off" whether a set of nodes in X is independent of another set Y, where X and Y are not necessarily parents and children? This is an important issue in designing inference algorithms.

Fortunately, the answer is yes and the methods are provided by the notion of **d-separation**, which means **direction-dependent separation** [14]. If each undirected path from a node in X to a node in Y is d-separated by E, we say X and Y are *blocked* , which means there will be no way at all for X and Y to communicate if we remove E, and thus conditionally independent.

Definition [14] : *Let X, Y and Z be three disjoint subsets of nodes in a DAG G, then Z is said to* **d-separate** *X and Y, iff along every undirected path from each node in X to each node in Y there is an intermediate node A such that either*

- *A is a head-to-head node (with converging arrows) in the path, and neither A nor its descendents are in Z, or*
- *A is in Z and A is not a head-to-head node.*

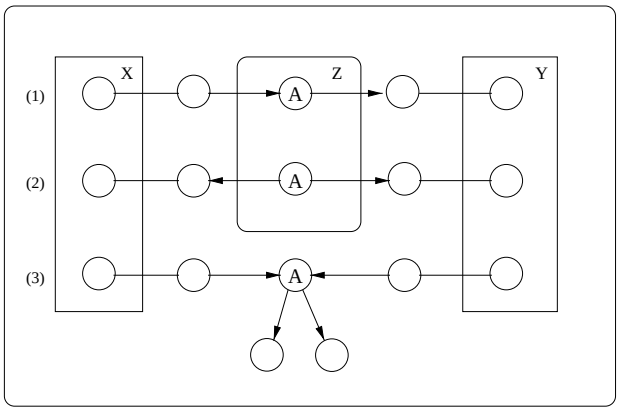

Figure 2: An Illustration of d-separation

To better understand the notion of d-separation, we enumerate all of the possible cases, as shown in figure 2. In case (1), or the so-called serial connections case, we say evidence A will block node X and Y if it is initiated. In case (2), or the so-called diverging connections case, evidence A will block the communications between its children, node X and Y in this case, if it is initiated. The above two cases correspond to item two in the above definition. In case (3), node A is a converging node and the communications between its parents will be open if it is initiated. That's why we say that neither A nor its descendents are in Z if A is a head-to-head node, which corresponds to item one in the above definition.

Based on the concept of d-separation, we can derive the inference algorithms, which come next.

3 Inference in Belief Networks

By now we can say that a belief network constitutes a model of the *environment* rather than, as in many other knowledge representation schemes such as rule-based systems and neural networks,

a model of the reasoning process. Actually, it simulates the mechanism that operate in the environment and thus allows for various kinds of inferences (also called **evidence propagation**) [18]. The question here is: How can one infer the (probabilities of) values of one or more network variables, given observed values of others? Or, in mathematics, we are going to find $P(Q = q|E = e)$, where Q is the query variable set and E is the set of evidence variables. Based on the choice of Q and E, there are four distinct kinds of inference patterns, as described below and shown in figure 3:

- **Diagnosis inferences**: From effects to causes, also called abductive inferences, bottom-up or backward inference. For example: "What is the most probable explanations for the given set of evidence?"
- **Causal inferences** : From causes to effects, also called predictive inferences, top-down or forward inferences. For example: "Having observed A, what is the expectation of B?"
- **Inter-causal inferences** : Between causes of a common effect. For example: "If C's parents are A and B, then what is the expectation of B given both A and C?" Namely, what is the belief of the occurrence of one cause on the effect given that the other cause is true? The answer is that the presence of one makes the other less likely (explaining away).
- **Mixed inferences** : combining two or more of the above.

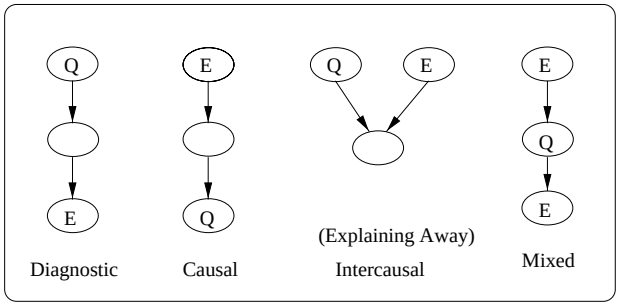

Figure 3: An Illustration of four inference patterns

There are basically three types of algorithms for propagating evidence: exact, approximate and symbolic. By *exact propagation* we mean a method that, apart from precision or round-off errors, computes the probability distribution of the nodes exactly. By *approximate propagation* we mean the answers computed are not exact but, with high probability, lie within some small distance of the correct answer. Finally, *symbolic propagation*, which computes the probabilities in symbolic form, can deal not only with numerical values, but also with symbolic parameters.

3.1 Exact evidence propagation

One of the most widely adopted algorithms is designed for a simple belief network model whose associated graph has a polytree structure. For more complicated multiply-connected networks, some transform methods, such as conditioning or clustering, can be used to obtain the equivalent polytree structure.

3.1.1 Exact propagation in polytrees

A **tree** is a connected undirectional graph in which for every pair of nodes there exists a unique path. We call a tree as **polytree** (also singly connected network) if every node in it may have more than one parents. Note also that in a polytree, every node X_i divides the polytree into two disconnected polytrees: one includes the nodes that can be accessed through its parents of X_i and the other includes the nodes that can be accessed through its children, as shown in figure 4. Thus for a give set of evidence nodes E, we say $E = E_i^+ \cup E_i^-$. The following is the derivation of the polytree inference algorithm.

