RELEVANCE BASED PROPAGATION IN BEYESIAN NETWORKS

Javiar Andres Pinto

June 1986
CSD-860098

UNIVERSITY OF CALIFORNIA

Los Angeles

Relevance Based Propagation in Bayesian Networks

A thesis submitted in partial satisfaction of the

requirement for the degree of Master of Science

in Computer Science

by

Javier Andres Pinto

1986

The thesis of Javier Andres Pinto is approved.

Norman C. Dalkey

Norman Dalkey

Richard E. Korf

Richard Korf

[illegible]

Judea Pearl, Chair

University of California, Los Angeles

1986

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT OF THE THESIS

Relevance Based Propagation in Bayesian Networks

by

Javier Andres Pinto

Master of Science in Computer Science

University of California, Los Angeles, 1986

Professor Judea Pearl, Chair

Bayesian Networks have been proposed as a mechanism for representing causal and inferential knowledge, and for providing a computational model for updating beliefs to reflect evidential information. The main theme of this thesis is improving the efficiency of this scheme by focusing its activity towards a predefined *target* hypothesis. All activities are regulated by parameters which measure their degree of relevance relative to the target hypothesis.

These parameters provide measures for: (1) determining which sensory node is the most important to test; (2) evaluating the impact of a single message on the target node; (3) evaluating a particular instantiation of a sensory node and (4) determining the maximum potential effect that any node can have on the target node.

Based on these measures, we propose a mechanism to thwart the propagation of irrelevant information. To do so, we define a *relevance region* that contains only nodes that may become relevant for assessing the target node, and establish a distributed mechanism for dynamically adjusting the boundaries of this region. Finally, we describe the implementation of *BAYNET*, a software tool for the graphical design of Bayesian networks.

CHAPTER 1

Introduction

A central problem in Artificial Intelligence research involves making the process of reasoning precise enough to allow the design of *automatic reasoning* systems. These systems often make inferences and draw conclusions from inaccurate and incomplete sources of information.

Many formalisms have been devised for *reasoning under uncertainty.* They have been applied in the development of decision support systems, most notably in expert system technology, which is the area of AI that has aroused the most interest in the industrial community in recent years. These systems normally deal with domains where the degree of complexity is too high to permit deterministic analysis and where *probabilistic reasoning* is more appropriate.

A widely used approach for representing inferential knowledge are the influence or causal networks, along with some kind of calculus to evaluate the evidence that impinges on the different hypotheses. This thesis is centered on the Bayesian Networks approach, with a calculus based on a formal probability theory.

A Bayesian Network (BN) is a directed graph in which the nodes represent propositions (or discrete variables) and the links represent direct causal relationships between the linked propositions (or variables). For example, let us consider the following three rules: "*If* the patient has the 'flu' *then* he/she will be sneezy," "*If* the patient has the 'flu' *then* he/she will have fever" and "*If* the patient has 'hay-fever' *then* he/she will be sneezy." From this small set of rules we can form the small Bayesian network of Figure 1.1, where node F denotes the proposition "The patient has the flu," H represents the proposition "The patient has hay-fever," S denotes "the patient is sneezy" and V denotes "the patient has fever." F, H, S and V are treated as propositional variables; each may attain the value of either true or false. If we needed to be more precise with respect to the representation of the fever node, we could make the continuous variable, the *patient's temperature*, discrete. The new variable would denote the range of possible temperatures of the patient.

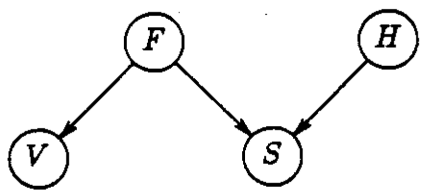


Figure 1.1: A Bayesian Network.

Normally, rules of the type used in the last example are not deterministic, e.g., it is not always true that a person with hay-fever will be sneezy. Thus, we need to introduce some measure of the uncertainty of the rules. In Bayesian networks this uncertainty is expressed by associating weight with the causal relation between nodes. These weights represent the conditional probabilities of a variable, given its causal antecedents. In our example we have to specify the probabities $P(S|H,F)$ and $P(V|F)$. For nodes without ancestors, the prior probabilities of their respective propositions are provided, e.g., we specify the prior probability that a patient has flu and the prior probability that a patient has hay-fever (i.e., $P(F)$ and $P(H)$).

Having defined a BN as a representation of the causal knowledge in a particular domain, we will normally be interested in confirming or refuting a single proposition called the target hypothesis. For example, the target could stand for the existence of a severe disease, the occurrence of a major event, etc.

The propagation of evidence in a BN usually involves updating the belief parameters of every node in the network after it receives a message from one of its neighbors. In this standard approach, updating takes place at every node in the network upon receiving a message, even if the effect of the new update on the target hypothesis is negligible. What is proposed here is to propagate new information only if its net effect on the belief of our target hypothesis surpasses a certain threshold; otherwise, the information is considered irrelevant and the propagation is postponed until additional information is gathered.

We define the *information value* of a node in terms of the impact that information regarding that node alone will have on the *belief vector* associated with the target node. On every node we will maintain *impact parameters*, which will be updated upon receiving new information. If the impact of this new information surpasses a certain threshold, it is propagated to the node's neighbors. We are interested in preserving the local nature of the computation, as is done in ordinary belief propagation, so that the impact parameters too will be updated only on the basis of information from the neighbors.

Since these impact parameters permit us to predict what effect future findings would have on the belief vector of the target hypothesis, they are also useful for deciding where data-gathering efforts should be concentrated. In medical diagnosis, for example, having identified a particular disease as our target hypothesis, we would be able to tell which evidence (medical test) is the most significant in order to confirm or refute the target hypothesis.

This thesis defines the impact parameters, and establishes a distributed scheme for updating their values.

This work adds two valuable features to the Bayesian Network technique of handling uncertainty. First, it reduces the computation involved in the evidence propagation phase by curtailing the propagation of irrelevant information. For example, in a large knowledge base containing models of different diseases, we may have models for many loosely connected diseases. In this type of system we will need some mechanism for narrowing down the reasoning to relevant information or else, every new piece of information will trigger updating of the entire system. For instance, if a patient is being treated for lung problems, we need to bring into focus information related to lung diseases and should ignore all information related to, say, knee pain. This will allow us to focus our reasoning resources on the relevant portion of the knowledge base, discarding unconnected information. This approach is especially beneficial in systems consisting of many loosely coupled subsystems.

Second, it facilitates a more efficient data acquisition process by focusing the queries on the most informative sources of evidence.

Chapter 2 provides a review of several alternative approaches to handling uncertainty, and discusses some of their advantages and disadvantages. Chapter 3 reviews the most important characteristics of Bayesian networks. The reader familiar with the topic may skip directly to chapter 4, where we discuss the extensions introduced to the Bayesian network formalism. In chapter 5 we discuss some of the issues regarding the implementation of the proposed scheme.

CHAPTER 2

Reasoning under Uncertainty

Reasoning under uncertainty amounts to drawing of conclusions from incomplete information utilizing uncertain rules of inference. As an example, given a rule *If A and B then C* in which both, the antecedents, A and B, and the rule itself are uncertain, we need to estimate how much we can commit ourselves to the conclusion C, given the degree of belief we have commited to A and B, and the confidence *Conf.* we have assigned to the rule[1].

Reasoning under uncertainty has become an important issue in current AI applications, and one around which much controversy has been raised. While some researchers advocate the use of Bayesian calculus, others have criticized it, arguing that it requires huge amounts of data or making unrealistic assumptions about the probability distributions used. New ad-hoc approaches have been introduced to overcome these particular limitations. Still others consider all approaches based on numeric degrees of belief to be "restricted because the set of numbers is not a sufficiently rich representation to support considerable heuristic knowledge about uncertainty and evidence" [Cohe83, p. 18]. In this chapter we

[1]At this point we are not making any assumptions about the formal meaning of *Conf.*, it can be interpreted as a conditional probability matrix $P(C|A,B)$, as a certainty factor, or in any form consistent with its conceptual meaning, which is a measure of the reliability of the rule.

will review some of the approaches being used to handle uncertainty, highlighting the most relevant characteristics of each. We will first describe the problem in more formal terms and provide a simple model based on Bayes rule[2].

2.1 Probabilistic Reasoning.

Let $H = \{h_1, h_2, ..., h_n\}$ be a set of hypotheses and $S = \{s_1, s_2, ..., s_m\}$ a set of findings that convey relevant information regarding one or more components of H. To simplify the analysis and without loss of generality, we assume all variables to be binary. We will define a belief vector in h_i as:

$$\underline{Bel}(h_i) = [Bel(h_i), Bel(\neg h_i)] = [P(h_i|S), P(\neg h_i|S)]. \tag{2.1}$$

where $Bel(h_i) + Bel(\neg h_i) = 1$. Our primary interest is to assess the value of a subset of hypotheses H_s of H based on the information available. The hypotheses included in H_s are determined by the problem domain and by the context defined by S. Since the acquisition of information is normally done incrementally, we need to provide a calculus for updating the belief vector of the hypotheses whenever new data arrives. This problem has been extensively studied in the domain of medical diagnosis [Szol78,Spie85], where we can associate the hypotheses with diseases or disease classes and the findings with symptoms. The belief vector $Bel(h_i)$ is obtained by using Bayes' formula:

$$P(h_i|S) = \frac{P(h_i)P(S|h_i)}{P(S)} \tag{2.2}$$

[2]The following presentation is based in the so called "idiot's Bayes" and on the formalization made in [Char83].

which suggests that, to compute $P(h_i|S)$, we would need to know the parameters $P(S|h_i)$, that is, $n(2^m - 1)$ parameters (plus the n prior probabilities of the hypotheses), which would be normally impossible to obtain. A significant reduction in the number of parameters occurs when conditional independence holds, allowing us to write:

$$P(S|h_i) = \prod_j P(s_j|h_i), \tag{2.3}$$

thus, reducing the number of parameters to mn plus the n prior probabilities $P(h_i)$. We do not need to worry about the parameter $P(S)$ in equation (2.2), since it can be regarded as a normalizing constant, which can be determined from the condition $\sum_{h_i} P(h_i|S) = 1$. The assumption of conditional independence of the findings given the hypothesis has been severely criticized e.g. "the assumption of conditional independence is usually false" [Szol78]. However, as Charniak pointed out [Char83], the absence of conditional independence of the findings given the hypotheses can often be removed by introducing intermediate states and causal reasoning, a technique not foreign to AI applications (see, for example, [Weis78]).

Whenever a new piece of evidence s_{m+1} arrives, the belief vector can be updated as follows:

$$P(h_i|S, s_{m+1}) = \alpha P(S|h_i)P(s_{m+1}|h_i) \tag{2.4}$$

where α is the normalizing factor. An approach often utilized is to take the logarithm of this equation, which yields:

$$\log(P(h_i|S, s_{m+1})) = \alpha' + \log(P(S|h_i)) + \log(P(s_{m+1}|h_i)) \qquad (2.5)$$

Now, $\log(Bel(h_i))$ can be interpreted as the sum of the weights of previous evidence supporting h_i and the weight of new evidence (i.e., $\log(P(s_{m+1}|h_i))$); α' can be left out since it appears as a constant in the computation of the weight of every hypothesis.

This simple probabilistic approach permits ranking the hypotheses according to the weights given to them by the available evidence. It does not handle the situations where one hypothesis can be considered as evidence towards derivation of another hypothesis or when the evidence itself has not been observed with certainty. These situations, and the propagation of evidence through chains of inferences, will be discussed under the topic of inference networks.

2.2 Inference Networks

An *inference network* is a graph in which the nodes represent propositions (either evidence or hypotheses) and the arcs represent elastic antecedent-consequent implications (i.e., *if-then* inference rules). As an example, the inference net of figure 2.1 represents three evidence nodes (S_1, S_2 and S_3), an intermediate node (I_1) and one hypothesis node (H_1).

Each proposition in the network has an associated degree of belief, which is the conditional probability of the proposition, given all the evidence at hand.

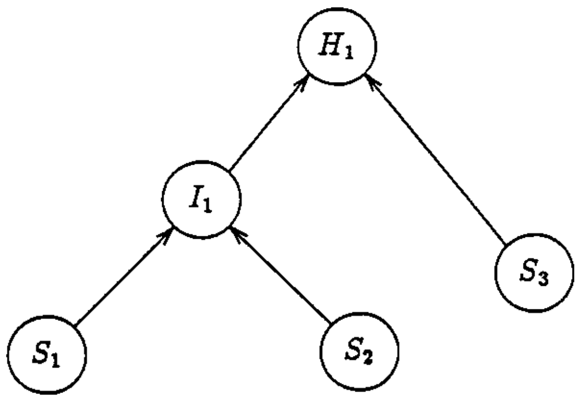


Figure 2.1: A simple Inference Network.

If we receive evidence supporting S_1, we will update our degree of belief in I_1, and from there we will propagate the update to node H_1. In this section we will describe the mechanism by which these belief parameters are updated.

2.2.1 Belief Computation

Consider a two node inference network $E \rightarrow H$. Using Bayes' formula, we can derive:

$$\frac{P(H|E)}{P(\neg H|E)} = \frac{P(E|H)}{P(E|\neg H)} \cdot \frac{P(H)}{P(\neg H)} \tag{2.6}$$

or

$$O(H|E) = \lambda \cdot O(H) \tag{2.7}$$

where $O(H) = \frac{P(H)}{P(\neg H)}$ and $O(H|E) = \frac{P(H|E)}{P(\neg H|E)}$ stand respectively for the prior and posterior odds of H, and $\lambda = \frac{P(E|H)}{P(E|\neg H)}$ is the likelihood ratio of E given H. This last equation tells us how to update the odds of H given that we observed E to be true; thus, the relation $P(H|E) = \frac{O(H|E)}{O(H|E)+1}$ gives the updated belief in H.

Now, if we observed that E was false, the update can analogously be done by:

$$O(H|\neg E) = \neg\lambda \cdot O(H). \tag{2.8}$$

where $\neg\lambda = \frac{P(\neg E|H)}{P(\neg E|\neg H)}$. This approach assumes that we have the values of λ and $\neg\lambda$ available. Their interpretation is that a high value for λ implies that the observation of E strongly suggests H; on the other hand, a low value for $\neg\lambda$ would mean that E is a necessary piece of evidence to conclude H. In the literature [Duda76], λ is also called *measure of sufficiency* and $\neg\lambda$ is called *necessity measure.* An inference rule is then specified as:

$$\text{If } E, \text{Then (to degree } \lambda, \neg\lambda) \ H \tag{2.9}$$

in which the values for λ, $\neg\lambda$ and $O(H)$ are supplied by the designer of the inference network model.

2.2.2 Combining and Propagating Evidence.

To illustrate the case where we have more than one piece of evidence impinging on a hypothesis node, let us assume we have n evidence nodes $E_1, ..., E_n$, all linked to the hypothesis node H. To handle this case we use a strategy analogous to the one described in the previous section. If we can make the assumption of conditional independence among the evidence nodes, given the hypothesis, then:

$$P(E_1, ..., E_n|H) = \prod_{i=1}^{n} P(E_i|H), \tag{2.10}$$

which gives:

$$O(H|E_1, ..., E_n) = \Lambda O(H), \tag{2.11}$$

where, by the assumption of conditional independence,

$$\Lambda = \frac{P(E_1, ..., E_n|H)}{P(E_1, ..., E_n|\neg H)} = \prod_{i=1}^{n} \lambda_i, \tag{2.12}$$

Similarly, we have:

$$O(H|\neg E_1, ..., \neg E_n) = \prod_{i=1}^{n} \neg\lambda_i \cdot O(H). \tag{2.13}$$

The posterior odds of other combinations of positive and negative evidence are handled analogously by taking the product of the corresponding combination of their positive and negative lambda parameters and the prior odds.

Finally, we need to specify how evidence is propagated through chains of inference. Suppose that we have the chain $E \rightarrow I \rightarrow H$ and that we have determined $P(I|E)$ and $P(\neg I|E)$. The question is how do we determine $P(H|E)$. From probability theory we know that:

$$P(H|E) = P(H|E, I)P(I|E) + P(H|E, \neg I)P(\neg I|E). \tag{2.14}$$

In a causal chain of this type it is assumed that the effect that evidence E has on the hypothesis H is indirect through the intermediate node I. Hence, knowing the value of the intermediate node I we may assume that the influence of E on H is blocked. From a probabilistic point of view this assumption may be translated as $P(H|E, I) = P(H|I)$. If this assumption holds, we may write:

$$P(H|E) = P(H|I)P(I|E) + P(H|\neg I)P(\neg I|E). \tag{2.15}$$

Now we can straightforwardly determine the effective likelihood ratio λ' as:

$$\lambda' = \frac{O(H|E)}{O(H)}. \tag{2.16}$$

The relation of equation 2.16 permits to obtain the effective likelihood ratios λ_i' and $\neg\lambda_i'$ corresponding to an uncertain piece of evidence E_i. This likelihood ratios can be used in 2.12 and 2.13 to compute the probability vector of a hypothesis, given the evidence nodes.

To illustrate the update and propagation scheme, we will use the inference network of figure 2.1 with the following parameters: $P(H_1) = 0.5$; $P(I_1) = 0.7$; $\lambda_{S_1} = 4$; $\neg\lambda_{S_1} = 0.25$; $\lambda_{S_2} = 6$; $\neg\lambda_{S_2} = 0.44$; $\lambda_{S_3} = 0.42$; $\neg\lambda_{S_3} = 2.3$; $\lambda_{I_1} = 4$; $\neg\lambda_{I_1} = 0.25$. Furthermore, assume that we have established that S_1 is known to be true and that S_2 and S_3 have both been denied. Under these circumstances, we can determine:

$$O(I_1|S_1,\neg S_2) = \lambda_{S_1}\neg\lambda_{S_2}O(I_1) = 4.1 \tag{2.17}$$

$$\Rightarrow P(I_1|S_1,\neg S_2) = 0.804 > P(I_1) \tag{2.18}$$

$$O(H_1|S_1,\neg S_2,\neg S_3) = O(H_1)\neg\lambda_{S_3}\lambda_{I_1}\left(\frac{P(I_1|S_1,\neg S_2) - P(I_1)}{1 - P(I_1)}\right) \tag{2.19}$$

$$= 1 \cdot 2.3 \cdot 4 \cdot \left(\frac{0.804 - 0.7}{1 - 0.7}\right) = 3.19 \tag{2.20}$$

$$\Rightarrow P(H_1|S_1,\neg S_2,\neg S_3) = 0.76. \tag{2.21}$$

If we had an uncertain observation (related to a terminal or leaf node), we could treat it as an intermediate node and consider $P(S|E)$ equal to the probability of S being true. This amounts to a straight interpolation between $P(H|S)$ and $P(H|\neg S)$.

Sometimes a problem of overspecification may arise. From equation 2.15 we may derive:

$$P(H) = P(H|I)P(I) + P(H|\neg I)P(\neg I), \tag{2.22}$$

which gives an expression to determine $P(H)$ in terms of $P(I)$. If we elicit these prior probabilities from the user (through the assessment of $O(H)$ and $O(I)$), they in general will not obey the relation 2.22. One approach to solve this problem is to use a linear interpolation function [Duda76] (this solution will introduce inconsistencies with the probabilistic relationship 2.15). For example, to determine $P(H|E)$, we can take a linear interpolation between $P(H|\neg I)$ and $P(H)$, or between $P(H)$ and $P(H|I)$ depending on whether the evidence tends to decrease or increase the belief in H.

2.2.3 Some Drawbacks of the Inference Network Formalism.

The inference network paradigm has been successfully utilized in a number of applications, notably in *PROSPECTOR*, which is a computer-consultant system intended to aid geologists in the evaluation of the favorability of an exploration site or region of finding mineral deposits of a particular kind [Duda79]. However, this approach entails certain inconveniences:

1. Overspecification of parameters: The model designer often wishes to specify the prior probabilities on the intermediate nodes. These probabilities will not necessarily be consistent; the prior probability of a hypothesis node

computed using (2.15) in the abscence of evidence (i.e. $P(I|E) = P(I)$) will generally not agree with that derived from the inference network.

2. The parameters λ and $\neg\lambda$ are not totally independent. The definition of λ and $\neg\lambda$ is such that, if either is less than one, then the other must be greater than one. In principle, it is not possible to define these parameters so that the presence of an evidence would strongly suggest a hypothesis, while its absence would remain neutral between its hypothesis and its negation ($\neg\lambda = 1$ implies $\lambda = 1$). This is also true in Bayesian networks, where both, the negative and the positive instantiations of a variable have effect over the related propositions.

3. No predictive inferences. In *PROSPECTOR* the propagation is done one-way, from evidence to hypotheses. The belief in any intermediate hypothesis is calculated based on information coming from its descendants (diagnostic), ignoring the support gathered by its parents (prospective). This results in erroneous conclusions in cases where the target hypotheses are not peripheral (i.e. intermediate) and, moreover, it prevents us from properly discriminating between significant and insignificant sources of evidence – a sensory node is judged to be significant when it may lead to high-impact outcome, even though the likelihood of such outcome is extremely low.

2.3 Causal-associational Networks (*CASNET*).

CASNET [Weis78] has been developed as a general approach to structure medical knowledge for computer-aided diagnosis and therapy. It has been used to construct a diagnostic system for the disease glaucoma. A *CASNET* model consists of a graph decomposed into three levels, the nodes in the bottom level representing *observations*, which can be either symptoms, signs or test results. The middle level, on which most of our discussion will be centered, is composed of intermediate pathophysiological states, which describe internal conditions assumed to take place in the patient. At the top level are the nodes related to disease categories, in which patterns of states and observations are summarized.

Nodes in the middle level are linked together in a *causal network*, which is a directed acyclic graph where each directed link represents a non-strict cause-effect relationship.[3] A disease process is characterized by a path in the graph; progression and severity of a disease corresponds to a progression along a directed path in the causal network. The strength of a cause-effect link is quantified by a frequency measure, which can be interpreted as the conditional probability of the ocurrence of the effect, given the cause. Starting states in the causal graph are assigned a starting frequency, which can be interpreted as a prior probability.

The middle level of pathophysiological states is linked to the bottom level of observations by a set of *associational links*, connecting observation nodes (or

[3]In a non-strict cause-effect relationship, a cause may be present without any of its effects occurring at the same time.

logical combinations among them) to the state about which they are providing evidence. These links are quantified by a *confidence measure* Q, which is a number between -1 and 1. Q_{ij} is the degree of confidence in the state j as a consequence of having observed i alone. Negative (positive) values in the confidence measure indicate decreased (increased) confidence in the state node due to the presence of the observation node.

The association between the top level of disease categories with the middle level of pathophysiological states is one of pure classification. States in the middle level are connected to disease categories in the top level by classification links, which can be represented by the triple (n, D, T), meaning that disease D is present whenever a causal path terminates in n, and T is a set of therapy recommendations for the patients that fall in the disease category D.

2.3.1 Measure of Confidence and Interpretation of the Causal Network

Every pathophysiological state n_j in the middle level has an associated *confidence measure*, which is initially set to zero. When the first test t_i with an associational link to n_j is obtained, the confidence measure is set to the link's confidence measure, i.e., $Cf(n_j) = Q_{ij}$. This measure is updated whenever new test results arrive, following the criteria that $Cf(n_j)$ remains the same until a more confident test appears, i.e., a test such that $|Q_{kj}| > |Cf(n_j)|$. If we receive a test with a magnitude of confidence equal to $Cf(n_j)$ but with opposite sign, we set $Cf(n_j)$ to zero and annotate conflict between tests, conflict that is resolved when a new and more confident test (i.e., with a higher magnitude of confidence measure) is available.

When the measure of confidence is over a certain positive threshold T, the state is assumed to be confirmed; otherwise, it is assumed to be denied. In all other cases, the node is undetermined.

A pathway in a causal nework is a directed path in the graph, whose starting node corresponds to a starting state (i.e., a state with no defined causes). Thus, the starting state represents the original cause of the disease of the patient. A pathway is considered admissible if it contains no denied nodes. For any given configuration of confirmed and denied nodes, there are many possible starting nodes, the most likely is chosen to be the one that explains the greatest number of confirmed states. If there is a tie beween two or more starting states, a likelihood is computed (based on the inverse weight, as explained later on), and the most likely state is chosen. The last confirmed state (in the direction of causality) determines the disease category from the top level of the network and once the disease category has been identified, the treatment can be chosen. The system permits the user to ask for different possible pathways or to identify the most probable progression of the disease based on the most likely pathways in the causal network.

2.3.2 Test Selection Criteria.

For every node n_j, in the middle level of pathophysiological states, two parameters are defined: forward weight and inverse weight, which measure the likelihood of the node based on diagnostic support and on its causal consequents respectively. Every admissible pathway leading to n_j contributes to the forward weight $w_F(j)$ of n_j by an amount equal to the product of the frequency measures of the links along that pathway, starting at the closest confirmed node, if any, or at the starting node, which contributes to the forward weight with its starting frequency. The total forward weight of n_j is the sum of these partial weights. For example, in the causal graph of figure 2.2, if node n_2 has been confirmed and all others are undetermined, the total forward weight of n_6 would be:

$$w_F(6) = w_F(6|4) + w_F(6|2) = 0.5 \cdot 0.9 + 0.5 \cdot 0.3 \cdot 0.1, \tag{2.23}$$

where the starting frequence of n_4 is considered to be 0.1.

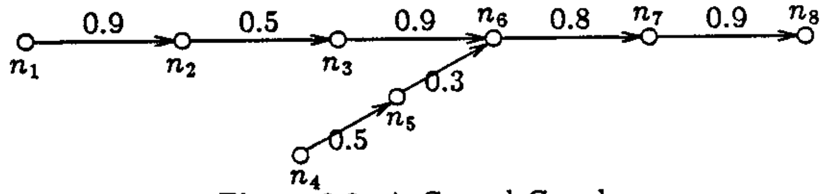


Figure 2.2: A Causal Graph

The inverse weight of node i with respect to node j is defined as:

$$w_I(i|j) = \frac{(w_F(j|i) \cdot w_F(i))}{w_F(j)}, \tag{2.24}$$

where $w_F(j|i)$ is the forward weight of n_j, considering that n_i has been confirmed. $w_F(j|i)$ is computed ignoring all confirmed nodes. Hence, this measure can be interpreted as the contribution of n_i to the weight of those pathways ending in n_j. The inverse weight $w_I(i)$ of node i is defined as:

$$w_I(i) = \max_j (i|j).^4 \tag{2.25}$$

In the example of figure 2.2 we compute the inverse weight of n_6 as:

$$w_I(6) = w_I(6|8) = w_F(8|6) \cdot w_F(6)/w_F(8) \tag{2.26}$$

A special case in the computation of the inverse weight of a node occurs whenever the node is on all the pathways to a confirmed node; the inverse weight is then set to one. For example, if n_7 were confirmed, then $w_I(6)$ would be one; on the other hand $w_I(5)$ would still be computed using equation 2.24. This is intuitively reasonable since, according to the causal model, n_7 could not possibly be confirmed if n_6 were denied.

Finally, the likelihood of a node is determined as the maximum between the forward and the inverse weights, i.e., $W_i = \max(w_F(i), w_I(i))$. W_i is interpreted as the degree to which we can expect n_i to be confirmed or disconfirmed, based on the causal evidence available from the network.

The previously defined weight determines the criteria by which a test is selected. Tests are assumed to be made incrementally. At any given point, the test that conveys the most confident information regarding the node with the greatest weight is chosen. The idea behind this criterion is that an unconfirmed node with a great chance to be confirmed (or denied) should be tested first.

[4]The maximum corresponds to the strongest support received from the causal consequences of n_i.

Besides the test selection criterion described in the previous paragraph, we can also have a *hypothesis driven* test choice. In this selection, the attention is focused upon pathways that could explain the patients' observations, even though the nodes in this pathways are undetermined.

2.3.3 Comments on *CASNET*

The handling of evidence, the confirmation and denial of nodes and the test selection deserve some attention. We notice, for instance, that the status of a node is determined solely on the basis of test results, without taking into account the support given by possible causes or by confirmed effects. In an extreme case, we could have a cause C and an effect E with strength close to 1 (i.e., $P(E|C) \approx 1$), have C confirmed and, still, E would not be considered confirmed until some test renders it confirmed. In fact, the test selection criterion would select this node to be tested first. The situation just described arises because of the separation of support given to a node by its causal relationship with neighboring nodes and the patient's observations. This separation seems artificial and unnecessary since a confirmed possible cause of a node n can give as much support to the belief in n as that given by a test result. In addition, no effect is attributed to partially confirmed hypotheses. These deficiencies can lead to wrong choices of nodes to be tested and, in some cases, to wrong decisions and conclusions. However, the causal graph paradigm is general enough to be adapted to a different mechanism of evidence propagation.

CASNET was implemented utilizing the disease glaucoma as a domain. The level of detail in which the knowledge about this disease is structured permits a very precise reasoning and, as reported by the authors, very accurate results. The applicability of *CASNET* to other domains is limited by the requirement of having a very well structured knowledge of the domain, where the cause-effect relationships among variables are well known. If we overlooked a part of the causal network by omitting links or possible pathways, we would end up with inconsistencies in the results displayed by the state of the network. For instance, we could have confirmed unexplained nodes (i.e., with no admissible pathway leading to them), a situation that can occur due to either erroneous test results or an incomplete causal network. The medical knowledge about glaucoma embodied in the causal network is so precise that it has been argued [Szol78] that *CASNET* could be converted to a categorical reasoning program, avoiding all probabilistic reasoning in the system. The main contribution of this work is the use of detailed causal models, which seems to be a natural way to express diagnostic knowledge in domains like medicine or electronic troubleshooting.

2.4 NESTOR

NESTOR [Coop84] is a computer-aided medical decision making program developed to help in the diagnosis of the diseases that cause hypercalcemia. Its tasks are the generation of a set H of hypotheses that would account for a set of findings given to the system, the ranking of these hypotheses according to their likelihood and an explanation of how well the set H accounts for each one of the findings. The system is designed to support (not to guide) the reasoning of physicians, and can be adapted to its user's requests. It permits, for instance, the evaluation of any hypothesis in terms of how well it explains the findings, or the comparison of two hypotheses on the same grounds.

NESTOR uses causal graphs to structure its domain knowledge, and the cause-effect relationships are quantified by probabilistic parameters.

2.4.1 Measure of Belief in *NESTOR*.

The measure of belief[5] used by *NESTOR* is based on the conditional probability of the hypothesis given all the findings. Bayes formula can be written as:

$$P(H_i|F) = \frac{P(F|H_i)P(H_i)}{P(F)} = \frac{P(F,H)}{P(F)} \tag{2.27}$$

and *NESTOR* computes the degree of belief in a hypothesis as $P(F,H)$. It then avoids the computation of $P(F)$ which "is not practically possible" [Coop84, page 80]. However, since the term $P(F)$ remains constant for every H_i, it then follows that the use of $P(F,H)$ is equivalent to the use of $P(H|F)$, the same reasoning we followed in section 2.1. There is an important difference between the interpretation of the measure of belief used by *NESTOR* and the one used in Bayesian Networks (to be discussed in detail in the next chapter). When *NESTOR* computes $P(H|F)$ for the causal network of figure 2.3, it does not

[5]Cooper uses the term *scoring metric*, which is consistent with our concept of measure of belief or degree of belief. We will use these terms interchangeably.

attribute to it the intuitive causal interpretation: "F indicates H', and H' causes H", since there is no direct causal relationship between the instantiated variable F and H. This kind of interpretation is only possible if evidence is propagated through causal links in both directions.

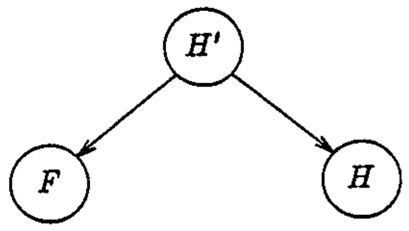


Figure 2.3: A Causal Graph in *NESTOR*

NESTOR represents the belief of an event E as a range in which its probability $P(E)$ might lie, i.e., $P_B(E) = [P_{LB}(E), P_{UB}(E)]$, where $P_{LB}(E)$ is the lower bound of $P_B(E)$ and $P_{UB}(E)$ is the corresponding upper bound. Furthermore, the precision of a range is defined as one minus its length, so that a point range has precision of 1, and a (0,1) range has a precision of zero. According to Cooper, the use of ranges instead of point probabilities is justified because point probabilities have unrealistic precision. The introduction of ranges allows flexible expression of the probabilities used in the system.

Another important characteristic of *NESTOR* is that it does not make indiscriminate global assumptions about the conditional independence of findings relative to the hypothesis. Conditional independence of the effects of a cause is assumed unless some explicit knowledge about their interdependence is available. Knowing that the effects are not independent, permits us to determine lower and upper bounds on the conditional probabilities. The use of these ranges will decrease the accuracy of the probabilities, but it will not produce wrong results.

2.4.2 Causal Graphs in *NESTOR*

NESTOR bases its representation of the knowledge embedded in the system on an acyclic causal graph, which is semantically equivalent to Bayesian networks and to the causal networks used by *CASNET*, i.e., the nodes represent propositions that are causally related to other nodes by the links in the graph, links which are directed from cause to effect. However, the causal network is not limited to pathophysiological states; in fact, nodes in *NESTOR* can represent diseases, etiologies, findings and intermediate states (pathophysiological states). For example, in figure 2.4 we have two findings (f_1 and f_2), two intermediate states (I_1 and I_2) and one etiology node (E). To determine the scoring metric, we use:

$$P(F,E) = P(F|E) \cdot P(E) \tag{2.28}$$

since the prior probability $P(E)$ is assumed to be known, the task of scoring the hypotheses is reduced to computing $P(F|E)$.

To compute $P(F|E)$, the graph is divided into levels, where the level of a node n is defined as the length of the longest path (in the graph) from n to an etiology node. Hence, the first level is occupied by etiology nodes only, the second and higher levels are occupied by intermediate nodes and findings. By definition, any node is at a level greater than the level of its ancestors. Finally,

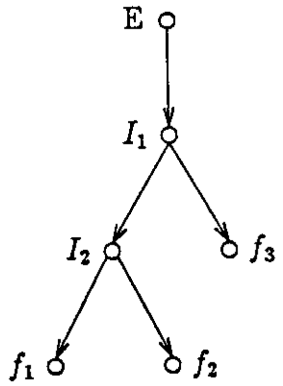


Figure 2.4: A Causal Graph with Intermediate States.

the value of $P(f_1, f_2|E)$ is computed as:

$$P(f_1, f_2|E) = \sum_{I_1,I_2} P(f_1, f_2, I_1, I_2|E), \tag{2.29}$$

in which we consider every possible instantiation of the intermediate nodes I_1 and I_2. Each instantiation is computed as:

$$P(f_1, f_2, I_1, I_2|E) = P(f_1|I_2)P(I_2|I_1)P(f_2|I_1)P(I_1|E) \tag{2.30}$$

where conditional independence of the effects given the causes has been assumed. We also assume that nodes at level j are causally influenced only by nodes at a level less than j, and the computation in (2.26) is referred to as *Probabilistic Causal Simulation.*

There are two levels of links in *NESTOR*, a first level of probabilistic links and a second level of functional links. Probabilistic links relate cause and effect between etiologies and findings and the causal relation is quantified by the conditional probability of the effect given the cause. In the second level, functional links can relate etiologies, findings and intermediate states, which are pathophysiological states or processes not clinically observable. The functional relationship between the cause and the effect can be one of two types, of either a positive or a negative correlation, i.e. if there is a positive (negative) correlation, an increase in the value of the cause variable monotonically increases (decreases) the value of the effect variable. The probabilistic links can only relate a single cause to a single effect. In cases where multiple causation is present, *NESTOR* combines the probability density functions of all the individual links. This combination is based on the correlation among the different causes (considered independently) and the effects.

As an example, consider figure 2.5 in which the two causes S and T are producing the same effect X. Both causes are positively correlated with their common effect (indicated by the "+" sign on the links). Let us consider that all variables can range over three possible values each, *high*, *middle* or *low*. Each probability $P(X = x_i|n = n_j)$ (n being any of the cause nodes) is specified by its upper and lower bounds. Based on this information, the ranges for $P(X|S,T)$ are determined based on the behavior of these probabilities considering the kind of correlation present between causes and effect. For example, the value $P_{LB}(X = high|S,T)$ is determined as:

$$P_{LB}(X = high|S,T) = max(P_{LB}(X = high|S), P_{LB}(X = high|T)), \qquad (2.31)$$

where we assume that both causes acting independently would increase the value of the effect, regardless of the value of the other cause (both causal relations are supposed to be positively correlated). Hence, both causes acting together will increase the value of the effect by at least as much as the strongest cause acting alone. The remaining five bounds are determined analogously.