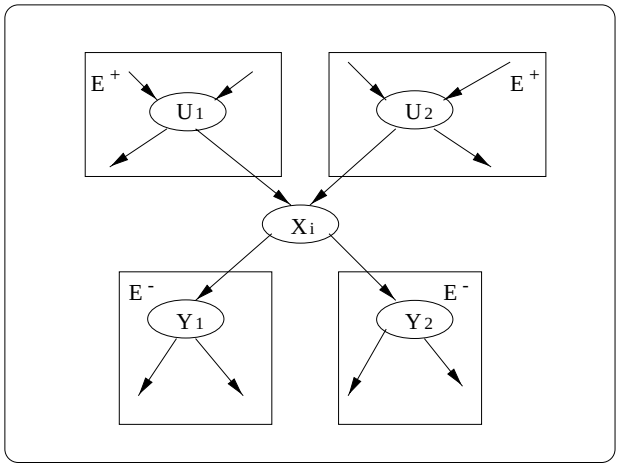

Figure 4: An Illustration of a polytree

- Decomposition:

$$\begin{aligned} p(x_i|e) &= p(x_i|e_i^+, e_i^-) \\ &= \frac{1}{p(e_i^+, e_i^-)} p(e_i^+, e_i^-|x_i)p(x_i) \\ &= kp(e_i^-|x_i)p(e_i^+|x_i)p(x_i) \\ &\propto p(e_i^-|x_i)p(x_i, e_i^+) \\ &\propto \lambda_i(x_i)\mu_i(x_i) \\ &\propto \rho_i(x_i) \end{aligned} \tag{7}$$

where, "$\propto$" stands for "proportional", k is the normalization factor, and $\lambda_i(x_i) \stackrel{\triangle}{=} p(e_i^-|x_i)$, $\mu_i(x_i) \stackrel{\triangle}{=} p(x_i, e_i^+)$.

Suppose for node X_i, sets $\{U_1, ..., U_p\}$ and $\{Y_1, ..., Y_c\}$ are parents and children of this node, respectively, then we can write

$$E_i^+ = \{E_{U_1X_i}^+, ..., E_{U_pX_i}^+\}$$

$$E_i^- = \{E_{X_iY_1}^-, ..., E_{X_iY_c}^-\}$$

- Calculation of $\mu_i(x_i)$:

$$\begin{aligned}\mu_i(x_i) &= p(x_i, e_i^+) \\ &= \sum_u p(x_i, e_i^+, u) \\ &= \sum_u p(x_i|e_i^+, u)p(e_i^+, u) \end{aligned} \tag{8}$$

Since by the case (3) of d-separation definition in section 2.2

$$\begin{aligned}p(e_i^+, u) &= p(\{e^+_{u_1x_i}, u_1\}, \ldots, \{e^+_{u_px_i}, u_p\}) \\ &= \prod_{j=1}^{p} p(e^+_{u_jx_i}, u_j), \end{aligned} \tag{9}$$

we have

$$\mu_i(x_i) = \sum_u p(x_i|e_i^+, u) \prod_{j=1}^{p} \underbrace{p(e^+_{u_jx_i}, u_j)}_{\mu_{u_jx_i}(u_j)}, \tag{10}$$

where $\mu_{u_jx_i}(u_j)$ is called the μ-message that carries the evidential information from node X_i's direct parent U_j, with its value as u_j.

- Calculation of $\lambda_i(x_i)$:

$$\begin{aligned}\lambda_i(x_i) &= p(e_i^-|x_i) \\ &= p(e^-_{x_iy_1}, ..., e^-_{x_iy_c}|x_i) \\ &= \prod_{j=1}^{c} \underbrace{p(e^-_{x_iy_j}|x_i)}_{\lambda_{y_jx_i}(x_i)}, \end{aligned} \tag{11}$$

where the last equality comes from diverging-node-case of d-separation definition, as shown in section 2.2. Here, $\lambda_{y_jx_i}(x_i)$ is called the λ-message that carries the evidential information from node X_i's direct child Y_j, with its value as y_j.

- Calculation of μ-message (top-down propagation):

Note that if we stand at node X_i, the message that it has to create is the μ-message from X_i down to its children and the λ-message upward to its parents. So we consider $\mu_{x_iy_j}(x_i)$ here. Also note that

$$e^+_{x_iy_j} = e^+_{x_i} \bigcup_{k\neq j} e^-_{x_iy_k} \tag{12}$$

where, e_i^+ is the evidence from Y_j's parent X_i and beyond while $\bigcup_{k\neq j} e^-_{x_iy_k}$ is the evidence from X_i's other children, or Y_j's cousins. Then

$$\begin{aligned}\mu_{x_iy_j}(x_i) &= p(e^+_{x_iy_j}, x_i) \\ &= p(x_i, e_i^+, \bigcup_{k\neq j} e^-_{x_iy_k}) \end{aligned}$$

$$
\begin{aligned}
&= \underbrace{p(e_i^+|x_i, \bigcup_{k\neq j} e^-_{x_i y_k})}_{p(e_i^+|x_i)}\, p(x_i, \bigcup_{k\neq j} e^-_{x_i y_k}) \\
&= p(e_i^+|x_i)\, \underbrace{p(\bigcup_{k\neq j} e^-_{x_i y_k}|x_i)}_{\prod_{k\neq j} p(e^-_{x_i y_k}|x_i)}\, p(x_i) \\
&= p(x_i) p(e_i^+|x_i) \prod_{k\neq j} p(e^-_{x_i y_k}|x_i) \\
&\propto p(x_i|e_i^+) \prod_{k\neq j} \underbrace{p(e^-_{x_i y_k}|x_i)}_{\lambda_{y_k x i}(x_i)}
\end{aligned} \tag{13}
$$

which means: to compute $p(e^+_{x_i y_j}, x_i)$, the μ-message from parent X_i to child Y_j, you have to know beforehand the λ-message from all other children of node X_i.