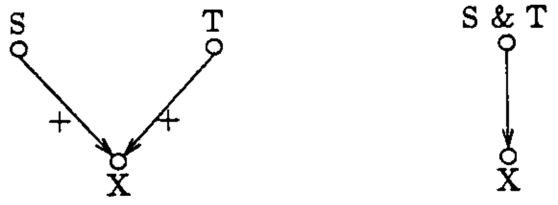


Figure 2.5: Combination of two convergent links.

The combination functions for cases with the same topology of the causal graph in the previous example but with different types of correlation (i.e. $--$, and $-+$ correlation pairs) are also determined in [Coop84]. In the general case, where we can have many links converging into a single effect, *NESTOR* combines all the positively correlated links into one link and all negatively correlated links into another, reducing this case to a two cause, one effect causal graph with a known combination function.

Cooper argues that conditional independence among effects given a common cause might not be a realistic assumption; hence, it considers divergent links as an abstraction of a more complicated lower level causal relationship (see figure 2.6), where additional intermediate states have been introduced. These intermediate states are linked to the rest of the nodes through functional links that are used to derive a combination function to convert divergent links into single links.

A hypothesis in *NESTOR* can be composed of several etiology nodes, i.e., more than one disease can be present at any given time. The question of how to select the hypotheses for evaluation is handled as follows:

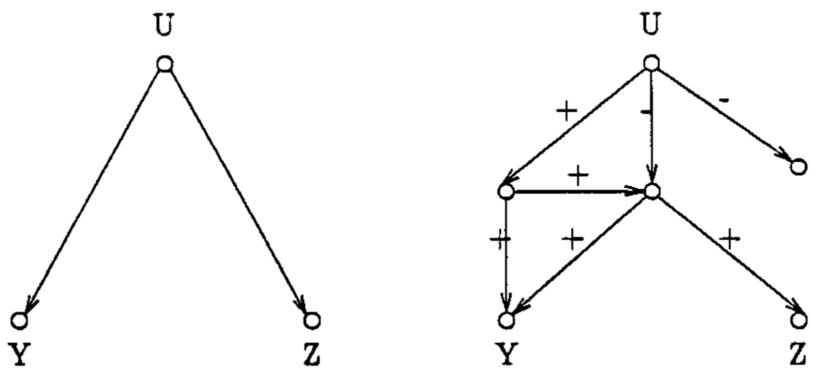


Figure 2.6: Functional links associated with a divergent pair.

As a first step, *NESTOR* considers all possible combinations of etiologies as hypotheses to be scored, although it restricts the etiologies to the ones that are causally connected to the set of findings. The set of hypotheses that can be generated is immense; hence, a heuristic mechanism for reducing the task of scoring them was used. *NESTOR* resorts to admissible optimization techniques that eliminate any hypotheses which, in the evaluation phase, are bound to have suboptimal score.

2.4.3 Comments on *NESTOR*.

The novelty of *NESTOR*'s approach with respect to the ones previously described is its consideration of a two level causal network, with a first level of probabilistic links and a second deeper level of functional links. Functional links are used to avoid making commitments regarding the values of the conditional probabilities associated with them. It then separates the causal knowledge from the probabilistic knowledge, which is limited to the first level of probabilistic links. *NESTOR* also uses ranges to express the probability values, arguing that point probabilities have an unrealistically high precision. In addition, the use of point probabilities is not compatible with the less precise functional links.

The computations yield results identical to those obtained in Bayesian networks, but are conducted in a global manner, considering all possible instantiations of intermediate nodes. In contrast to Bayesian networks, these computations are lengthy, void of conceptual meaning, and do not facilitate saving of partial results, each new finding, requires restarting the computation from scratch.

2.5 Reasoned Assumptions.

As we pointed out earlier in this chapter, there has been a great deal of criticism of all numeric approaches to uncertainty management. In this section we will briefly review the research in the area of *reasoned assumptions* developed by Jon Doyle [Doyl83,Doyl82].

2.5.1 Uncertainty and Reasoned Assumptions.

Doyle argues that the use of numerical assessments for the uncertainty of rules, such as "if A, then with uncertainty X, C" hides the knowledge of those exceptions that have made the rule uncertain. He then continues, saying that we should make explicit those cases in which the rule is not applicable and replace the rule above with the inference rule:

$$A \parallel B \vdash C. \tag{2.32}$$

Equation 2.32 is the basic tenet of this approach and is called a *reason*, where A, B and C are sets of propositions. The interpretation of 2.32 is: If every proposition in A has been concluded and no proposition in B has been concluded, then conclude every proposition in C. The conclusions derived from a set of reasons are called *reasoned assumptions*. For example, we can encode a rule like "*If* the patient is sneezy and does not have fever, *then* assume that the patient has hay-fever" as:

$$\{\text{sneezy}\} \parallel \{\text{has fever}\} \vdash \text{has hay-fever}, \tag{2.33}$$

and we may conclude the reasoned assumption "the patient has hay-fever," provided that we have previously concluded that the patient is sneezy and but not that the patient has fever.

To express uncertainty we may use either *default reasons* or *defeasible reasons*. Default reasons express inferences in which the conclusions may be assumed to hold unless they are proven to be incorrect or their negation had been previously concluded. They are expressed as:

$$A \parallel \neg C \vdash C. \tag{2.34}$$

A defeasible reason R is expressed as:

$$A \parallel \{Defeated(R)\} \vdash C, \tag{2.35}$$

which is interpreted as a reason that can be used to derive C only if the reason itself has not been ruled out (defeated). There are many other kinds of reasons that can be used to express the inferential knowledge (causal or otherwise) in a given domain; for example, premises can be expressed as: $\emptyset \parallel \emptyset \vdash p$, which means that p is always true.

Let S be a set of reasons. An *admissible extension* $AE(S)$ of S is a set $E \supseteq S$ such that, if $A \parallel B \vdash C$ and every element of A is in E and no element of B is in E, then every element of C is in E. Another requirement for E is that, for every element X of E, there is at least one sequence of reasons such that X can be concluded from them and that the sequence is originated from elements in S (i.e., there are no circular arguments for any reasoned assumption).

For every set S of reasons, there may be many different admissible extensions, or there may be no extension at all if we start with contradictory premises. In general, however, we will have many different extensions for any given S. For example, let us consider the following set S of reasons:

$$a \parallel \neg c \vdash c \tag{2.36}$$

$$\emptyset \parallel c \vdash \neg c \tag{2.37}$$

$$\emptyset \parallel \emptyset \vdash a \tag{2.38}$$

which has two possible extensions:

1. $S \cup \{c, a\}$, and
2. $S \cup \{\neg c, a\}$.

Hence, given many possible extensions, we need to decide which is the most satisfying one, i.e., which is the one with the highest probability and which includes the correct assumptions. Doyle assigns a probability measure to each extension (the sum of the probability measures for all extensions equals 1.0). This measure increases monotonically with the inverse of the number of elements in the extension. It then assigns a probability to a proposition *c* equal to the sum of the probabilities of those extensions that contain *c*. It then follows that we can obtain degrees of belief in propositions from a categorical formulation of a model in terms of reasons. Then, Doyle argues that "reasoned assumptions express a more fundamental notion of uncertainty than numerical stipulations of certainty" [Doyl83, p. 42].

2.5.2 Comments.

The technique devised by Doyle is very appealing since it permits a rigorous expression of inferential knowledge in any domain, providing a formal mechanism to handle uncertainty, and is certainly better suited for expressing knowledge without having to make commitments to particular numbers (e.g. conditional probabilities).

Yet, the approach contains major drawbacks. First, just as any categorical reasoning system, this approach requires an extensive and precise knowledge of the problem domain. Even though it is ideal to be provided with such a knowledge, it is generally understood that its codification is extremely difficult and expensive. Moreover, the scoring mechanism used by Doyle seems to be arbitrary, as the assignment of a probability value to an extension in terms of its cardinality may prove inadequate, it does not consider the possibility of assigning some prior preference to certain admissible extensions over others.

We maintain that any system should contain as much of its knowledge as possible codified by this kind of formalism, avoiding, wherever possible, the use of probabilistic reasoning. However, the use of externally provided probabilities cannot be left out and must be included both to score different extensions and to summarize knowledge which we prefer to keep implicit. For example, if we were asked to detail the conditions under which the outcome of a coin will be head, we would probably have to encode a volume full of microscopic molecular interactions over which we have no control. On the other hand, we may use our past experience and our limited knowledge of the area of interest to give a numerical summary of 50% which, for many practical cases, would be quite adequate. We conclude that the use of a probabilistic mechanism for handling uncertainty is still a necessity.

2.6 Summary.

In this chapter we have summarized some of the approaches being used to handle uncertainty. It is not intended as a comprehensive survey of the area, and we have left out some important approaches. However, our selection is broad enough to appreciate the extension of the field, as well as the advantages and possible disadvantages of the particular approach used in this thesis.

CHAPTER 3

Bayesian Networks

In this chapter we will provide a review of the Bayesian Network formalism, emphasizing those aspects considered more relevant to this thesis. A more detailed description of the BN formalism can be found in [Pear85c].

3.1 Introduction

A *Bayesian Network* is a computational device aimed for modelling inferential reasoning of humans using the framework of Probability Theory. The network represents a causal model for the problem domain, where the nodes are multi-valued variables. Each variable represents a set of mutually exclusive hypotheses and/or the possible results of observations. For example, a hypothesis node may stand for the identity of a mineral deposit, and an observation node may be associated with the level of sulfur in a rock sample. The interrelationship between nodes is made explicit by the links connecting them. If the value of node M is known to be directly influenced by the value of node N, we include a directed link from node N to node M. More formally, a Bayesian network is a directed acyclic graph in which nodes denote propositions (or discrete variables) and links represent direct causal relationships between the connected propositions (or variables).

The links point from causes to effects.

The Bayes network formalism provides a mechanism for determining a degree of belief in each one of the propositions represented in the network. We define each component of the ***degree of belief vector*** of a node as:

$$Bel(X_i) \triangleq P(X = X_i|D), \tag{3.1}$$

where D is a set of instantiated variables representing all the evidence available, and X_i is one of the possible values of the variable X. In addition, Bayesian networks provide a scheme of updating the degree of belief when new data is added to the system.

A straightforward way of handling evidential reasoning is to first obtain the joint probability distribution of all the variables in the network and then compute:

$$Bel(X_i) = \frac{P(X_i, D)}{P(D)}. \tag{3.2}$$

However, this approach requires storing and computing a very large number of probability values, each corresponding to the joint probability of some combination of states for all the variables in the knowledge base. This approach is, in general, hard to implement because of the very large number of parameters involved. Alternatively, we can simplify the computation by making explicit use of the dependencies among the variables. A Bayesian network specifies the existence of a direct causal relationship among small subsets of variables, and these relationships are quantified by conditional probability matrices of the effects given the causes. For example, if we have two nodes, E and B, and both can act as causes for node A, we would have the links $E \rightarrow A$ and $B \rightarrow A$ and the conditional probability matrix $P(A|B,E)$. Hence, we are assuming that a given proposition is dependent on the joint ocurrence of its immediate ancestors.

The Bayesian network approach relies in some assumptions of conditional independence which is explicit in the topology of the network. These assumptions are the conditional independence of a set of manifestations given its common cause (independence among the children given its parent) and the independence of two nodes in a directed path given an intermediate node. These notions of independence are captured in the concepts of *path blocking* and *graph separability* introduced in page 47.

To determine the belief vector of a proposition, the evidence in the network is separated at every node into *prospective* and *diagnostic* evidence. We immediately notice a major difference between the Bayes networks approach and the probabilistic approaches considered earlier, namely, the inclusion of prospective support for the computation of the degree of belief in the propositions. Prospective support is obtained from the parent nodes of a proposition, and correspond to top-down inferences as opposed to the bottom-up (diagnostic) inferences, normally the only ones considered in other uncertainty management formalisms.

In order to illustrate the modelling of causal knowledge using Bayesian networks, we will take the following situation[1]:

> One day at the office, Mr. Holmes receives a telephone call from his neighbor Dr. Watson, stating that he hears a burglar alarm sound from the direction

[1]This is a modified version of an example used by Pearl [Pear85a].

of Mr. Holmes' house. Preparing to rush home, Mr. Holmes recalls that Dr. Watson is known to be a tasteless practical joker and, therefore, he decides to first call his other neighbor, Mrs. Gibbons, who, in spite of occasional drinking problems, is far more reliable.

When Mr. Holmes called Mrs. Gibbons, he soon realized that she was in a somewhat tipsy mood. Instead of answering his question directly, she went on and on describing her latest operation and how terribly noisy and crime-ridden the neighborhood had become. As he finally hung up, all Mr. Holmes could make out of the conversation was that there was an 80% chance that Mrs. Gibbons did hear an alarm sound from her window.

Immediately after his conversation with Mrs. Gibbons, as Mr. Holmes is preparing to leave his office, he recalls that his daughter Christie is due to arrive home any minute and, if confronted by an alarm sound, would probably (.7) phone him for instructions. Now he wonders whether he should not wait a few more minutes in case she calls.

As he is pondering this question, Mr. Holmes remembers having read in the instruction manual of his alarm system that the device is sensitive to earthquakes and can be accidentally triggered (.2) by one. He realizes that if an earthquake had ocurred, it would surely (.9) be on the news. So, he turns on his radio and waits around for either an announcement or a call from Christie.

From the given information we can structure the graph of figure 3.1, in which we have identified seven nodes along with the propositions they represent. They are:

1. **E**, "an earthquake has taken place in the surroundings of Holmes' house".
2. **B**, "a burglar entered Holmes' house".
3. **A**, "the burglar alarm went off".
4. **R**, "the radio announces the occurrence of an earthquake".
5. C, "Holmes' daughter calls him up".

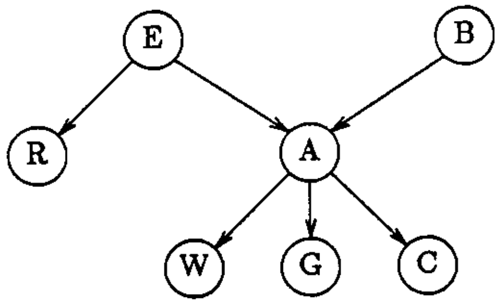


Figure 3.1: A Bayes Network.

6. **W**, "Watson calls".

7. **G**, "conversation with Gibbons".

In this figure the prospective support received by node A (denoted as (D_A^+)), comes from nodes R, E and B, while the diagnostic support of A (denoted D_A^-) (the one given from below the node) is received from nodes W, G and C. In order for this separation to be possible, the data subsets D_A^- and D_A^+ must be disjoint. This requirement is met in singly-connected networks, which disallow the presence of more than one undirected path (ignoring the directionality of the links) between any pair of nodes (e.g., figure 3.1). Multiply-connected networks can be converted into singly-connected ones by conditioning over a subset of variables forming a cycle cutset. In this section we will consider only singly-connected networks; later, we will discuss how multiply-connected networks can be treated.

For any node in a singly-connected causal network, we can separate D into disjoint the subsets of data, each reachable through one outgoing or incoming link. In figure 3.1, from the standpoint of node A, we can separate:

$$D = \left\{ D_{E,A}^+, D_{B,A}^+, D_{A,C}^-, D_{A,G}^-, D_{A,W}^- \right\}, \tag{3.3}$$

where $D^+_{M,N}$ represents the data available from the subnetwork on the tail side of the link $M \rightarrow N$, and $D^-_{M,N}$ is the data available on the head side of the the same link (so that $D = D^-_{M,N} \cup D^+_{M,N}$). Assuming conditional independence among the children given their parent, we can write:

$$P(D|A) = P(D^+_{E,A}, D^+_{B,A}|A)P(D^-_{A,C}|A)P(D^-_{A,G}|A)P(D^-_{A,W}|A), \tag{3.4}$$

Thus, using Bayes equation, we can compute the belief vector as:

$$Bel(A_i) = P(A_i|D) = \alpha P(D^-_{A,W}|A_i)P(D^-_{A,G}|A_i)P(D^-_{A,C}|A_i)P(A_i|D^+_{E,A}, D^+_{B,A}), \tag{3.5}$$

where α is a normalizing constant. The conditional probability of node A, given the data coming from its ancestors, can be rewritten as:

$$P(A_i|D^+_{E,A}, D^+_{B,A}) = \sum_{jk} P(A_i|B_j, E_k)P(B_j|D^+_{B,A})P(E_k|D^+_{E,A}). \tag{3.6}$$

Hence, the degree of belief in any node is computed as a function of the causal evidence received from each parent separately and from the diagnostic support given by each of its successors. Utilizing the standard nomenclature of Bayesian Networks, we write:

$$\pi_A(B_i) = P(B_i|D^+_{B,A}), \tag{3.7}$$

and

$$\lambda_C(A_i) = P(D^-_{A,C}|A_i). \tag{3.8}$$

Combining equations 3.7 with 3.8 and 3.6, we may rewrite 3.5 as:

$$Bel(A_i) = \alpha\lambda_W(A_i)\lambda_G(A_i)\lambda_C(A_i)\left[\sum_{jk} P(A_i|B_j, E_k)\pi_A(B_j)\pi_A(E_k)\right]. \tag{3.9}$$

In summary, to compute the belief vector of a proposition, we need the π parameters of its immediate ancestors, the λ parameters from its successors and the conditional probability matrix that relates the node to its immediate causes. Nodes with no specified causes (i.e., with no incoming links) can be classified as *primary etiologies*. They represent the propositions that would ultimately explain the data available from the evidence nodes and whose causal support is beyond the scope of the knowledge base. For these nodes, the conditional probability matrix takes the form of a vector of prior probabilities.

In theory, the size of the conditional probability matrices stored at each node grows exponentially with the number of its immediate ancestors. However, various techniques can be used to encode these matrices economically, (e.g. see [Kim84] and [Coop84]). We will assume that these higher order probability matrices are available. The leaves in the causal graph represent the evidence nodes, into which all the information that enters the system is placed. The strength of the causal relationship established by the links is quantified by the conditional probability of the effect given its cause, e.g., the link $A \rightarrow C$ is quantified by the probabilities: $P(C|A) = 0.7$, and $P(C|\neg A) = 0$.