- Calculation of λ-message (bottom-up propagation):

  Since

$$
e^-_{x_i y_j} = e^-_{y_j} \bigcup_{k\neq i} e^+_{x_k y_j}, \tag{14}
$$

  where $e^-_{y_j}$ is the evidence from X_i's child Y_j and beyond, while $e^+_{x_k y_j}$ is the evidence from Y_j's other parents, or X_i's cousins.

  Let $\mathbf{v} = \{\mathrm{v}_1, ..., \mathrm{v}_\mathrm{q}\}$ be the set of parents of node Y_j other than X_i, then we have

$$
\begin{aligned}
\lambda_{y_j x_i}(x_i) &= p(e^-_{x_i y_j}|x_i) \\
&= \sum_{y_j, v} p(e^-_{y_j}, y_j, v, e^+_{v,y_j}|x_i) \\
&= \sum_{y_j, v} \underbrace{p(e^-_{y_j}|y_j, v, e^+_{v,y_j}, x_i)}_{p(e^-_{y_j}|y_j)} \underbrace{p(y_j|v, e^+_{v,y_j}, x_i)}_{p(y_j|v,x_i)} \underbrace{p(v, e^+_{v,y_j}|x_i)}_{p(v,e^+_{v,y_j})} \\
&= \sum_{y_j, v} p(e^-_{y_j}|y_j) p(y_j|v, x_i) p(v, e^+_{v,y_j}) \\
&= \sum_{y_j} p(e^-_{y_j}|y_j) \sum_{v} p(y_j|v, x_i) p(v, e^+_{v,y_j}) \\
&= \sum_{y_j} \underbrace{p(e^-_{y_j}|y_j)}_{\lambda_j(y_j)} \sum_{v_1,...,v_q} p(y_j|\pi_{y_j}) \prod_{k=1}^{q} \mu_{v_k y_j}(v_k)
\end{aligned} \tag{15}
$$

which means, to compute $p(e^-_{x_i y_j}|x_i)$, the λ-message from child Y_j to parent X_i, we have to know the μ-message from all other parent $\mathbf{v}$ of this particular child Y_j.

In summary, the belief distribution of variable X_i is based on two-way message passing and can be computed iteratively as follows, see also [3]:

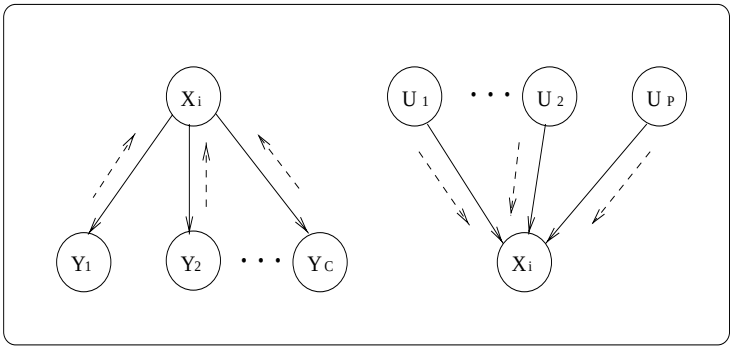

Figure 5: calculation of $\mu_i(x_i)$ and $\lambda_i(x_i)$

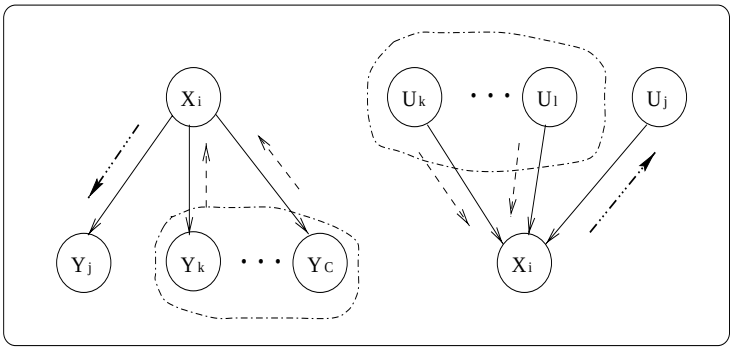

Figure 6: calculation of μ-message and λ-message

- If X_i has received the μ-messages from all of its parents, then calculate $\mu_i(x_i)$, see figure 5.
- If X_i has received the λ-message from all of its children, then calculate $\lambda_i(x_i)$, see figure 5 .
- If $\mu_i(x_i)$ has been calculated, then for every child Y_j from which X_i has received the λ-message from all of its other children, calculate and send the μ-message, which is $\mu_{x_i y_j}(x_i)$, see figure 6.
- If $\lambda_i(x_i)$ has been calculated, then for every parent U_j from which X_i has received the μ-message from all of its other parents, calculate and send the λ-message, which is $\lambda_{x_i u_j}(u_j)$, see figure 6.

For all non-evidential nodes X_i, we can calculate $\rho_i(x_i) = \lambda_i(x_i)\mu_i(x_i)$, the unnormalized probability $p(x_i|e)$, and we can normalize this over all non-evidential nodes to obtain the desired probability. The computational complexity of this algorithm is linear in the number of nodes, which is quite desirable if the application happens to be this case.

3.1.2 Exact propagation in multiply-connected networks

In practice one may encounter a probabilistic model that can not be represented by a polytree, in which cases we have to work with multiply-connected networks (graphs that contain loops).

To evaluate such networks exactly, one has to find some schemes to turn the network into an equivalent polytree structure. There are two major ways to perform this task, namely, *clustering* and *conditioning* , as shown in figure 7.

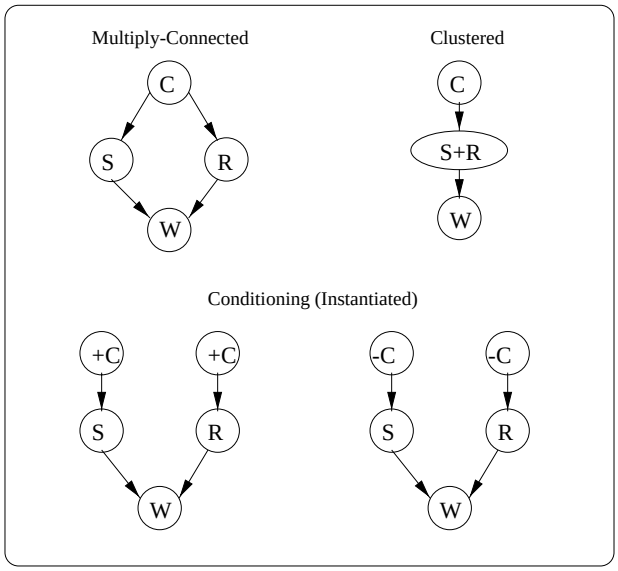

Figure 7: An Illustration of Multiply-connected Networks

Clustering method (also called clique-tree propagation [12]), the most popular and fastest propagation method for multiply-connected networks, works as follows. Starting with a directed network representation, the network is transformed into an undirected graph that retains all of its original dependencies. This graph, called Markov network, is then triangulated to form local clusters of nodes (cliques) that are tree-structured. The complexity for this method is exponential in the size of the largest clique found in some triangulation of the network.

Conditioning involves breaking the communication pathways along the loops by instantiating a selected group of variables while clustering consists of building associated graphs in which each node (mega-node) corresponds to a set of variables rather than just to a single variable (i. e. a join tree). Both methods result in a polytree.

3.1.3 Exact propagation in Gaussian networks

For continuous random variables, the Gaussian case is mostly studied since their distributions can be characterized simply by the mean vector and covariance matrix, and by central limit theorem, it is a rather general formulation. The evidence propagation in a Gaussian network is based on the following theorem[3].

Theorem (Conditionals of a Gaussian distribution) *Let Y and Z be two sets of random variables under a multivariate Gaussian distribution with mean vector and covariance matrix given by*

$$\mu = \begin{pmatrix} \mu_Y \\ \mu_Z \end{pmatrix}$$

and

$$\Sigma = \begin{pmatrix} \Sigma_{YY} & \Sigma_{YZ} \\ \Sigma_{ZY} & \Sigma_{ZZ} \end{pmatrix}$$

then the conditional probability distribution of Y *given* $Z = z$ *is also multivariate Gaussian with mean vector* $\mu_{Y|Z=z}$ *and covariance matrix* $\Sigma_{Y|Z=z}$ *given by*

$$\mu_{Y|Z=z} = \mu_Y + \Sigma_{YZ}\Sigma_{ZZ}^{-1}(z - \mu_Z)$$

$$\Sigma_{Y|Z=z} = \Sigma_{YY} - \Sigma_{YZ}\Sigma_{ZZ}^{-1}\Sigma_{ZY}$$

Thus, given a set of evidential nodes $E \subset X$ with values $E = e$, we can easily derive the mean vector and covariance matrix for the interested propositions $Y \subset X$ by simply replacing every Z above with E.

3.2 Approximate evidence propagation

Although we can exploit conditional independence in a belief network for probabilistic inference, exact inference in an arbitrary belief network is NP-hard [5]. Thus, from the practical point of view, exact propagation methods may be restrictive or even inefficient in situations where the network structure requires a large number of computations, which will cost huge amount of memory and other computing resource. That's why we are interested in approximate propagation.

The basic idea is based on *Monte Carlo simulation*, which is to generate a sample of size N from the JPD of the variables and then use the generated sample to compute the approximate belief of some certain events given the evidence. The belief is represented by the *relative frequency* of events in the sample size, rather than the probability of the events, and thus approximation. Based on how the samples can be generated from the JPD, we can classify approximate propagation methods as stochastic simulation methods (such as acceptance-rejection sampling, uniform sampling, likelihood weighing, back-forward sampling, Markov sampling), and deterministic search methods (such as systematic sampling and maximum probability search).