In the case of multiple causation (i.e., many causes for the same effect), we specify the conditional probability of the effect given the joint occurrence of the causes. This is exemplified by node A, which has two possible causes, an earthquake or a burglary, and for which we assume the matrix[2]:

$$\begin{aligned} P(A|E,B) &= 1.0 \quad P(A|\neg E,B) = 0.949 \\ P(A|E,\neg B) &= 0.2 \quad P(A|\neg E,\neg B) = 0.01 \end{aligned} \tag{3.10}$$

The information regarding the occurrence of the primary etiologies is summarized by their prior probabilities, which are part of the domain knowledge. In the example we have chosen the same prior probability of 0.003 for both primary etiologies (i.e. $P(E) = 0.003$, and $P(B) = 0.003$). These probabilities can be justified based on statistical data available about crime rates, geological activities, or from other relevant sources. If our interest had been to determine the exact probability of an earthquake on that day, we would have had to provide a detailed causal model of an earthquake, including geological stresses, climatological conditions, gravitational interaction with the moon and other celestial bodies, etc. Our interest, however, is far from that, a crude number, summarizing our belief in the occurrence of an earthquake, would suffice.

To exemplify the computation of the belief vector, we will use equation 3.9. Before doing so, however, a word of caution pertains to the computation of the parameters $\lambda_W(A)$ and $\lambda_G(A)$, which correspond respectively to the support given by the statement made by *Watson* and the conversation with Gibbons to the proposition "the alarm went off." We may feel tempted to assess the values

[2]Our treatment of E is rather simplistic, since the decomposition of it into E and $\neg E$ would not take into consideration the strength of an earthquake. Imagine a 7.8 degree earthquake (Richter scale), for that event we would probably have $P(A|\text{very strong } E) \approx 1.0$. Hence, in a more detailed model we would define several possible states for E, each accounting for a different strength of an earthquake.

$P(W|A)$ and $P(G|A)$, amounting to assigning probabilities to all possible ways in which these two conversations could have developed. This would be impossible to enumerate or even to articulate and, fortunately, would not be required because these nodes represent evidence which is already available. Instead of specifying the matrices $P(G|A)$ and $P(W|A)$, we give the likelihood ratios[3]:

$$\lambda_W(A) : \lambda_W(\neg A) = 9 : 1, \tag{3.11}$$

$$\lambda_G(A) : \lambda_G(\neg A) = 4 : 1. \tag{3.12}$$

To determine the parameter $\lambda_C(A) = P(D^-_{A,C}|A)$, we notice that $D^-_{A,C}$ is empty. In these cases we assign:

$$[\lambda_C(A), \lambda_C(\neg A)] = [1, 1] \tag{3.13}$$

which is interpreted as carrying no diagnostic support from C to any possible instantiation of node A. To compute the parameters $\pi_A(B)$ and $\pi_A(E)$ we notice that both $D^+_{E,A}$ and $D^+_{B,A}$ are empty; hence, we obtain:

$$[\pi_A(B), \pi_A(\neg B)] = [0.003, 0.997], \tag{3.14}$$

$$[\pi_A(E), \pi_A(\neg E)] = [0.003, 0.997]. \tag{3.15}$$

Based on these values and on equation 3.9, we compute:

$$Bel(A) = [0.328, 0.672], \tag{3.16}$$

[3]We are assigning a 90% chance that Dr. Watson's call originates out of a serious concern, and a 10% that he is being a tasteless joker again. Also, we assign an 80% chance that Mrs. Gibbons heard the alarm.

which means that, given the information surrendered by Dr. Watson and Mrs. Gibbons, there is a 32.8% chance that the alarm went off. Similarly, the belief vectors of B and E can be computed as:

$$Bel(B) = [0.07, 0.93] \tag{3.17}$$

$$Bel(E) = [0.0165, 0.9835] \tag{3.18}$$

3.2 Propagation and Update

Whenever new evidence is gathered, we need to propagate the information through the entire network and update the belief vectors of all its propositions. To describe the mechanism of propagation and update, we will use the *message passing paradigm* of object-oriented programming languages. Each node A is given a processor that computes (in parallel) the associated belief vector and passes information to the processors associated with A's neighbors.

When a sensory node N is instantiated as N_j, its belief vector becomes $\underline{Bel}(N) = (0, ..., 0, 1, 0, ..., 0)$ with 1 at the j^{th} position. This assignment constrains the belief vectors of its neighbors to new values that can be obtained by applying equation 3.9. In turn, these new assignments will constrain the values of the belief vectors of all neighbors of already updated nodes. Therefore, the new information sets up multi-directional a propagation process in the network, reaching equilibrium once all constraints are satisfied [Pear85b].

In the computation of the belief vector of a node in equation 3.9, each term can be computed locally by the process associated with the corresponding neighboring node. From the definition of $\pi_C(A) = P(A_i|D^+_{A,C})$, partitioning $D^+_{A,C}$ into its components $D^+_{E,A}, D^+_{B,A}, D^-_{A,W}, D^-_{A,G}$ and applying Bayes rule, we obtain:

$$\pi_C(A_i) = \alpha\lambda_W(A_i)\lambda_G(A_i)\left[\sum_{jk} P(A_i|B_jE_k)\pi_A(B_j)\pi_A(E_k)\right] \tag{3.19}$$

Hence, to compute the prospective π parameters, we need to combine the parameters of the neighbors according to the previous equation. In an analogous way, we obtain:

$$\lambda_A(E_i) = \alpha\sum_{jk}[\pi_A(B_j)\lambda_G(A_k)\lambda_W(A_k)\lambda_C(A_k)P(A_k|B_j,E_i)] \tag{3.20}$$

Then, after updating the belief vector of node A, the corresponding processor[4] computes the λ and π parameters associated with every neighboring node and propagates these newly computed messages to all neighbors. Normally, the relaxation process is activated when a sensory node becomes instantiated. Starting with the sensory node, the messages are updated (according to equations 3.19 and 3.20) and sent to the neighbors, which update their own belief vectors and compute new messages for their neighbors. To illustrate this relaxation process, let us assume that in our example Mr. Holmes turned on his radio and heard that there was an earthquake 50 miles north of his home. Then, in the computation of the belief vector of E, we would have:

$$[\lambda_R(E), \lambda_R(\neg E)] = [1, 0], \tag{3.21}$$

[4]In the future when we refer to a node we will mean either the node, or the processor associated with that object. The precise meaning will be given by the context.

meaning that the radio broadcast gives full support to the belief in the earthquake. Given this new parameter, node E recomputes its belief vector, making $Bel(E) = 1.0$ and also updates the message $\pi_A(E)$ to its son A, which becomes:

$$[\pi_A(E), \pi_A(\neg E)] = [1, 0]. \tag{3.22}$$

In the next relaxation step, node A updates its belief vector, obtaining:

$$Bel(A) = [0.9, 0.1]. \tag{3.23}$$

Node A also updates the messages $\lambda_A(B)$ and $\pi_C(A)$, and this process goes on until all constraints are satisfied.

This example demonstrates a nonmomotonic behavior – if we compute the new value of the belief vector of node B, we obtain:

$$Bel(B) = [0.013, 0.987] \tag{3.24}$$

a reduction of the belief in the burglary. This effect of *explaining away* the burglary would not have happened if there were no diagnostic evidence supporting the alarm sound. Hence, we say that nodes B and E have been "connected" by the evidence received by its common son A.

3.3 Properties of the Bayesian Network Formalism

A very important characteristic of causal graphs representation is that they provide a pictorial view of the dependency relationships among the variables in the model. To identify these dependencies in Bayes networks we make use of the concepts of *path blocking* and *graph separability.*

Definition 3.1 Path blocking: *An arbitrary path from N_a to N_b is said to be blocked by node y whenever there is a subpath $x - y - z$ of the original path (N_a, N_b), such that either:*

- *y is an ancestor of both x and z $(x \leftarrow y \rightarrow z)$, and y is instantiated,*
- *x is directly connected to z through y $(x \rightarrow y \rightarrow z)$ or vice versa $(z \rightarrow y \rightarrow x)$, and y is instantiated,*
- *y is a descendant of both x and z $(x \rightarrow y \leftarrow z)$ and y has not received any diagnostic support (i.e., none of y's descendants is instantiated).*

In our example, instantiation of node A will block nodes W, G and C (according to the first case) and will block the path between node B and node W (second case). On the other hand, if Watson had not called, and the conversation with Gibbons had not taken place, nodes E and B would be blocked (third case).

Definition 3.2 Graph separation: *two arbitrary nodes in the network are separated by a subset S of instantiated variables if all paths*[5] *between them are blocked by S.*

From this definition it can be shown that, if nodes N_1 and N_2 are separated by S then:

$$P(N_1|S, N_2) = P(N_1|S) \tag{3.25}$$

[5]This definition is valid for any network, not only for singly connected ones.

Henceforth, we will use the term separation (or connectivity) to mean separation in the sense of definition 3.1. In Bayesian networks, graph separation is a dynamic property, i.e., two nodes may be separated in one data context (defined by the set D of instantiated nodes) and connected in another.

In Holmes' example we notice that nodes B and E are connected, since the intervening variable A has received evidential support. If there were no evidence impinging on node A from its descendants, these nodes would be separated.

3.4 Propagation in Multiply Connected Networks.

We have described the propagation scheme as a multidirectional relaxation process, using both predictive and diagnostic inferences. Executing the same scheme in multiply connected networks would lead to a *circular reasoning* problem. As an example, let us take the small network of figure 3.2. Upon the arrival of evidence to node y we relax a constrain with node z (a symmetric process follows in the direction of node x), then we proceed to relax the constraint between z and x, updating x's belief vector. However this last update will render x and y un-relaxed once again, and a new cycle of relaxation begins. This process may eventually converge, but to an incorrect value.

The reason that this problem occurs in multiply connected graphs is that our separation of prospective and diagnostic evidence at every node will no longer be valid and the definition of D^- (likewise the definition of D^+) will not hold. In figure 3.2, for example, there would be no distinction between D_y^- and D_y^+.

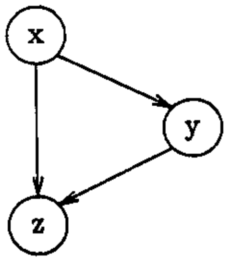


Figure 3.2: A Multiply Connected Bayes Network.

One way to handle multiply connected networks is to break all their undirected cycles by *conditioning* [Pear85b]. This is achieved by finding a small subset S of nodes in the network such that every cycle will be broken by instantiating all nodes in S. This set will be referred to as "cycle-cut-set". For example, in figure 3.2 the instantiation of either node, x or y, would break the cycle. Thus, if we want to propagate some data, we first instantiate a node, say x, and propagate this evidence, once equilibrium is reached we instantiate x to a different value and propagate again. The final degree of belief in a proposition is obtained as the linear combination of the degrees of belief of the proposition resulting from every possible instantiation of node x. An obvious limitation of this approach is that the computational complexity grows exponentially with the cardinality of the set S, thus making it hard to deal with highly connected networks.

In this thesis we will develop a propagation mechanism for singly-connected networks, relying on the process of conditioning to handle the multiply-connected ones.

3.5 Influence Diagrams

A different approach to the computation of conditional probabilities in a Bayesian Network is presented by the influence diagram approach [Howa84]. Influence diagrams have been developed for the analysis of probabilistic models, and their semantic content is equivalent to that of a BN.

The power of these systems relies on the development of causal or inferential models on which we have a considerable degree of independence. An important difference between these two approaches is that, in an influence diagram, the goal is to find the conditional probability $P(\mathbf{K}|\mathbf{I})$, where **K** and **I** are arbitrary subsets of S (the set of nodes in the diagram); whereas in a BN we obtain the conditional probability of individual propositions given the data at hand.

The analysis of influence diagrams usually involves external, global computations to determine the joint probability distribution of selected subsets of variables of the diagram.

In the Bayesian network approach, on the other hand, the computation is performed at the nerwork itself, by passing messages between the nodes. One of the advantages of this approach is its computational simplicity. Also, when new information is received, the mechanism is one of update; previous computations are not discarded. In contrast, the computation of the probability distributions in influence diagrams is done "from scratch" any time new information arrives, not taking into consideration old information.

Finally, a distributed approach permits us to identify the sources of information at every node. This can be used to track down the sources of a given modification. With this information, we can construct meaningful explanations for the changes on the belief parameters of the nodes in the system. By contrast, computing the belief parameters based on global characteristics of the network normally involves computational steps which are void of conceptual correlates.

3.6 Summary.

Bayesian Networks provide a mechanism for uncertainty management that is applicable to many areas, notably in decision support systems and intelligent reasoning systems. The formalism provides a calculus for updating the degrees of belief of the variables in a given a causal model. The calculus is very efficient in singly connected networks. There is also a mechanism of handling multiply connected networks; but its complexity is exponential in the number of nodes necessary to break all the undirected loops in the network.

An important feature of this calculus is the distinction between prospective and diagnostic supports. This distinction is compatible with human reasoning, and it facilitates rich explanations of the system's reasoning, since the flow of evidence can be identified by tracing down the messages.

CHAPTER 4

Relevance-Based Propagation in Bayesian Networks

4.1 The Problem

Once the problem domain has been modeled by means of a Bayesian Network we are in the position of feeding information into the system and have the numerical degrees of belief updated for every node in the network. However, most of the times, we want to focus our efforts on assessing the value of a small subset of nodes; thus, a propagation throughout the whole network seems unnecessary. This claim is in concordance with the way humans focus their attention over a small set of salient hypotheses.

For example, let us consider the task of debugging a system S. We normally choose a hypothesis H that would explain the ill behaviour of S, and try to gather evidence to support or deny H. The new evidence may carry enough information to confirm H, in which case we terminate the debugging process, or, if H had left some aspects of the misbehavior of S unexplained, we iterate this process, postulating more refined hypotheses. If, instead, the evidence denies H, we disregard this hypothesis and choose another plausible H as a new target of attention.

This motivates our interest in focusing the network activity towards updating the belief in *one* target hypothesis.

The importance of the information impinging on a sensory node will be measured according to the impact that information has on the belief of the target node. While in the approach all the information is propagated, even if it is *irrelevant* for the computation of the target hypothesis, here we propose to limit the propagation so that only relevant information is propagated. Doing so will decrease the amount of data traffic in the network and the amount of computation spent on the propagation.

There is yet another important motivation for focusing attention on a target hypothesis. Let us assume that we have defined a Bayesian Network for a given problem domain, and that a subset S of the nodes (normally the leaves) are defined to be sensory nodes (for example, in a medical domain a sensory node may be associated with a particular test, and its values range over all the possible outcomes of the test). In general, the instantiation of any of these sensory nodes involves a positive cost, and the utility of the information they convey might be insufficient to justify this cost. Thus, it is important to decide which node in S should be instantiated first, on the basis of the information it renders to the target node. That is, we can assign priorities to the sensory nodes based on their degree of *informativeness*.

Finally, having a measure of the relevance of the nodes in S enables us to decide when to stop the acquisition of information. Without this measure we might end up spending resources in meaningless and possibly very costly tests. Even worst, we might erroneously decide that we have acquired enough information to assess the target node. Termination should occur only when none of the uninstantiated sensory nodes promises a higher benefit than its cost.

As an example, let us consider the burglary case. Here, Holmes' main interest is focused towards the burglary, and the information he receives is considered relevant only if it helps him reach a decision regarding whether or not there was a burglar in his house. For instance, if he is sure that the alarm did indeed go off, based on his conversation with Dr. Watson, he will regard the conversation with Gibbons as irrelevant (unless, of course, she could give him direct information about a burglar). However, he would still be interested in knowing if there was an earthquake in the surroundings, since this information would provide an alternative explanation for the alarm sound and, hence, would reduce the suspicion in the target hypothesis, namely the burglary.

Our objective is to define parameters of relevance for each node in the network. These parameters will be called *Impact Parameters*, since they are used to evaluate the impact information flowing in the network has on the belief in the target hypothesis. We require that these parameters be updated using the general propagation scheme without altering its original properties[Pear85c], namely:

1. Efficiency in storage and time.

2. Local and asynchronous computations, i.e., the final values should be en-

tirely independent of the control mechanism that activates the individual operations.

3. New information diffuses through the network in a single pass. The time required for completing the updating should be proportional to the diameter of the network.

This implies that the impact parameters, like all other parameters, will have to be computed and *updated* in a distributed fashion, i.e., using only information from neighboring nodes. The proposed mechanism of propagation adds important features to the original scheme proposed by Pearl [Pear85c], and in fact, as we will show in the following sections, keeps the aforementioned properties.

4.2 Definition of the Impact Parameters.

The purpose of the impact parameters is twofold. First they should allow us to measure the relevance, with respect to the target hypothesis, of messages pending propagation. Secondly, they should permit us to evaluate the potential worth of testing a node before it is instantiated. In order to formalize these considerations we need the following definitions:

Definition 4.1 *A* Target Node *is a salient node in the Bayesian Network that serves as a reference for judging the relevancy of the information.*

In the burglary case of last chapter it would be natural to define B (the event of burglary) as our target hypothesis (see figure 3.1).

Definition 4.2 Path neighbor or Path node. *For every node there exists a unique undirected path*[1] *to the target node*[2]*. The path neighbor N^P of node N is defined as the immediate neighbor of N that is along its path to the target node T.*

In figure 4.1, N^P is N's path neighbor with respect to T. In Holmes' case, the earthquake's path node is the alarm A.

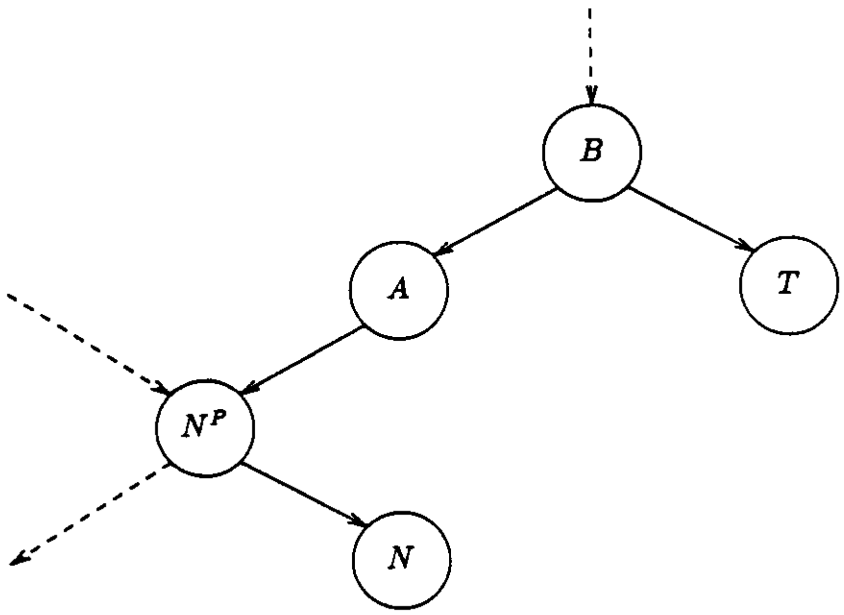


Figure 4.1: N^P is a Path Neighbor of N

Definition 4.3 $D^+_{n,m}$ and $D^-_{n,m}$ data subsets. *Each link $(n \rightarrow m)$ separates the network into two unconnected subgraphs. $D^+_{n,m}$ represents the data that is available on the sub-network containing the node n. Similarly, $D^-_{n,m}$ represents the data available from the other sub-network (i.e. the one that contains m). (See figure 4.2).*

[1]A Bayesian Network is a Directed Acyclic Graph (DAG). When we refer to an undirected network (or an undirected path) we are referring to the network (or path) in the graph obtained by ignoring the directionality of the arcs in the DAG.