The framework for simulation methods can be summarized as follows [3]:

- For $j = 1$ to N
  - Generate a sample $x^j = (x_1^j, ..., x_n^j)$ using the simulation probability distribution $h(x)$.
  - Calculate $s(x^j) = \frac{p(x^j)}{h(x^j)}$

- For every possible value y of Y, approximate $p(y)$ by

$$p(y) \approx \frac{\sum_{y \in x^j} s(x^j)}{\sum_{j=1}^{N} s(x^j)}$$

3.3 Symbolic evidence propagation

In the exact and approximate evidence propagation methods discussed above, we assume that the JPD of the model has been assigned numerically, that is, all the parameters must be assigned fixed numeric values. However, there are some cases that such assignments may not be available, for example the domain expert may would rather give only ranges or vague descriptions of the values for the parameters than their exact values. In such cases, symbolic propagation, which leads to solutions expressed as functions of the parameters in symbolic form, will play the role. The answers to specific queries can be obtained by simply plugging the values of the parameters into the symbolic solutions, without doing the propagation again. Moreover, symbolic propagation can also be used in sensitivity analysis, which concerns the sensitivity of the results to changes in parameter values.

It is observed that computing (generating) and simplifying symbolic expressions is a computationally expensive task, and it becomes increasingly inefficient when dealing with large networks or large set of symbolic parameters. So in practice, even in the case of simple network architecture, we usually restrict the symbolic parameters to a rather small set to carry out efficient propagation and analysis.

3.4 Remarks

As discussed above, exact evidence propagation in an arbitrary belief network is NP-hard. Fortunately, the complexity can be estimated prior to actual processing and when the estimates exceed reasonable bounds, an approximation methods such as stochastic simulation can be used instead. But even approximate inference (using *Monte Carlo simulation*) is also NP-hard if treated in general [7]. For many applications, however, the networks are small enough (or can be simplified sufficiently) so that these complexity results are not fatal. For applications where the usual inference methods are impractical, we usually develop techniques customer-tailored to particular network topologies, or particular inference queries. So specifying efficiently and accurately the structure as well as CPT for belief networks entails both keen engineering insights of the problem domain and the indispensable good sense of simplification to obtain the appropriate trading-off. It is still somewhat an art.

4 Learning Belief Networks

In previous discussions we assumed that both the network structure and the associated CPT are provided by human experts as the prior information. In many applications, however, such information is not available. In addition, different experts may treat the systems in various ways

and thus give different and sometimes conflicting assessments. In such cases, the network structure and corresponding CPT can be estimated using data and we refer to this process as *learning*. Even if such prior information does exist, it is still desirable to validate and improve the model using data. For more information on learning in belief networks, we refer to [6] [10].

As one may expect, learning belief networks consists of both structural learning (deriving the dependency structure G) and parametric learning (estimating P). The structure of the network may be *known* or *unknown*, and the variables in the network may be *observable* or *hidden*. There are basically four types of problems [18]:

- **1. Known structure, fully observable**: The only thing needed is to specify the CPT and this can be done by directly using the statistics of the data set S. Search methods such as hill-climbing or simulated annealing can be exploited to accomplish the fitting of data. Here, we treat this problem as a parameter learning problem and describe the statistical method for parameter estimation, as shown in section 4.1.
- **2. Known structure, hidden variables**: As stated before, human experts would rather give dependence relationships between random variables than the corresponding numerical values, especially when not all of the variables are observable. Therefore we can say that finding the topology of the network is often the easy part and thus the *known structure, hidden variable* learning problem is of great importance. Such problems are quite analogous to neural network learning, where gradient descent (or ascent) methods can be used. In section 4.2, we describe the gradient descent method for the adaptive probabilistic networks [1].
- **3. Unknown structure, fully observable**: We have to first reconstruct (extract) the topology, which entails determining the best possible structure through a space of available alternatives. The search is basically enumerative. Fitting the data to a particular structure reduces to problem 1. In section 4.3, we discuss briefly structure learning considerations.
- **4. Unknown structure, hidden variables**: When there are some hidden variables, the previous extraction techniques would not apply and there is no known good algorithms for this problem. So we don't include a section for this problem.

4.1 Parameter Learning

In this section, we begin with the simple one-parameter learning case, then we move to multi-variable parameter learning in belief networks.

4.1.1 Simple parameter learning

Imagine we have a coin, and we conduct an experiment whereby we flip the coin in the air, and it comes to land as either head or tail. We assume, as usual, that different tosses are independent, and that, in each toss, the probability of the coin landing heads is some real number θ. Our goal here is to estimate θ. To do this, we need to repeat the above experiment and see. If we toss it 100

times, of which 34 come up heads. What is the estimate for θ? Intuitively, it seems obvious that the answer is 0.34.

More formally, we can define the *likelihood* function as

$$P(D|\theta) = \theta^h(1-\theta)^t, \tag{16}$$

which is the probability with which we get a particular data set D with h heads and t tails given that the probability θ has a certain value. It is straightforward to verify that the value of θ which maximizes $P(D|\theta)$ is $\frac{h}{h+t}$. This is called *maximum likelihood (ML)* estimate for θ.

This seems plausible, but is overly simplistic in many cases. Suppose in one experiment we get 3 heads out of 10 tosses. It might be quite reasonable to conclude that the parameter θ is 0.3. But what if we know that the coin is actually *fair* ? We would be much less likely to jump to the conclusion that the parameter θ is 0.3. In this regard, we say that *prior knowledge* should play an important role in parameter estimation.

In Bayesian statistics, we simply put a distribution over anything about which we have uncertainty. In this case, since we are uncertain about θ, we define a prior distribution $P(\theta)$.