[2]This is so, because we are dealing only with singly connected networks.

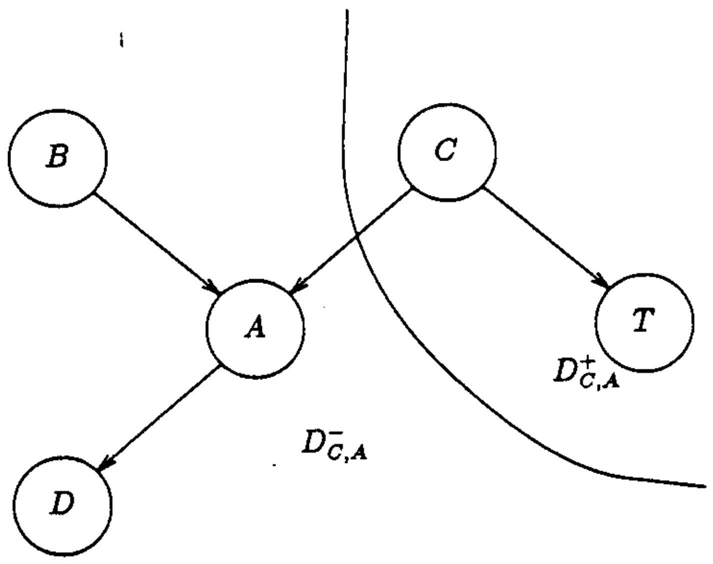


Figure 4.2: Data subsets separated by the link (C, A).

Before proceeding with the definition of the impact parameters, we will express the degree of belief in the target hypothesis in terms of the belief vector of an arbitrary node N.

Let us assume we have an arbitrary Bayesian network with target node T, and a node N, whose path neighbor is N^P. We want to determine how the information impinging on N affects the degree of belief in T. Consider the portion of the Bayesian Network pictured in figure 4.3, where the directionality of the link between N^P and N has been arbitrarily chosen.

In chapter 3 we defined:

$$Bel(T_i) = P(T_i|D), \tag{4.1}$$

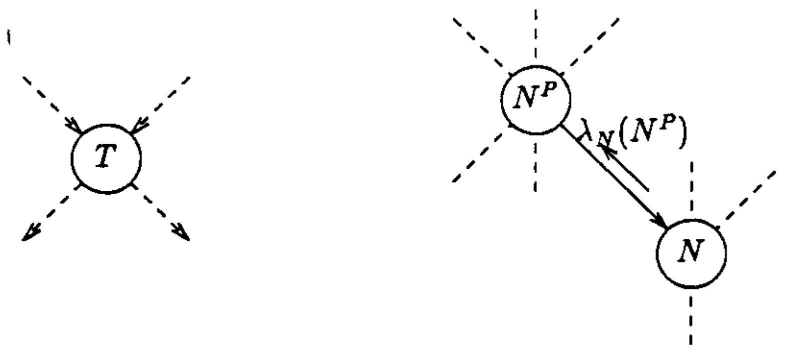


Figure 4.3: Arbitrary Bayesian Network with target node T.

where D is all the data available from sensory nodes, and T_i is one of the possible instantiations of the target node. By conditioning on N we can write:

$$P(T_i|D) = \sum_j P(T_i|D, N_j)P(N_j|D), \tag{4.2}$$

which can also be written as a vector-matrix product:

$$\underline{Bel}(T) = \underline{Bel}(N) \cdot \left[S^{N,T}\right], \tag{4.3}$$

where the components of the matrix $S^{N,T}$ are defined as:

$$S^{N,T}_{ij} = P(T_j|D, N_i). \tag{4.4}$$

This motivates the following definition:

Definition 4.4 Sensitivity matrix. *The sensitivity matrix $s^{N,M}$ of node N with respect to node M is defined as:*

$$s^{N,M}_{ij} = P(M_j|D, N_i) \tag{4.5}$$

Equations 4.2 and 4.3 show that the sensitivity matrix obeys the relation:

$$\underline{Bel}(M) = \underline{Bel}(N) \cdot \left[s^{N,M}\right]. \tag{4.6}$$

Thus, the sensitivity matrix of N with respect to M measures the effect that variations in the belief vector of node N will have on the belief vector of node M.

The question now is how to compute this matrix, and how to update it by local computations each time new data arrives. This is shown in the next section.

4.3 Propagating and Updating the Impact Parameters.

Our approach would be to recursively define the sensitivity matrix of any node with respect to the target node T in terms of two quantities: (1) its sensitivity matrix with respect to its path neighbor, and (2) the sensitivity matrix of its path neighbor with respect to the target node. The former will be denoted by lower case s and the latter by capital S.

Definition 4.5 Path-cut node: *An intermediate node m on the path (n,t) between n and t is said to be a* path-cut *for (n,t) if the instantiation of m renders nodes n and t separated.*

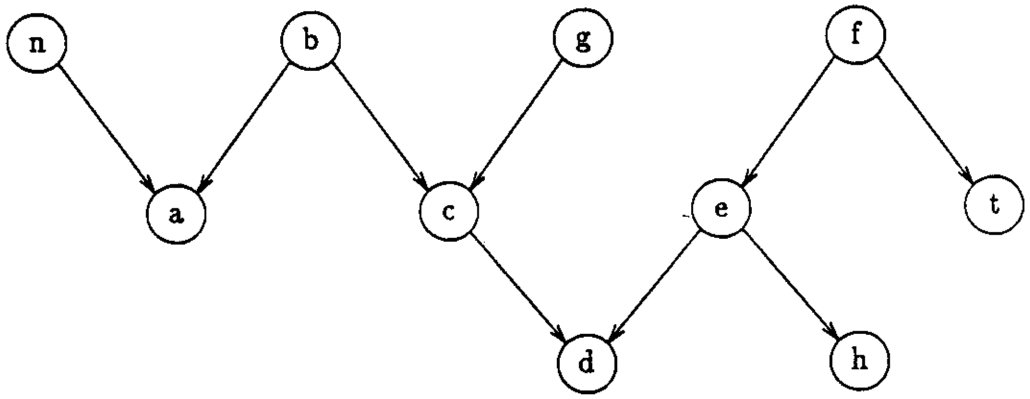


Figure 4.4: a and d are not Path-cut Nodes.

In figure 4.4, for example, nodes b, c, e and f are path-cut nodes. On the other hand, neither a nor d are path-cut nodes.

Based on this definition we introduce the following theorem:

Theorem 4.1 *If M is a path-cut node for path (N,T), then:*

$$\left[S^{N,T}\right] = \left[s^{N,M}\right]\left[S^{M,T}\right]. \tag{4.7}$$

Proof: We know that:

$$S_{ij}^{N,T} = P(T_j|D,N_j), \tag{4.8}$$

by conditioning on M:

$$S_{ij}^{N,T} = \sum_k P(T_j|D,N_i,M_k)P(M_k|D,N_i) \tag{4.9}$$

we recognize that:

$$s_{ik}^{N,M} = P(M_k|D,N_i). \tag{4.10}$$

Also, since M separates node N from node T we have:

$$\begin{aligned} P(T_j|D,N_i,M_k) &= P(T_j|D,M_k) && (4.11)\\ &= S_{kj}^{M,T}. && (4.12)\end{aligned}$$

So,

$$S_{ij}^{N,T} = \sum_k s_{ik}^{N,M} S_{kj}^{M,T} \tag{4.13}$$

which is equivalent to equation 4.7.

□

Now, if we want to compute the sensitivity matrix of a node N in terms of the sensitivity matrix of its path neighbor N^P, and if N^P is a path-cut node for the path (N,T), we can use the following relation:

$$\left[S^{N,T}\right] = \left[s^{N,N^P}\right]\left[S^{N^P,T}\right] \tag{4.14}$$

In cases where N^P is not a path-cut node we will have to resort to a different strategy.

Lemma 4.1 . *For any undirected path (n,t) of length greater than two*[3]*, at most one of any two consecutive nodes along the path is a non-path-cut.*

Proof: Suppose that we have a sub-path $m_1 - m_2 - m_3$, such that the three nodes m_1, m_2, and m_3 belong to the path (n,t). A necessary condition for m_2 to be a non-path-cut node is that both m_1 and m_3 be its ancestors. Now, for m_1 (or m_3) to be also non-path-cut for (n,t) we need m_2 to be an ancestor of it, which is impossible. It then follows that two consecutive nodes in a path cannot simultaneously be non-path-cut nodes.

□

Therefore, the only case in which equation 4.7 is not applicable is illustrated by figure 4.5, where R is N^P's path neighbor and the path that joins R and T is arbitrary. In this case N^P is not a path-cut node for (N,T), but lemma 4.1

[3]Where the length of the path is equal to the number of links it traverses.

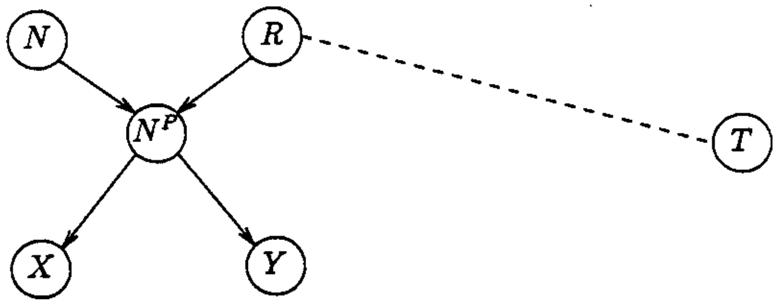


Figure 4.5: A father-son-father Path to the Target Node.

guarantees that R is path cut. Thus using theorem 4.1 we can write:

$$S^{N,T} = \left[s^{N,R}\right]\left[S^{R,T}\right] \tag{4.15}$$

Equations 4.15 and 4.14 suggest that we may determine the sensitivity matrix of any node N with respect to the target node T recursively:

$$S^{N,T} = \left[s^{N,M}\right]\left[S^{M,T}\right]. \tag{4.16}$$

Where M will be chosen as the closest (N,T) ***path-cut*** node to N. In the extreme case, we know that:

$$\left[S^{T,T}\right] = [I]. \tag{4.17}$$

where I is the identity matrix. In figure 4.4, for example, the sensitivity matrix $S^{c,t}$ would be computed as:

$$\begin{aligned} \left[S^{c,t}\right] &= \left[s^{c,e}\ \right]\left[S^{e,t}\right] && (4.18)\\ &= \left[s^{c,e}\ \right]\left[s^{e,f}\right]\left[S^{f,t}\right] && (4.19)\\ &= \left[s^{c,e}\ \right]\left[s^{e,f}\right]\left[S^{f,t}\right]\left[S^{t,t}\right] && (4.20)\end{aligned}$$

We will use the terminology introduced by the following definitions:

Definition 4.6 T-path-cut neighbor *of a node. Node m is said to be T-path-cut of node n, if m is the closest node to n such that m is path-cut node for the path* (n, T).

In figure 4.6 N is T-path-cut of node X and node M is T-path-cut of node A.

Definition 4.7 Neighborhood matrix. *The sensitivity matrix $s^{N,M}$ of a node N with respect to its T-path-cut node M is called the* neighborhood matrix *of node N.*

This motivates the need for calculating the neighborhood matrix $s^{N,M}$ of any node N, where M is N's T-path-cut node.

We distinguish the following three cases:

1. (son-father relation). M is the father of N (see figure 4.6). From equation

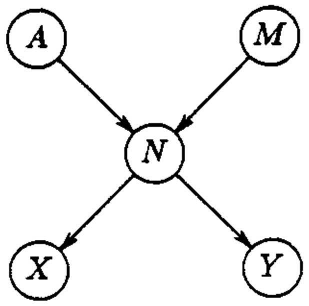


Figure 4.6: M is a father of N

4.5 we know that:

$$s_{ij}^{N,M} = P(M_j | D, N_i) \tag{4.21}$$

and the sensitivity matrix of N with respect to M is given by (see derivation A.1 appendix A) the expression:

$$s_{ij}^{N,M} = \alpha_i \pi_N(M_j) \sum_k P(N_i|M_j, A_k)\pi_N(A_k). \tag{4.22}$$

2. (father-son relation). M is a son of N (see figure 4.7). In this case we have

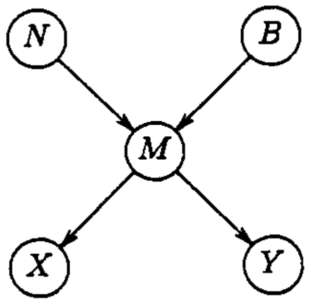


Figure 4.7: M is a son of N

(derivation A.2, appendix A):

$$s_{ij}^{N,M} = \alpha_i \lambda_X(M_j)\lambda_Y(M_j) \sum_k P(M_j|N_i, B_k)\pi_M(B_k) \tag{4.23}$$

3. (spouse-spouse relation). M is a father of one of N's sons (see figure 4.8). It can be shown that (see derivation A.3 appendix A):

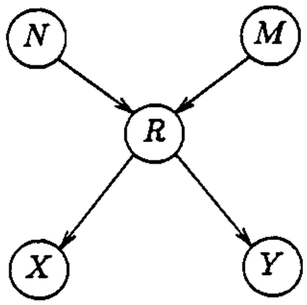


Figure 4.8: M is a son of N

$$s_{ij}^{N,M} = \alpha_i \pi_R(M_j) \sum_k \lambda_X(R_k)\lambda_Y(R_k)P(R_k|N_i, M_j). \tag{4.24}$$

The results obtained in equations 4.22, 4.23, and 4.24, along with theorem 4.1 provide a local mechanism for updating the sensitivity matrices of every node with respect to a predefined target. We see that the belief-support messages (λ and π) are sufficient also for updating the local matrices $s^{N,M}$ and, by virtue of equation 4.7, this should be sufficient for updating the sensitivity matrices as well.

In the standard approach, every time a node's information is updated it sends out new messages (π's and λ's) to its immediate neighbors, thus triggering message propagation through the entire network. In the new approach we also maintain sensitivity matrices in all nodes, and propagate additional messages to keep these sensitivity matrices updated. To describe this propagation we will make use of the nomenclature introduced by the following definitions:

Definition 4.8 Targetted message, *is a message whose recipient is the path neighbor of the sender. Pictorially, a targetted message goes in a direction towards the target node. (Figure 4.9).*

Definition 4.9 Non-targetted message, *is a message whose recipient is not the path neighbor of the sender or, in other words, it is a message going in a direction away from the target node. (Figure 4.9).*

Notice that every message can be labeled as either targetted or non-targetted, since for every node there is a unique neighbor closer to the target (the path node).

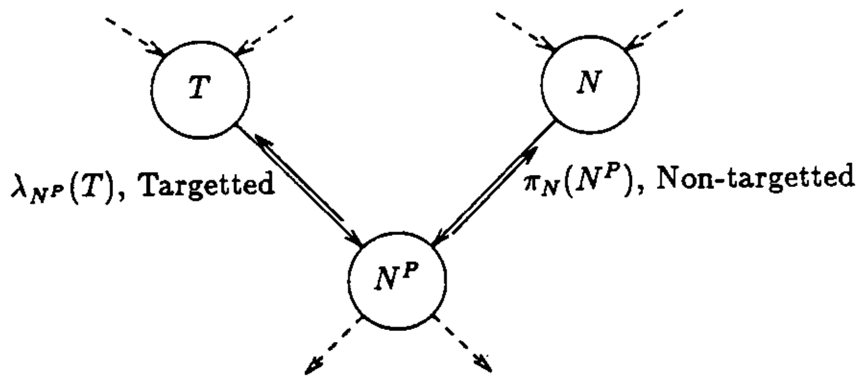


Figure 4.9: Targetted and Non-targetted messages.

From theorem 4.1, we know that the sensitivity matrix of node N may be computed as the product of its neighborhood matrix and the sensitivity matrix of its T-path-cut node. Later on we will show that, for the purpose of updating the sensitivity matrices, it is enough to devise a mechanism to update the neighborhood matrices. A description of this update follows.

The conditions under which the neighborhood matrix $s^{N,M}$ requires different updating are illustrated by the following two cases:

1. N is a son of its path neighbor (figure 4.10), and the path between M and T is arbitrary.

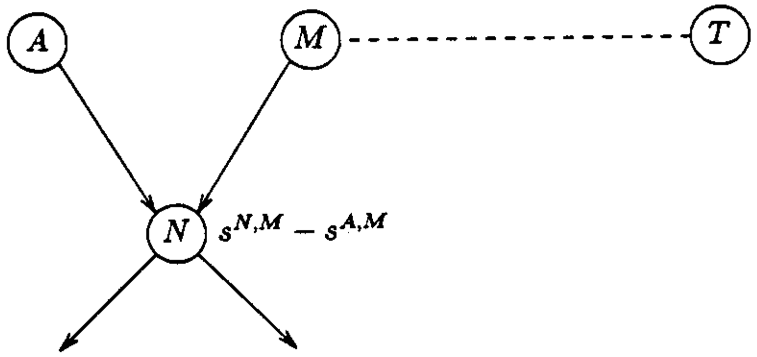


Figure 4.10: N is a son of its path neighbor.

2. N is a father of its path neighbor (figure 4.11), and the path between N and T goes through one of M's sons.

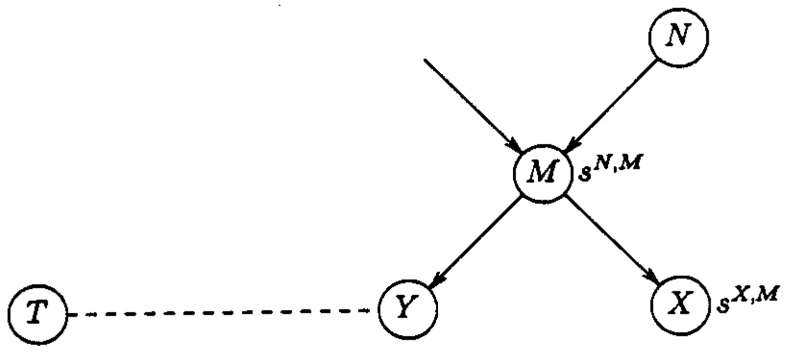


Figure 4.11: N is a father of its path neighbor.

Each node will store its own sentivity matrix, but the update of this matrix will not necessarily be done locally. The labels on the nodes of figures 4.10 and 4.11 indicate which nodes update the neighborhood matrices.

In the first case whenever N receives a λ message, it will have to update the matrix $s^{A,M}$, according to equation 4.24. Additionally, when N receives a π message from any node other than A it also updates $s^{A,M}$ (equation 4.24). Also, from equation 4.22, N updates $s^{N,M}$ whenever it receives a π message, independently of the node that issues the message.

In the second case, M has to maintain the sensitivity matrices of its parents, which are obtained from equation 4.23. The update of $s^{N,M}$ is ensued whenever M receives a message from a node other than N itself.

In the propagation approach proposed here, the scheme for updating the belief vectors of each node has remained unaltered. To update the sensitivity matrices, however, we need to add a new processing step. In the following discussion we will consider the Bayesian Network approach as implemented in a network of independent objects (one object for each node), in line with the object-oriented programming paradigm. Each object is associated with a separate processor in a distributed processing environment. Each node will have a two-way communication channel with its neighbors. The relaxation process is executed by message-exchange between neighboring nodes. When new data arrives at the network, the originated message is propagated throughout the network in a direction away from the source. The updating equations are such that nodes at the periphery of the network do not reflect back messages which they absorb. Therefore, the process must finish in time proportional to the network's diameter.

It would seem natural to let each node update its own sensitivity matrix with respect to the target node. However, this may not be the best way to handle the computation. The cases illustrated by figures 4.7, and 4.8 suggest that the corresponding $s^{N,M}$ sensitivity matrices would be better computed by N's path-neighbor (either M, in figure 4.7 or R in figure 4.8), rather than by N itself. This is so because, in both cases, $s^{N,M}$ is a function of the conditional probability matrix of N's path-neighbor. Also, if we compute $s^{N,M}$ in node N in the aforementioned cases, then $s^{N,M}$ would have to be updated whenever N's path-neighbor receive new messages. In this way, we avoid an unnecessary increase in the communication overhead introduced by this scheme.

We found it more convenient to let each node maintain the sensitivity matrices $s^{N,M}$ of all its fathers, with the exception of the path-neighbor. Also, the sensitivity matrix of a node will be maintained by itself only if its path-neighbor is one of its ancestors, otherwise this matrix is maintained by its path-neighbor.

In the previous paragraphs we have determined the general scheme for the updating of the neighborhood matrix of a node. However, we are interested in the computation of:

$$\left[S^{N,T}\right] = \left[s^{N,M}\right] \cdot \left[S^{M,T}\right] \tag{4.25}$$

which also requires the updating of the sensitivity matrix $S^{M,T}$. In the case illustrated by figure 4.10, we need to update $S^{M,T}$ whenever N receives a message. In the second case,the update follows whenever M receives a message. Fortunately, the following lemma shows that these updates are not necessary:

Lemma 4.2 *Given a path (N,T), let M be any path-cut node for this path, then $S^{M,T}$ is invariant with respect to targetted messages originated at node N.*

Proof: By definition:

$$S^{M,T}_{ij} = P(T_j|M_i, D) \tag{4.26}$$

if M is a path-cut node for the path (N,T), then the instantiation of M will render N and T separated, blocking any messages originated at N.

$\square$

In the figure 4.10, when N receives new messages from any source it would update the sensitivity matrices of A and N with respect to node M. Later on, N would issue new messages, including a targetted message to node M. Our previous lemma guarantees that this targetted message will not affect $S^{M,T}$, hence the updates of $S^{A,T}$ and $S^{N,T}$ are correctly achieved by updating $s^{A,M}$ and $s^{N,M}$ only. This argumentation also applies to the second case (figure 4.11).

In this section we have established a scheme for propagating and updating the sensitivity matrices. This scheme is such that whenever new information arrives at the sensory nodes, it propagates throughout the whole network, simultaneously updating the sensitivity matrices at the nodes traversed.

4.4 Measure of Information.

4.4.1 Value of incoming information.

To determine the relevancy of incoming data to the target hypothesis we need to measure the expected decrease of current uncertainty of the target proposition, due to anticipated or actually received incoming data. First, let us consider the case of evaluating the merit of input data (instantiated sensory nodes). We assume that the change of uncertainty of the Target can be quantified by some function:

$$M(T|D, D') = F(\underline{P}(T|D), \underline{P}(T|D, D')) \tag{4.27}$$

where D' is the newly incoming data, and D is the data available before receiving D'. The function F should return a value which measures the amount of variation that $\underline{Bel}(T)$ (i.e. $\underline{P}(T|D)$) has experienced. The simplest function F is the straight sum of the variations caused by the new incoming data on $\underline{Bel}(T)$'s components, i.e.:

$$M_0(T|D,D') = \sum_i |P(T_i|D) - P(T_i|D,D')| \tag{4.28}$$

The advantage of using M_0 as a figure of merit is its computational simplicity and ease of interpretation by a naive user. However, there are more standard measures of information that can also be utilized, e.g., a measure of variation based on the concept of entropy. The entropy is understood here as a measure of the effort necessary for removing the uncertainty associated with a sample space E. If E is partitioned into a finite number of exhaustive and mutually exclusive events E_k, with known probabilities $P(E_k)$ then,

$$H(E) = -\sum_k P(E_k) \log P(E_k). \tag{4.29}$$

Applying this measure to the target hypothesis T yields:

$$H(T) = -\sum_{k=1}^{t} P(T_k|D) \log P(T_k|D). \tag{4.30}$$

Considering the data D as background information, we define the measure of the information rendered by the new data as the variation experienced by $H(T)$, i.e.:

$$\begin{aligned} M_1(T|D,D') &= \Delta H(T) && (4.31) \\ &= \sum_{k=1}^{t} |P(T_k|D) \log P(T_k|D) - P(T_k|D,D') \log P(T_k|D,D')| \, . && (4.32) \end{aligned}$$

In order to decide whether or not to propagate incoming data, we can judge its merit by either M_0 or M_1. So, comparing M with a positive threshold m_L, defined by the user of the system, we make the decision of propagating the information only if $M \geq m_L$. However, the decision of not sending some information, which is normally taken locally at sensory nodes, will have to be revised every time the impact parameters are modified. This necessity can be illustrated with the example of figure 4.12.

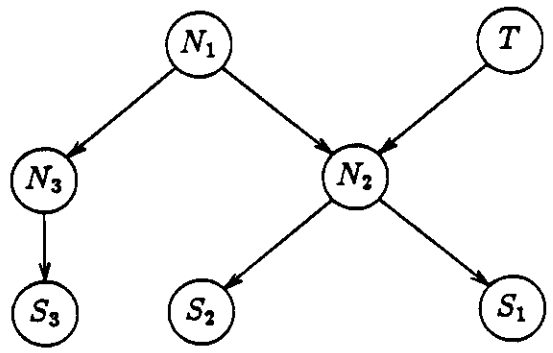


Figure 4.12: Revising the informativeness of a node.

Let S_e be the set of instantiated sensory nodes, initially empty. Under these circumstances, N_1 and T are separated. Thus, if we instantiate S_3, getting $S_e = \{s_3\}$, the message $\lambda_{s_3}(N_3)$ would carry no information relevant to T and, therfore, will remain stored at S_3. However, if N_2 receives diagnostic support from one of its sons, e.g. $S_e = \{s_2, s_3\}$, N_1 and T get connected. We then need to reevaluate the relevance of the message $\lambda_{s_3}(N_3)$, and, in case it exceeds some threshold, we should permit its transmission to N_1

4.4.2 Potential Value of Uninstantiated Nodes.

While M_0 and M_1 allow us to determine the merit of incoming data to N, we also need a measure of the potential informativeness of uninstantiated nodes. If we instantiated a node N then its informativeness would be measured by some function:

$$M'(T|D,N) = F(P(T|D), P(T|N,D)). \tag{4.33}$$

Not knowing the actual outcome of N, we should take the weighted average of M over all possible outcomes of N, that is:

$$M' = \sum_j P(N_j|D)F(P(T|D), P(T|N_j,D)). \tag{4.34}$$

For example, using the information-theoretical approach, we can calculate the conditional entropy of T using the weighted average:

$$H(T|N) = -\sum_j \left[P(N_j|D)\sum_k P(T_k|D,N_j)\log P(T_k|D,N_j)\right], \tag{4.35}$$

and take the difference:

$$R(N) = H(T|N) - H(T). \tag{4.36}$$

While the term $H(T)$ is a constant with respect to node N, the convinience of $R(N)$ is that it takes the value zero for irrelevant nodes.

An important advantage of the entropic measure is that it is based on a well founded body of theory, and hence it inherits qualities that an ad-hoc measure lacks[Dalk85].

4.4.3 Evaluation of λ and π messages.

An important characteristic of Bayesian Networks is their capability to receive and propagate data in a distributed and asynchronous way. When a sensory node is instantiated, the λ messages originated are propagated only if their value, as estimated by equation 4.32, surpasses a threshold m_L. Assume that we have instantiated several nodes, and that their corresponding messages are propagated concurrently in the network. Furthermore, assume that a node N receives new messages from its neighbors and that these messages have been originated from a set D_n of data nodes. After updating the belief vector of node N we compute new λ and π messages to send to N's neighbors. However, before sending these messages we need to reevaluate their combined effect on the target node. Analogously to equation 4.32 we may write:

$$M_n(T|D,D_n) = \sum_{k=1}^{t} \left[P(T_k|D)\log P(T_k|D) - P(T_k|D,D_n)\log P(T_k|D,D_n)\right], \tag{4.37}$$

where, by virtue of equation 4.3,

$$\underline{P}(T|D,D_n) = \underline{P}(N|D,D_n) \cdot \left[S^{N,T}\right]. \tag{4.38}$$

Notice that $\underline{P}(N|D,D_n)$ is the updated belief vector of node N, which is locally available at node N.

In conclusion, whenever a node is updated with more than one new message, the effect of this update is measured, and, if it surpasses a threshold (m_L), it updates and propagates new λ and π parameters. Updates due to single messages are evaluated only once. For example, if we instantiate a single node and find it relevant, all the messages originated are propagated without further questioning. In contrast, the instantiation of several nodes, each one relevant by itself, may result in a combination on which their individual effects are cancelled. The latter case is evaluated at those nodes that receive more than one message.

4.5 Propagation Scheme.

In previous sections we have derived a mechanism by which we can determine the worth of new data coming to a sensory node, and also to measure the potential information that an uninstantiated node can convey to reduce the uncertainty of the target node. In section 4.1 we expressed our interest in focusing the activity in the network on a target hypothesis and its *area of influence.* To formalize this idea we will introduce the following definition:

Definition 4.10 *The* influence area (IA) *of the target node T is defined as the set of nodes that (individually) can potentially reduce the uncertainty in the target node by at least a certain predetermined threshold r, where r is a user defined constant.*

Definition 4.11 *The* maximum potential effect $\overline{R}(N)$ *of node N on the belief of the target node is defined as:*

$$\overline{R}(N) = \max_j \left| \sum_k P(T_k|D, N_j) \log P(T_k|D, N_j) - \sum_k P(T_k|D) \log P(T_k|D) \right| \tag{4.39}$$

To determine the Influence Area of a node we will use the maximum potential effect of a node as given by the definition 4.11. The rationale behind the use of this measure will be discussed after the introduction of the algorithm to obtain the IA of the target node.

For a Bayesian Network with a subset D of instantiated sensory nodes, algorithm 1 (see figure 4.13, page 77) will return T's area of influence. With this algorithm we traverse all possible paths, starting at the target node in a direction away from it. When a node n is visited, its maximum potential effect ($\overline{R}(n)$) over the target node is determined. If $\overline{R}(n)$ surpasses the threshold p, the node and all its ancestors are included in the IA. On the other hand, if $\overline{R}(n)$ does not surpass the threshold p, the node is left out of the IA, and the traversal of the paths that go through n is stopped, leaving the nodes located farther away from the target (and having n in their path to the target) out of the Influence Area. This area of influence varies with the available data, as described in the example of figure 4.12.

Algorithm 1 will return:

- IA = The set of nodes in the *influence area* of the target node T.
- $BOUND$ = set of nodes that separate the influence area from the rest of the nodes, (short for BOUNDARY).

This algorithm makes the implicit assumption that the influence of a node N on T's belief vector decreases with the distance between N and T, i.e., we are assuming that $\overline{R}(N) \leq \overline{R}(N^P)$, where N^P is N's path neighbor. This assumption may not be valid in cases where N^P is not a T-path-cut neighbor of N. In those

Algorithm 1:

```
1. $IA \leftarrow \{T\}$;

2. $OPEN \leftarrow \{$ $T$'s neighbors + parents of $T$'s sons $\}$;

3. $BOUND \leftarrow \emptyset$;

4. While $OPEN \neq \emptyset$

     • pick a node $n$ from $OPEN$ randomly,

     • If $\overline{R}(n) \geq p$ then

         $IA \leftarrow IA + \{n\}$

         $OPEN \leftarrow OPEN +$ $n$'s neighbors + parents of $n$'s sons $-IA$

       Else

         $BOUND \leftarrow BOUND + \{n\}$;
```

Figure 4.13: Algorithm to determine the area of influence of T.

cases we explicitly consider the inclusion of N in IA, even if $\overline{R}(N^P)$ happens to fall under the threshold r.

When new evidence arrives to a sensory node in a Bayesian network it is propagated according to the following criteria:

1. The impact of the data on the belief vector of T is measured, if it exceeds the threshold m_L then the node sends updated messages to its parent nodes, triggering a propagation through the influence area of T. If the sensory node were originally out of the influence area of T, or if its impact were not high enough then this data would be considered irrelevant and propagation would not ensue. In the latter case, the data remains stored at the sensory node, and is propagated later if the IA gets extended up to this node.

2. Once the new data has been approved (i.e. is found to be relevant) all targetted messages originated by its propagation are updated and sent over to its correspondents.

3. Non targetted messages are propagated only in the influence area, however this influence area may change as a result of this new data. To update the IA we can evaluate $\overline{R}(n)$ for every node on the periphery of IA. As a result, the influence area may shrink or expand.

As an example, let us consider the Bayesian network of figure 4.12. Assume that neither S_1 nor S_2 have been instantiated, and that new data arrives at S_3.

Since nodes T and N_1 are separated, we have

$$P(T_k|D) = P(T_k|D, N_{1j}), \tag{4.40}$$

so that $\overline{R}(N_1) = 0$, leaving nodes N_1, N_3 and S_3 out of the influence area of T. Now, assume that one of N_2's sons is instantiated, thus connecting T and N_1. By virtue of step 3 above, $\overline{R}(N_1)$ is reevaluated and, depending on the resulting value of $\overline{R}(N_1)$, IA is extended. The extension of the influence area could go all the way down to S_3, depending on the parameters of links (N_1, N_3) and (N_3, S_3).

In order to include a node N in the IA of the target node, we used the maximum potential effect that N could have on T (according to equation 4.36). Normally, this measure will overestimates this effect; however, if we used a lesser value, e.g., the average $R(N)$, we would be discounting rare but possibly crucial events. For example, assuming that we have nodes N_1 and T connected, we may have a situation where $N_1 =$ false has no noticeable effect on T, but $N_1 =$ true has a significant effect on T. Furthermore, assume that the probability of $N_1 =$ false is very high (e.g. greater than 0.95). Now, if we use the average potential effect of node N_1 (equation 4.39), it will probably render N_1 (and its descendants N_3 and S_3) out of the influence area of the target node. Furthermore, assume that data that confirms N_1 arrives to S_3. Since S_3 is out of the Influence Area, the new data is not propagated, and its effect (which would be very important) would not reach the target node. Using of the maximum-potential-effect measure remedies this situation, at the expense of widening the influence area.

The resulting influence area is not minimal, since $\overline{R}$ overestimate the importance of individual nodes. However, if we wanted to use the more precise measure $R(N)$, we would have to determine the updated values of the belief vector of node N (i.e., $\underline{Bel}(N) = \underline{P}(N|D)$ see equation 4.35). To do so, the new incoming messages will have to be propagated all the way from the sensory nodes to the nodes in the boundary of IA (from S_3 to N_1 in the example); propagation that we wanted to eliminate in the first place. By contrast, the resulting strategy is based on the heuristic of including in the IA every node that can *potentially* affect the target node over a given threshold. This is a conservative heuristic, but it guarantees that all relevant information will eventually reach the target node, and that grossly irrelevant data will not be propagated.

In chapter 3 we discussed the process of conditioning to handle multiply connected networks. If C is a minimal "cycle-cut-set" for the Bayesian network, then the computational complexity of the propagation is exponential on the cardinality of C. By restricting the propagation to the influence area, we not only reduce the overall network size, but we also reduce the cycle-cut-set. Thus, we reduce the computational complexity to an exponential in the cardinality of C'; where C' is the cycle-cut-set of the influence area (since the influence area is a region of the network we have $|C| \geq |C'|$).

There are two problems that we have left out of this thesis. First we have considered T as a single target hypothesis, while it may be important to consider T as a small set of hypotheses, and to govern the propagation of data in the network by considering all hypotheses in T. If this set is small compared to the total size of the network we may simply maintain impact matrices relative to all combinations of the variables in T. This may work well for two or three hypotheses, but may become very expensive as the size of T grows.

Secondly, we have not considered the evaluation of the combined effect of a set of multiple sensory nodes on a target hypothesis. It may well be the case that two sensory nodes would have a combined effect greater than the sum of the two acting independently. One way to deal with this problem is to simulate the combined effect on the target hypothesis of groups of pairs and triplets of the sensory nodes under consideration, and determine the belief on the target node for each group. This approach straightforward, but it becomes too expensive for large sets of sensory nodes.

4.6 Summary

In this chapter we have defined the set of parameters that have to be maintained at each node in order to locally compute the $Bel(T)$, where T is a target node. We have also established a measure of informativeness to judge the relevance of incoming data. This measure allows us to control the propagation of information througout the network, and to categorize the relevancy of potential sources of information. In the next chapter we will discuss some implementation issues.

CHAPTER 5

Implementation Issues

In this chapter we discuss some of the implementation issues of the Bayesian network approach. In appendix B we discuss lower level technical details of one particular implementation.

One of the most important characteristics of the Bayesian network approach is that the network is turned into a computational architecture of independent processors, which compute and update the belief parameters of the underlying propositions in a distributed manner. In this architecture we map each node in the Bayesian network with a processor and each link with a bidirectional communication path between the associated processors.