Then we have a joint distribution of both the tosses and the parameter θ:

$$\begin{aligned} P(x_1, \cdots, x_N, \theta) &= P(x_1, \cdots, x_N|\theta)P(\theta) \\ &= P(D|\theta)P(\theta) \\ &= P(\theta)\theta^h(1-\theta)^t, \end{aligned} \tag{17}$$

where $P(x_1, \cdots, x_N|\theta) = P(D|\theta)$ is just the likelihood function. Now as we see more data, the *posteriori* distribution over the parameter changes. In particular, using Bayes rule, we have

$$P(\theta|D) = \frac{P(\theta)P(D|\theta)}{P(D)}, \tag{18}$$

where the denominator is the normalizing factor, and in this case is equal to

$$\int_0^1 P(\theta)P(D|\theta)d\theta. \tag{19}$$

An appropriate such distribution for a parameter is the *Beta* distribution. A Beta function distribution is parameterized by two numbers α_h, α_t. Intuitively, these correspond to the number of imaginary heads and tails that have been seen. The Beta function function with these parameters has the following form,

$$P(\theta) = Beta(\theta|\alpha_h, \alpha_t) \propto \theta^{\alpha^h - 1}(1-\theta)^{\alpha_t - 1} \tag{20}$$

and some of the Beta distributions are shown in figure 8.

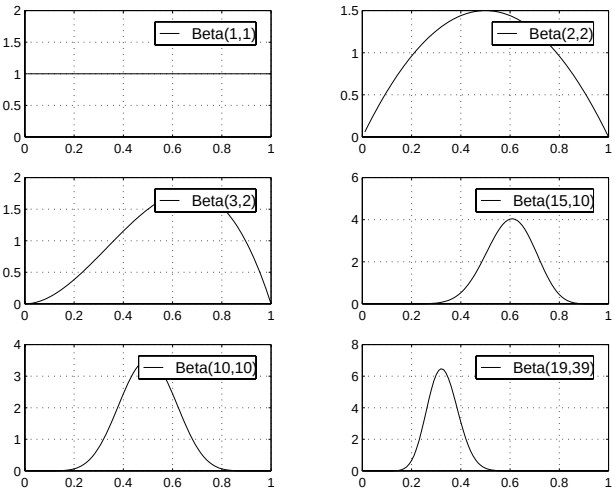

Figure 8: An Illustration of some Beta Distributions

Beta distributions have properties that make them particularly useful for parameter estimation. *First*, for $Beta(\theta|\alpha_h, \alpha_t)$, the probability of next coin toss coming out head is:

$$\begin{aligned} P(X = h|\theta) &= \int_0^1 P(\theta)P(X = h|\theta)d\theta \\ &= \int_0^1 P(\theta)\theta d\theta \\ &= \frac{\alpha_h}{\alpha_h + \alpha_t}. \end{aligned} \tag{21}$$

This supports our intuition that the Beta distribution corresponds to having seen α_h heads and α_t tails (imaginary or real). *Second* , as we get more data, specifically we get a data set D with h heads and t tails, we have

$$P(\theta|D) = Beta(\theta|\alpha_h + h, \alpha_t + t). \tag{22}$$

Again, this supports our intuition.

Staring from a rather uniform ("flat") distribution, the Beta distribution will become more and more focused around some certain value. The use of an entire distribution rather than a single number to model the parameter θ has the advantage that if reflects not only our current estimate for the value of θ, but also our degree of confidence about that. This is very important in cases where the answers to our queries are very sensitive to the value of θ.

4.1.2 Parameter learning for a fixed belief network

Basically, the parameters to be determined in a belief network can be treated simply as a bunch of individual coins: one for each variable X_i and each assignment of values to its parents. For example, in the simple network $A \to B$ we have three coins to toss, if both A and B are assumed

to be binary: one of A, one for $B|a$, and one for $B|\bar{a}$. We treat each one as a separate coin, and estimate its value separately.

For each coin, the parameter process can be summarized as follows:

- 1. Start from a prior Beta distribution.
- 2. Gain more data as evidence.
- 3. Obtain the *a posteriori* distribution, which is also Beta.
- 4. Go to 2.

4.2 Adaptive Probabilistic Networks

In this section, we describe for problem 2, which is adaptive probabilistic networks with hidden variables, the gradient ascent method proposed in [1].

Suppose we are given a belief network with fixed structure and possible hidden variables. The initial values for the CPTs might be set randomly and data are generated independently from some underlying distribution. The objective here is to find the CPT parameters $\mathbf{w}$ that best model the data. The parameter $\mathbf{w}$ is actually a 3-dimensional matrix, with its element w_{ijk} defined as the probability that variable X_i takes on its j^{th} possible value assignment given its parents U_i takes on their k^{th} possible value assignment, or

$$w_{ijk} = P(X_i = x_{ij}|U_i = u_{ik}). \tag{23}$$

Usually, a maximum *a posteriori* (MAP) analysis assumes a prior distribution $P(\mathbf{w})$ on the network parameters and adjusts $\mathbf{w}$ to maximize $P_{\mathbf{w}}(D)P(\mathbf{w})$. Here we assume that each possible setting of $\mathbf{w}$ is equally likely *a priori* , so that the *maximum likelihood* (ML) model is appropriate. This means that the aim is to maximize $P_{\mathbf{w}}(D)$, the probability assigned by the network to the observed data D when the CPT parameters are set to $\mathbf{w}$. Since $\mathbf{w}$ consists of conditional probability values, we have $w_{ijk} \in [0, 1]$. Further, in any CPT, the entries corresponding to a particular conditioning case (an assignment of values to the parents) must sum to 1, i.e., $\sum_j w_{ijk} = 1$. Those are constraints for the optimization problem. To this end, we can write the problem as:

$$\begin{cases} \max_w P_{\mathbf{w}}(D) \ or \ \max_w \ln P_{\mathbf{w}}(D) \\ s.t. \ w_{ijk} \in [0, 1], and \\ \sum_j w_{ijk} = 1 \end{cases} \tag{24}$$