In order to implement this architecture in a single processor machine, we have used the object-oriented programming paradigm. Each object has an associated local memory and a set of functions that characterize that object. In our scheme, each node is represented as an instance of an object belonging to a common object-type or class. This class is a programming abstraction of the processor.

In principle, each one of the objects can communicate with any other object by passing messages. However, we constrain the message passing to those paths where the receiver is either a son or a parent of the sender, disallowing direct communication between unconnected nodes. Then, the λ and π messages will be associated with actual messages sent by the processor objects.

We assume that there is no central processor, nor is there a common memory. Hence, each processor must have local memory, where it will keep the following information:

1. identification of the node represented;
2. a list of all the neighboring processor objects (nodes), each element of the list with a pointer to the object representing that processor;
3. a copy of the last message received from each neighboring processor;
4. a pointer to its path neighbor;
5. the sensitivity matrix of the node with respect to the target node;
6. the belief vector of the node;
7. status of the node, kept only for sensory nodes. This status may be either positive instantiation, negative instantiation, or no instantiation (no data arrived); and
8. conditional probability parameters of the node with respect to the joint ocurrence of its direct ancestors.

With this information we can locally compute the belief vector of a node given the set of messages received from its neighbors.

Once the network has been established, we need to initialize the network parameters. To do so, we start with the root nodes (i.e., nodes with no ancestors). We initialize the belief vector of the root nodes to their prior probabilities (given by the user). Afterwards, the root nodes trigger a propagation phase by sending π messages compatible with their prior probabilities. These messages propagate until equilibrium is reached. At this point (i.e., before receiving any data), each node will hold its prior belief.

After obtaining the prior belief of each node, the *implicit evidence* nodes are instantiated, and the corresponding messages are issued, initiating a second propagation phase. After this initialization is performed, the network reaches its stable initial state.

The initialization phase is done before a target node is defined. The computation and update of the Sensitivity matrices are disabled until a target node is defined. Even though we did not analyze the case where two or more target nodes are considered, a careful choice of the target node may make the definition of more than one target unnecessary. For example, we may want to assess two hypothesis nodes. Normally, these two nodes will be related, and we could choose the target to be an intermediate node that, if instantiated, would render information about the original two. Another possibility is to use a dummy node, and connect it with the nodes we are interested in assessing. A more precise study of these schemes is left for further research.

When the target node has been selected, the identity of the path neighbor can be determined at each node. This is done simply by triggering an identification process, where each node n sends its identity to those neighbors that will have n as the path-neighbor. This process starts at the target node and spreads from there to the rest of the network, always in a direction away from the target node.

In the distributed updating scheme, each node is activated as soon as it receives a new message from a neighboring node. In this activation the associated processor computes its belief vector incorporating the new information. At this time, the processor updates the sensitivity matrices as described in chapter 3. This scheme, however, is not applicable in a single-processor system, where a different activation approach must be used.

Let us assume that we have a Bayesian network with a subset of sensory nodes instantiated (either positively or negatively). Furthermore, assume that the messages originated due to these instantiations have yet to be issued. In a distributed environment these messages would be concurrently computed and propagated. In a single processor system we would need to serialize this propagation according to some control mechanism. We have implemented four different update mechanisms, which are described in appendix B. One, *recency-driven propagation*, seems more appealing because it is intuitively more similar to the distributed scheme. In this scheme we iterate on an update loop. In each cycle of the update loop we take all nodes that have outstanding messages (not taken into account in previous updates) in an arbitrary order, update their belief parameters and send new messages to its neighbors. In any case, the control mechanism utilized is independent of the final result [Pear85c]. The final scheme used will depend on the goal sought (e.g. conceptual transparency, efficiency).

On page 66 the mechanism by which we update the sensitivity matrices was described. In this approach a node updates the sensitivity matrices according to a criteria of efficiency. We did not always compute the sensitivity matrix of the node itself, instead we computed the sensitivity matrices of some of the node's neighbors. In the implementation of the system, each node checks whose sensitivity matrix must be computed by checking the relationship of the node with its own path neighbor.

The guidelines given in this chapter have been used to construct the BAYNET program. In appendix B we describe that program and give some insights on the lower-level details of implementation.

CHAPTER 6

Summary and Conclusions

Bayesian networks provide a theoretical framework for representing causal and inferential knowledge. Once a model for a particular domain has been established, we need to provide strategies for data acquisition and propagation. In the original system all information is treated equal, regardless of the relevance it may have on the particular problem on which we are working. Thus, the necessity of having a scheme for goal-based activity in the network arises.

In this thesis we have established a scheme that enables us to define a *target* node, so that any piece of information or data flowing in the network can be evaluated in terms of its relevance with respect to the this *target* node.

First defined are *impact parameters*, composed of a *neighborhood matrix* and a *sensitivity matrix*, which are locally stored and updated at each node. These matrices permit one to locally establish the belief vector of the target node as a function of the belief vector and the sensitivity matrix of a node N.

Secondly, a mechanism for distributed computation and update of the *impact parameters* is defined. The scheme proposed offers the advantage of not introducing any communication overhead since it uses the same parameters defined for the update of the belief vector at every node.

We also established measures for determining:

1. the potential value of uninstantiated nodes, especially useful for determining which sensory nodes should be instantiated first;
2. the value of incoming information, used to judge the relevancy of new data, which is propagated only if it renders useful information to the target node.
3. the relevancy of single π and λ messages. Any arbitrary message can be evaluated and either discarded or delivered, based on the impact it will have on the target node;
4. maximum potential effect of a node, used as a criteria to decide whether to include any particular node in the *influence area* of the target node;

Finally, we describe the implementation of **BayNet**, a software tool on which we implemented the propagation scheme proposed here.

We have considered here only the evaluation of the effect that a *single* node will have on the target hypothesis. However, it is also important to be able to determine a measure of the combined effect of the instantiation of several nodes. This work should be a basis for developing such a scheme.

APPENDIX A

Derivation of Equations

Derivation A.1 *Proof of equation 4.22:*

From the definition of the sensitivity matrix we know that:

$$s_{ij}^{N,M} = P(M_j|D,N_i) \tag{A.1}$$

Separating D as D_N^+ and D_N^-, and applying Bayes equation:

$$\begin{aligned} P(M_j|D,N_i) &= P(M_j|D_N^+,D_N^-,N_i) && \text{(A.2)} \\ &= \frac{P(D_N^-|M_j,N_i,D_N^+)P(M_j|N_i,D_N^+)}{P(D_N^-|N_i,D_N^+)}, && \text{(A.3)} \end{aligned}$$

the data underneath N is independent of whatever is above N given the instantiation of node N, so:

$$\begin{aligned} P(M_j|D,N_i) &= \frac{P(D_N^-|N_i)P(M_j|N_i,D_N^+)}{P(D_N^-|N_i)} && \text{(A.4)} \\ &= P(M_j|N_i,D_N^+) && \text{(A.5)} \end{aligned}$$

which is intuitively evident, since D_N^- has been isolated from T, also decomposing D_N^+ into its components:

$$P(M_j|D_N^+,N_i) = P(M_j|D_{A,N}^+,D_{M,N}^+,N_i), \tag{A.6}$$

applying Bayes:

$$P(M_j|D_N^+, N_i) = \frac{P(D_{A,N}^+, D_{M,N}^+, N_i|M_j)P(M_j)}{P(D_{A,N}^+, D_{M,N}^+, N_i)}, \tag{A.7}$$

$D_{M,N}^+$ becomes independent of N_i and $D_{A,N}^+$ given the value of node M, so:

$$\begin{aligned} P(M_j|D_N^+, N_i) &= \frac{P(D_{A,N}^+, N_i|Mj)P(D_{M,N}^+|M_j)P(M_j)}{P(D_{A,N}^+, D_{M,N}^+, N_i)} && \text{(A.8)} \\ &= \frac{P(D_{A,N}^+, N_i|Mj)P(M_j|D_{M,N}^+)P(D_{M,N}^+)}{P(D_{A,N}^+, D_{M,N}^+, N_i)} && \text{(A.9)} \\ &= \alpha_i P(M_j|D_{M,N}^+)P(D_{A,N}^+, N_i|Mj) && \text{(A.10)} \end{aligned}$$

where α_i is constant with respect to M_j, but not so with respect to N_i, to obtain α_i we observe that $\sum_j P(M_j|D, N_i) = 1.0$. Now:

$$\begin{aligned} P(D_{A,N}^+, N_i|M_j) &= \sum_k P(D_{A,N}^+, N_i|M_j, A_k)P(A_k|M_j) && \text{(A.11)} \\ &= \sum_k P(N_i|M_j, A_k)P(D_{A,N}^+|A_k)P(A_k|M_j), && \text{(A.12)} \end{aligned}$$

the variables A and M are independent given no information beneath their common son N, so we may write:

$$P(D_{A,N}^+, N_i|M_j) = \sum_k P(N_i|M_j, A_k)P(D_{A,N}^+|A_k)P(A_k), \tag{A.13}$$

or,

$$P(D_{A,N}^+, N_i|M_j) = \sum_k P(N_i|M_j, A_k)P(A_k|D_{A,N}^+)P(D_{A,N}^+), \tag{A.14}$$

Finally, treating $P(D_{A,N}^+)$ as a constant, we obtain:

$$P(D_{A,N}^+, N_i|M_j) = \alpha_i \sum_k P(N_i|M_j, A_k)P(A_k|D_{A,N}^+), \tag{A.15}$$

or

$$P(D^+_{A,N}, N_i|M_j) = \alpha_i \sum_k \pi_N(A_k) P(N_i|M_j, A_k). \tag{A.16}$$

Combining the result of equation A.16 with equation A.10 we obtain:

$$s^{N,M}_{ij} = \alpha_i P(M_j|D^+_{M,N}) \sum_k P(N_i|M_j, A_k)\pi_N(A_k). \tag{A.17}$$

so,

$$s^{N,M}_{ij} = \alpha_i \pi_N(M_j) \sum_k P(N_i|M_j, A_k)\pi_N(A_k). \tag{A.18}$$

The term α_i can be obtained by normalizing the rows in the sensitivity matrix $s^{N,M}$.

□

Derivation A.2 *Proof of equation 4.23*

$$s^{N,M}_{ij} = P(M_j|D, N_i) \tag{A.19}$$

Applying Bayes and decomposing D into its components:

$$\begin{aligned} P(M_j|D, N_i) &= \frac{P(N_i, D|M_j)P(M_j)}{P(N_i, D)} & \text{(A.20)} \\ &= \frac{P(N_i, D^+_{B,M}, D^-_M|M_j)P(M_j)}{P(N_i, D)}, & \text{(A.21)} \end{aligned}$$

The data beneath M is separated from the rest when M is instantiated, so:

$$\begin{aligned} P(M_j|D, N_i) &= \frac{P(N_i, D^+_{B,M}|M_j)P(D^-_M|M_j)P(M_j)}{P(N_i, D)} & \text{(A.22)} \\ &= P(D^-_M|M_j)P(M_j|N_i, D^+_{B,M})\frac{P(N_i, D^+_{B,M})}{P(N_i, D)}, & \text{(A.23)} \end{aligned}$$

separating D^-_M into its components and considering that they are mutually independent given M, we may write:

$$P(M_j|D,N_i) = \alpha_i P(M_j|N_i, D^+_{B,M}) P(D^-_{M,X}|M_j) P(D^-_{M,Y}|M_j). \tag{A.24}$$

which can be rewritten as:

$$P(M_j|D,N_i) = \alpha_i P(M_j|N_i, D^+_{B,T}) \lambda_X(M_j) \lambda_Y(M_j) \tag{A.25}$$

now:

$$P(M_j|N_i, D^+_{B,T}) = \sum_k P(M_j|N_i, B_k, D^+_{B,T}) P(B_k|N_i, D^+_{B,T}), \tag{A.26}$$

but we know that M_j is independent of $D^+_{B,T}$ given B, and also that B is independent of M given no information underneath M, therefore we conclude that:

$$P(M_j|N_i, D^+_{B,T}) = \sum_k P(M_j|N_i, B_k) P(B_k|D^+_{B,T}), \tag{A.27}$$

finally combining equations A.27 with A.25, and A.19 we obtain:

$$S^{M,T}_{ij} = \alpha_i \lambda_X(M_j) \lambda_Y(M_j) \sum_k P(M_j|N_i, B_k) \pi_T(B_k) \tag{A.28}$$

$\square$

Derivation A.3 *Proof of equation 4.24:*

Analogous to equation A.10 we may prove that:

$$P(M_j|D,N_i) = \alpha_i P(M_j|D^+_{R,N_P}) P(D^-_{R,X}, D^-_{R,Y}, N_i|M_j) \tag{A.29}$$

where:

$$P(D^-_{R,X}, D^-_{R,Y}, N_i|M_j) = \sum_k P(D^-_{R,X}, D^-_{R,Y}, N_i|M_j, R_k)P(R_k|M_j) \qquad \text{(A.30)}$$

considering that the data underneath R is independent of N and R given R, we may write:

$$P(D^-_{R,X}, D^-_{R,Y}, N_i|M_j) = \sum_k P(D^-_{R,X}|R_k)P(D^-_{R,Y}|R_k)P(N_i|M_j, R_k)P(R_k|M_j) \qquad \text{(A.31)}$$

then, applying Bayes:

$$P(D^-_{R,X}, D^-_{R,Y}, N_i|M_j) = \sum_k P(D^-_{R,X}|R_k)P(D^-_{R,Y}|R_k)P(R_k|M_j, N_i)P(N_i|M_j) \qquad \text{(A.32)}$$

knowing that $P(N_i|M_j) = P(N_i)$ we may write:

$$P(D^-_{R,X}, D^-_{R,Y}, N_i|M_j) = \beta_i \sum_k \lambda_X(R_k)\lambda_Y(R_k)P(R_k|M_j, N_i) \qquad \text{(A.33)}$$

from equations A.33 and A.29 we conclude that:

$$s^{N,R}_{ij} = \alpha_i \pi_R(M_j) \sum_k \lambda_X(R_k)\lambda_Y(R_k)P(R_k|N_i, M_j). \qquad \text{(A.34)}$$

□

APPENDIX B

BAYNET

B.1 Introduction

In this appendix we describe the most important aspects of the system built to handle the propagation scheme proposed in this thesis. **BAYNET** is a system that has been designed as a graphical tool for the design of Bayesian networks. **BAYNET** has been implemented on Apollo workstations in UCLA's Computer Science Department using the T language[Rees84], which is a dialect of LISP. We made extensive use of two software packages: (1) The T-*Flavors* package, which implements the notion of Flavors[Wein81], (2) and the GATE environment[Muel84], which provides primitive objects to handle graphs on the screen of the workstation.

BAYNET has been built using an object oriented programming approach. We defined two classes of objects:

1. **pnode:** Each node in a Bayesian network corresponds to a pnode object in BAYNET. The Bayesian network is represented as a collection of instances of pnodes. Each instance contains pointers to the immediate neighbors of the node it represents, so that the network can be traversed following these

links.

2. tensor: This object type has been defined to work with multidimensional matrices. The computation and update of the belief parameters and sensitivity matrices are done using the functions defined for the tensor flavor.[1] Thus, it is essential to understand the definition of this object in order to comprehend the implementation of the system.

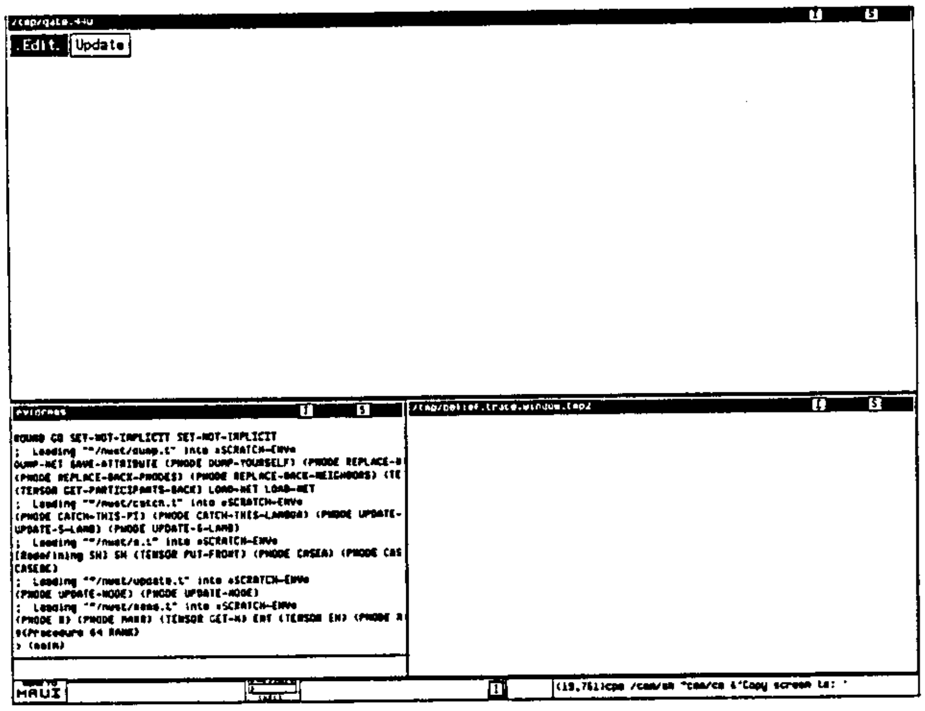
Here we review the implementation of these object types, emphasizing those aspects that are more relevant from the point of view of the design of the system. In section B.2 we provide an overview of the system interface, to give a flavor of how the system actually works. In sections B.3 and B.4, we describe the attributes of a pnode object and the functions or methods defined to manipulate these attributes. In section B.5 we explain how the sensitivity matrices are computed. Section B.6 describes the user interface. Finally, in section B.7 we describe the most important characteristics of the tensor object, and we use the computation of the belief parameters of a given node in a Bayesian network as an example.

B.2 Overall Description and Example.

In this section we describe how a user should interact with *BAYNET* in order to build and display a Bayesian network and how data is entered and propagated through it.

[1]The terms *flavor*, *object type* and *class* will be used interchangeably.