This is essentially a constrained optimization problem and gradient ascent methods can be applied to solve this problem. One way to cope with the constraints is to use Lagrange multiplier method, which projects $\nabla\mathbf{w}$ onto the constraint surface. An alternative method, commonly used in statistics, is to re-parameterize the problem so that the new parameters automatically respect

the constraints on w_{ijk} no matter what their values. In this case, we can define parameters β_{ijk} such that

$$w_{ijk} = \frac{\beta_{ijk}^2}{\sum_{j'} \beta_{ijk}^2}, \tag{25}$$

This can easily be seen to enforce the constraints given above. Furthermore, a local minimum with respect to β_{ijk} is also a local minimum with respect to w_{ijk}, and vice versa.

The usefulness of gradient-based methods depends on the ability to compute the gradient efficiently. This is one of the many keys to the success of gradient descent methods in neural networks, where *back-propagation* is used to compute the gradient of an error function layer by layer with respect to network parameters. Similar situation exists here in belief networks, where the inference algorithm already incorporates all of the necessary computations so that no additional back-propagation is needed. This gradient can be computed locally by each node using information that is available in the normal course of belief network calculations, as shown below.

$$\begin{aligned}
\frac{\partial \ln P_{\mathbf{w}}(D)}{\partial w_{ijk}} &= \frac{\partial \ln \prod_{l=1}^{m} P_{\mathbf{w}}(D_l)}{\partial w_{ijk}} \\
&= \sum_{l=1}^{m} \frac{\partial \ln P_{\mathbf{w}}(D_l)}{\partial w_{ijk}} \\
&= \sum_{l=1}^{m} \frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)}.
\end{aligned} \tag{26}$$

Note that the aim is to find a simple local algorithm for computing each of the expressions $\frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)}$. In order to get an expression in terms of information local to the parameter w_{ijk}, we introduce X_i and U_i by averaging over their possible values:

$$\begin{aligned}
&\frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)} \\
= \; &\frac{\frac{\partial}{\partial w_{ijk}} \left(\sum_{j',k'} P_{\mathbf{w}}(D_l | x_{i,j'}, u_{i,k'}) P_{\mathbf{w}}(x_{i,j'}, u_{i,k'}) \right)}{P_{\mathbf{w}}(D_l)} \\
= \; &\frac{\frac{\partial}{\partial w_{ijk}} \left(\sum_{j',k'} P_{\mathbf{w}}(D_l | x_{i,j'}, u_{i,k'}) P_{\mathbf{w}}(x_{i,j'} | u_{i,k'}) P_{\mathbf{w}}(u_{i,k'}) \right)}{P_{\mathbf{w}}(D_l)}.
\end{aligned} \tag{27}$$

Observe that the important property of this expression is that w_{ijk} appears only in one term in the summation: the term for $j' = j, k' = k$. For this term, $P_{\mathbf{w}}(x_{i,j'}|u_{i,k'})$ is just w_{ijk}, so we have

$$\begin{aligned}
\frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)} &= \frac{P_{\mathbf{w}}(D_l | x_{i,j}, u_{i,k}) P_{\mathbf{w}}(u_{i,k})}{P_{\mathbf{w}}(D_l)} \\
&= \frac{P_{\mathbf{w}}(x_{i,j}, u_{i,k} | D_l) P_{\mathbf{w}}(D_l) P_{\mathbf{w}}(u_{i,k})}{P_{\mathbf{w}}(x_{i,j}, u_{i,k}) P_{\mathbf{w}}(D_l)} \\
&= \frac{P_{\mathbf{w}}(x_{i,j}, u_{i,k} | D_l)}{P_{\mathbf{w}}(x_{i,j} | u_{i,k})} \\
&= \frac{P_{\mathbf{w}}(x_{i,j}, u_{i,k} | D_l)}{w_{ijk}}.
\end{aligned} \tag{28}$$

This last equation allows us to "piggyback" the computation of the gradient on the calculations of posterior probabilities done in the normal course of probabilistic network operation. Essentially, any standard inference algorithm, when executed with the evidence D_l, will compute the term $P_{\mathbf{w}}(x_{i,j}, u_{i,k}|D_l)$ as a by-product. So we are able to use the inference algorithms mentioned in section 3 as a substrate for the learning system.

Here is the algorithm for adaptive probabilistic network:

---

function APN(N,D) returns a modified probabilistic network

- **inputs:** N, a probabilistic network with CPT entries $\mathbf{w}$ and D, a set of data cases
- **repeat until** $\triangle w \approx 0$
  - $\triangle w \leftarrow 0$
  - **for each** $D_l \in \mathbf{D}$
    - set the evidence in N from D_l
    - for each variable i, value j, conditioning case k

$$\triangle w_{ijk} \leftarrow \triangle w_{ijk} + \frac{P_{\mathbf{w}}(x_{i,j}, u_{i,k}|D_l)}{w_{ijk}}$$

  - $\triangle w \leftarrow$ **the projection of** $\triangle w$ **onto constraint surface**
  - $\mathbf{w} \leftarrow \mathbf{w} + \alpha\triangle\mathbf{w}$
- **return** N

This algorithm is similar to the training algorithm of a neural network with hidden units using back-propagation, and it converges to $\mathbf{w}$ that locally maximizes $P_{\mathbf{w}}(D)$.

4.3 Learning Structures

Like before, we have a data set D and we would like to find a belief network that best explains D. The problem can be divided into two sessions: finding the structure and filling in the parameters for the fixed structure. In this section, we give a brief description of the former session only.

First, we need a *quality measure* based on which a set of candidate networks can be distinguished. Such quality measures are also called *scoring functions*. Bayesian quality measures, such as maximum *a posteriori* (MAP) and maximum likelihood (ML) principle, are quite commonly used [19].

Second, given the scoring function, the goal is simply finding the network that has the highest score. This is a search problem over a combinatorial space—the space of belief networks—of super-exponential size. To do this, we have to define properly the search space, and then choose a good search algorithm.

In order to obtain the best belief network with the highest score, given the search space, the most obvious algorithm is greedy hill-climbing, as described below.

Greedy Belief Network Search

- Pick a random network structure B as a starting point
- Pick parameters for B (using Maximum likelihood, for example)
- Compute score for B
- Repeat
  - Let $B_1, \cdots, B_m$ be the successor networks of B
  - Pick parameters for each B_i
  - Compute score for each B_i
  - Let B' be the highest scoring B_i so far
  - If score(B') > score(B) then $B \leftarrow B'$
  - else return(B)

Like all hill-climbing algorithms, the algorithm above is subject to local maxima. To get around this, some exploration-exploitation trading-offs such as ϵ–greedy methods or guided search exploration schemes, can be incorporated. For details on this, we refer to [20].

Another search algorithm, the so-called K2-algorithm proposed by Cooper and Kerskovits[6], draws much attention. Assuming all nodes are ordered in some way, this algorithm starts with the simplest network, namely a network without links. For each variable X_i, the algorithm adds to its parent set π_i the node that is lowered numbered than X_i. This is repeated until either adding new nodes does not increase the quality or a complete network is attained.

5 Conclusions

In this report, we describe several important aspects of belief networks: definition, semantics, inference and learning. In a word, we can say that a belief network is a good framework to integrate experts' prior knowledge and statistical data and it constitutes a model of the *environment* rather than, as in many other knowledge representation schemes, a model of the reasoning process. The contributions of belief networks can be summarized as follows [2]:

- Natural and key technology for diagnosis
- Foundation for better, more coherent expert systems
- Supporting new approaches to planning and action modeling
  - planning using Markov decision processes
  - new framework for reinforcement learning
- New techniques for learning models from data, see also [8].

Note that the introduction above is by no means complete or exhaustive. It is supposed to provide background knowledge on what a belief network is, what it can do and how. For more information in this area, we refer to [3][4][11][14].

References

[1] J. Binder, D. Koller, S. Russell, K. Kanazawa, " Adaptive Probabilistic Networks with Hidden Variables." *Machine Learning*, in press, 1997.

[2] J. S. Breese and D. Koller, "Bayesian Networks and Decision-Theoretic Reasoning for Artificial Intelligence", Tutorial for AAAI'97, 1997

[3] E. Castillo, J. M. gutierrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models,* Springer, 1997

[4] E. Charniak, "Bayesian Networks without Tears", *AI Magazine,* 12, 4 , pp. 50-63, 1991

[5] G. F. Cooper, "Computational complexity of probabilistic inference using Bayesian belief networks(research notes)", *Artificial Intelligence,*42, pp. 393-405, 1990

[6] G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data", *Machine Learning,* 9, 309-347, 1992

[7] P. Dagum and M. Luby, "Approximately probabilistic reasoning in Bayesian belief networks is NP-hard", *Artificial Intelligence,*pp. 141-153, 1993

[8] N. Friedman, M. Goldszmidt, D. Heckerman, and S. Russell, "Challenge: Where is the Impact of Bayesian Networks in Learning?", Technical Report, 1997

[9] D. Heckerman and M. Wellman, "Bayesian Networks", *Communications of the ACM,* vol. 38, no. 3, pp. 27-30, 1995

[10] D. Heckerman, "A tutorial on learning with Bayesian networks", Microsoft Research Technical Report MSR-TR-94-09, 1996

[11] F. V. Jensen, *An Introduction to Bayesian Networks*, UCL, 1997

[12] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems(with discussion)", *Journal Royal Statistical Society,*Series B 50 (2), pp. 157-224, 1988

[13] J. Pearl, "Fusion, Propagation, and Structuring in Belief Networks", *Artificial Intelligence,* vol. 29, pp. 241-288, 1986

[14] J. Pearl, *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference,* Morgan Kaufmann, 1988

[15] J. Pearl, "Belief Networks Revisited", Technical Report, 1994

[16] J. Pearl, "Bayesian Networks", Technical Report, 1995

[17] J. Pearl, "Bayesian Networks", to appear in *MIT Encyclopedia of the Cognitive Science* , 1997

[18] S. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach,* Prentice-Hall, 1995

[19] S. E. Shimony and E. Charniak, "A New Algorithm for Finding MAP Assignments to Belief Networks", in *Uncertainty in Artificial Intelligence 6* , Elsevier Science Publisher,pp. 185-193, 1991

[20] S. Thrun, "The role of exploration in learning control", in: D.A. White and D.A. Sofge, eds., *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 527-559, Van Nostrand Teinhold, New York, 1992