After loading the system, the user has three windows (see figure B.1): (1) bottom left is the T window for direct interaction with the T interpreter; (2) at the top is the Bayesian network drawing window (BND) and (3) an extra window E for textual information, mainly for explanation and debugging purposes.



```
BOUND CO SET-NOT-IMPLICIT SET-NOT-IMPLICIT
;  Loading "~/nuet/dump.t" into *SCRATCH-ENV*
DUMP-NET SAVE-ATTRIBUTE (PNODE DUMP-YOURSELF) (PNODE REPLACE-B
(PNODE REPLACE-BACK-PNODES) (PNODE REPLACE-BACK-NEIGHBORS) (TE
(TENSOR GET-PARTICIPANTS-BACK) LOAD-NET LOAD-NET
;  Loading "~/nuet/catch.t" into *SCRATCH-ENV*
(PNODE CATCH-THIS-PI) (PNODE CATCH-THIS-LAMBDA) (PNODE UPDATE-
UPDATE-S-LAMB) (PNODE UPDATE-S-LAMB)
;  Loading "~/nuet/a.t" into *SCRATCH-ENV*
[Redefining SH] SH (TENSOR PUT-FRONT) (PNODE CASEA) (PNODE CAS
CASEBC)
;  Loading "~/nuet/update.t" into *SCRATCH-ENV*
(PNODE UPDATE-NODE) (PNODE UPDATE-NODE)
;  Loading "~/nuet/name.t" into *SCRATCH-ENV*
(PNODE B) (PNODE RANK) (TENSOR GET-N) ENT (TENSOR EN) (PNODE R
#{Procedure 64 RANK}
> (main)
```

Figure B.1: Starting State

The system can exist in two states: in an *interaction loop* (IL), where the user interacts with the system through the BND window using the mouse device; or in a T-interaction state, where the user interacts with the T interpreter through the T window using the keyboard.

In turn, the BND window can be in two modes, *edit* or *update*. In *edit* mode (where the system starts), the user may draw a Bayesian network utilizing the mouse device. To do so, the left button of the mouse "pops up" a menu, which provides items for creating nodes and links between them. For example, figures B.2 and B.3 show the sequence of steps leading to the two node network of figure B.3b.

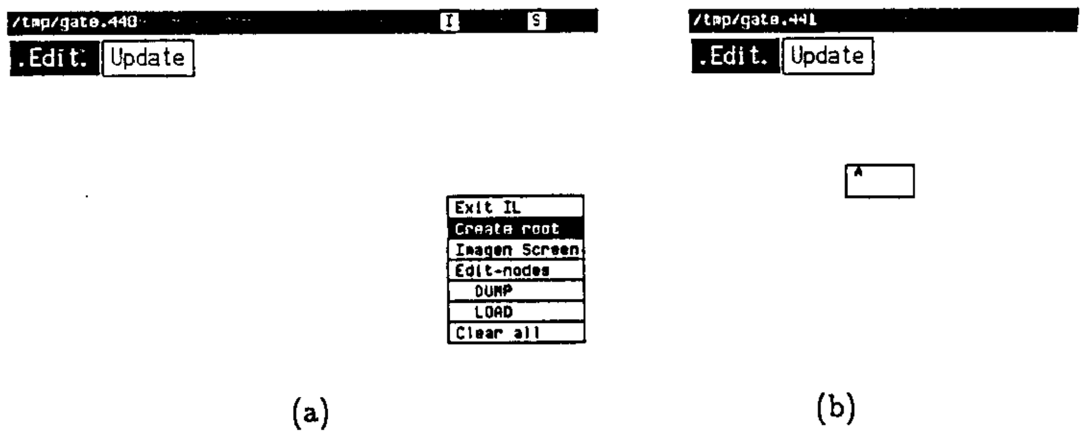


Figure B.2: Creating a Single Node, (a) selecting the menu item and (b) the created node

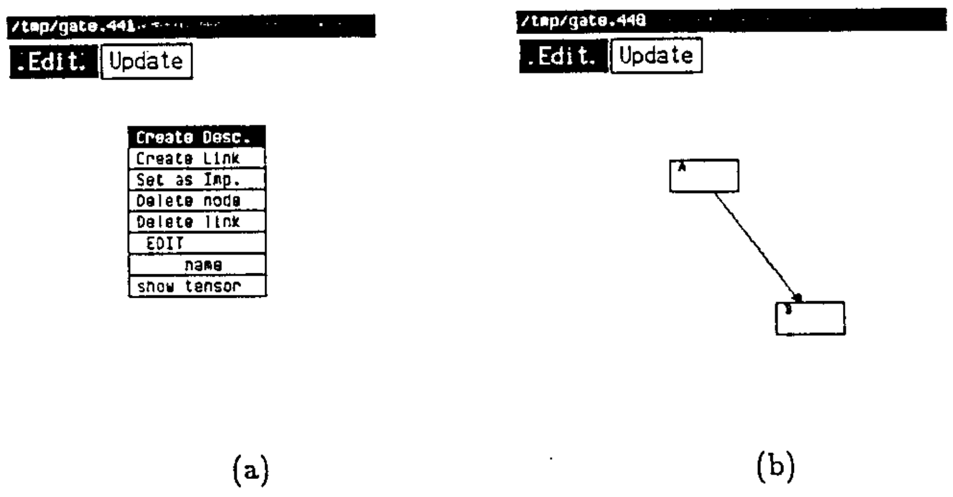


Figure B.3: Adding a Descendant to Node *A*

Following an identical process, we build the network of figure B.4.

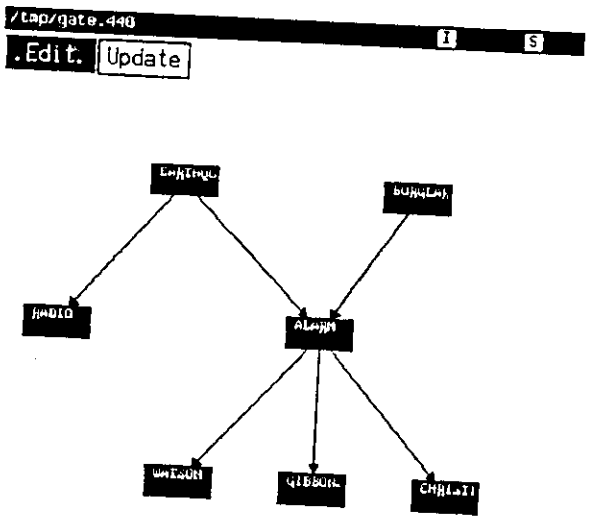
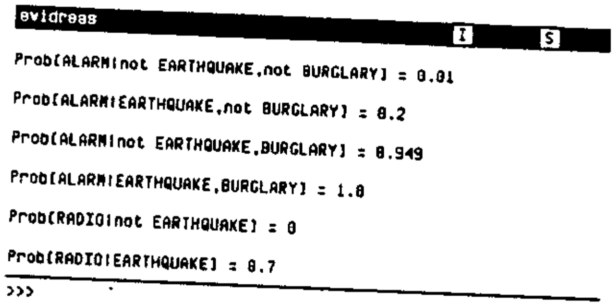


Figure B.4: Complete Bayesian Network

Once the network is complete, we edit the link parameters, stored at each node. To do so, we select a node for editing and provide the information requested on the *T* window (see figure B.5).



```
Prob[ALARM|not EARTHQUAKE,not BURGLARY] = 0.01
Prob[ALARM|EARTHQUAKE,not BURGLARY] = 0.2
Prob[ALARM|not EARTHQUAKE,BURGLARY] = 0.949
Prob[ALARM|EARTHQUAKE,BURGLARY] = 1.0
Prob[RADIO|not EARTHQUAKE] = 0
Prob[RADIO|EARTHQUAKE] = 0.7
>>>
```

Figure B.5: Entering Conditional Probability Parameters

In this example, nodes *Watson* and *Gibbons* represent implicit evidence nodes. We specify so by selecting the *"Set as Imp."* menu item (see figure B.2b).

Once all the parameters have been defined, we may change to *update* mode (by clicking the left button of the mouse on the *update* box). Immediately after moving to update mode, **BAYNET** starts propagating messages through the network until it reaches equilibrium. At equilibrium, all nodes will display the belief on their corresponding propositions (see figure B.6). So, in this case the belief in the burglary would be 0.0699.

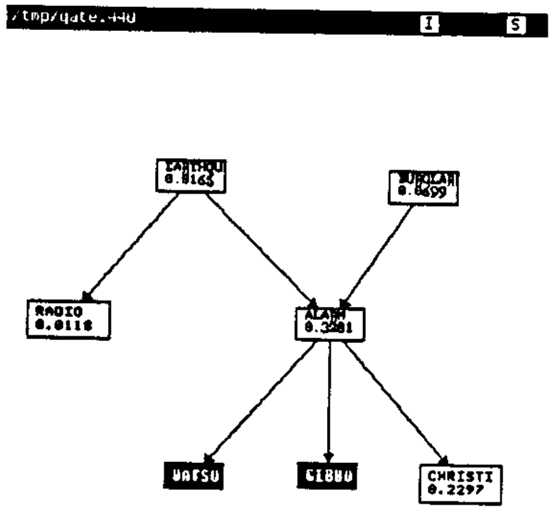


Figure B.6: Initial State of the Burglary Bayesian Network

The situation of figure B.6 corresponds to the moment at which neither a call from Holmes' daughter, Christie, has been received, nor radio news has been broadcasted. Now, we may want to determine which sensory node, R or C, is more important for assessing the burglary node. First, we issue the command "(send ^*burglary* 'target)" on the T window, identifying the *burglary* node as the target. This message is propagated through the network, computing all the sensitivity matrices. Second, the message "(send ^⟨*node*⟩ 'r)" will return ⟨*node*⟩'s relevancy measure, as given by equation 4.36,

$$\begin{aligned} R(c) &= 0.038 \\ R(r) &= 0.000421 \end{aligned}$$

This shows that there would be less uncertainty (and more information) by knowing whether there was a radio broadcast than by knowing whether Christie has called. The small value for $R(r)$ can be easily explained, considering the very low belief of its only possible cause.

On the other hand, if we receive a confirmation that Christie has called, we may want to determine how relevant this new information is to the target proposition (the burglary). This relevancy is determined using equation 4.32, where $P(T_k|D)$ is simply the current value of $Bel(T_k)$, and where:

$$P(T|D,C) = (0 \;\; 1) \cdot S^{C,B} \tag{B.1}$$

Hence, we obtain:

$$M_1(T|D,C) = 0.2641 \tag{B.2}$$

which is a measure of the information rendered by the instantiation of node C.

The E window is used to display messages that are intended to explain the reasons for the modification of the belief parameters of certain nodes. At this time, the explanations consist of the specification of the neighbors which sent messages which affected the belief change.

In the subsequent sections of this chapter we provide a more detailed description of the implementation of this program.

B.3 Pnode Object Type.

When we want to define a node for a particular Bayesian network, we create an instance of a *pnode* flavor. A pnode is a specialization of the *web* flavor defined in GATE. It inherits all the properties of webs (for details see [Muel84]), including the necessary features for drawing nodes on the screen of the workstation, drawing links between the nodes and handling "pop-up menus", etc.

Once the Bayesian network has been drawn, the program structures all the information it needs on the attributes of the pnode class. For each node N the following attributes are kept in its associated pnode object:

1. *tensor*: This object, of type tensor, represents the conditional probability matrix of the node, given its parents, or in the case of nodes with no ancestors, it represents the prior probability vector. This matrix is entered interactively upon user request.

2. *fathers*: This is a list of one element for each parent of the node, each of the form "$(\langle p\rangle, \pi_N(\langle p\rangle), \langle u\rangle)$", where $\langle p\rangle$ is a pointer to the corresponding parent, $\pi_N(\langle p\rangle)$ is the last π message received from this parent, and $\langle u\rangle$ is a boolean variable that is true only if $\pi_N(\langle p\rangle)$ has not been included in the last computation of the belief vector of N.

3. *sons*: This is a list with one element for each son of N, each element of the form "$(\langle s\rangle, \lambda_{\langle s\rangle}(N), \langle u\rangle)$," where $\langle s\rangle$ is a pointer to the son, $\lambda_{\langle s\rangle}(N)$ is the

last λ message received from this son and $\langle u \rangle$ is a boolean variable that is true only if $\lambda_{\langle s \rangle}(N)$ has not been included in the last computation of the belief vector of N.

4. *implicit?*: Boolean variable which is true for those leaves representing implicit nodes.

5. *excited?*: Boolean variable that is true whenever there are outstanding messages (λ's or π's) that have not been included in the last update of the belief vector, i.e., it is true when the constraints between the node and each of its immediate neighbors are unrelaxed.

6. *S*: The sensitivity matrix of the node with respect to the target node. It is initialized as the identity matrix.

7. *belief*: The belief vector of the node, which is represented as a simple list equal to $(\underline{Bel}(\neg N) \quad \underline{Bel}(N))$.

8. *par-mod?*: Boolean variable that is true whenever the parameters of the node (conditional probability matrix) have been changed.

9. *data*: Used only for leaf nodes, it represents the data available at sensory nodes. It has a value (1 1) if the node has not been instantiated, a value (0 1) if it has been confirmed, and a value of (1 0) if it has been denied.

10. *path-node*: A pointer to the node's path-neighbor.

In addition to the aforementioned attributes of the pnode object type, we have defined a set of methods, which will be better described in the context of an example.

B.4 Methods for the *pnode* Flavor.

Suppose that we want to enter the Bayesian network for the example in figure 3.1. In this section we will describe the sequence of steps involved in this process, especially from a system viewpoint, i.e., we shall describe the functions and methods invoked through the process.

The first step is to invoke the T interpreter and load the appropriate programs. The command "(main)" executed on the T window will start the system, which will automatically create a second window of reasonable size to draw the Bayesian network. Now, the system gets into an *interaction loop* phase, in which it will recognize only commands issued with the *mouse* device. The GATE environment provides methods for handling *menus*, so that we obtain a menu by pressing a button of the mouse. The entire process, from creation of the Bayesian Network to subsequent interaction with the system, can be done solely by use of the mouse. However, the description below will be done in terms of T function calls, which may be issued from the T window or from any function in the T environment. So, the description would be useful for implementing extra capabilities to the running system.

B.4.1 Setting up a Bayesian Network

BAYNET can be in either of two states, *edit* or *update*. The user can switch from one state to the other as many times as necessary. However, it should not be possible to switch from *edit* to *update* state unless the network is completely defined. To do this, the system provides different menus of commands in both states, however; these menus can be bypassed by calling the functions directly. In the *edit* state we can draw and modify the topology of the network, while in the *update* state we can enter data and propagate it through the network. The parameters of every node can be modified in both states.

The following is a list of functions available to set up a Bayesian network:

1. **create-node**, (parameters: ask-name?). If ask-name? is false (nil), it generates a name for the node; otherwise, it asks the user for one. The function will wait for the user to click with the mouse on the position in the screen where the node is to be depicted. It then creates a node in that position, giving it a rectangular shape, displaying the name of the node in the rectangle itself. The system keeps a list of pointers to all active nodes in the system, which is updated whenever a new node is created. Each pointer will have the name "^$\langle name \rangle$," where $\langle name \rangle$ is the name given to the node to which it points. When the pointer is evaluated, it returns the actual object to which it is pointing.

   In the burglary example, we call this function once for every node. So, issuing the command "(create-node t)" will makq the system prompt us for the name of the node (say *burglary* for the burglary node) and create the node wherever the mouse is clicked.

2. **create-link**, (parameter: pnode). Once we have more than one node on the screen, we may link any two nodes by calling this function. Then, "(create-link ⟨pnode⟩)" will make the system wait for the user to click in a node, say ⟨pnode2⟩ and create a link from ⟨pnode⟩ to ⟨pnode2⟩. Besides drawing an arrow between the two nodes, BAYNET will update the sons and fathers attributes of the nodes involved in the linkage.

3. **create-descendant**, (parameters: pnode, ask-name?). This is equivalent to consecutively creating a node (with create-node) and a link (with create-link) to that node, with its tail at pnode.

4. **set-implicit**, (parameter: pnode). This function converts a node to *implicit*. It should be applied only to a leaf node. As a side effect, the function changes the size of the node and the font type used for the text inside it.

5. **delete-node**, (parameter: pnode). It deletes the node received as a parameter and also deletes all the links to or from this node.

6. **delete-link**, (parameters: pnode). It deletes the link that starts at the node specified in the pnode parameter and ends at the node where the mouse is clicked.

7. **edit-node**, (parameter: pnode). This function asks for the name of the node and for its conditional probability matrix with respect to its parents.

B.4.2 Entering Data

To enter data into the network, the following functions are defined:

1. **set-data**, (parameters: pnode, d). When this function is called, the pnode is either instantiated as true (if d is true), or instantiated to false (if d is nil). This function should be applied only to sensory nodes.

2. **clear-evidence-data**, parameters: pnode. It resets the data attribute of a sensory node to (1 1).

3. **change-tensor**, (parameter: pnode). This function is called to change the conditional probability matrix of a node once in the "update" state.

4. **create-aux-evid-node**. There are situations where evidence impinges directly on a node that has not been considered in the network. For example, suppose that Holmes receives a call from a friend, who comments on a minor earthquake he felt around the time the alarm is said to have gone off. The network does not include a node to account for this event. In this situation we may add an "auxiliary evidence node" as a son of the "earthquake" node since it provides evidence impinging directly on the earthquake.

B.4.3 Propagation

The Bayesian network approach permits update of the belief vectors of the nodes in the network in a completely asynchronous way. We have implemented the following update mechanisms:

1. **recency-driven propagation:** When the recency driven propagation is called, all the nodes that have outstanding messages (i.e., that have their *excited?* attribute as true) are updated. To update, we simply send an *update* message (described later on). After receiving it, the node computes: (1) its new belief vector and (2) new messages to send to its neighbors. The new messages are delivered to the neighbors, exciting them. After a first phase, the nodes that were originally excited become updated, and other nodes are left excited. This process is continued until all nodes in the network have been updated.

   Let L be the list of all the nodes in the network. This update mechanism can then be expressed as:

   ```
   (do () ((any-excited-nodes? L))
     (walk (lambda (n)
                (if (send n 'excited?)
                    (send n 'update)))
      L))
   ```

where the "any-excited-nodes?" predicate would check if there is still an excited node in the list L.

2. **fixed-order propagation:** Establish an arbitrary ordering among the nodes and update each one in that order. This update is repeated until a cycle through all the nodes is completed without finding any excited node.

3. **random-update:** Choose an arbitrary node in the network and send it an *update* message. Repeate until n consecutive *update* messages are received by non-excited nodes, where n is a user constant.

4. **selective-update:** Send the *update* message to a chosen node.

The implementation of these update mechanisms makes use of the *update-node* method, which is sent to the nodes requiring update. Next, we define this method, along with a set of methods that allow the computation of π and λ messages and establish the communication among the different nodes. These are:

1. **update-node** (parameters: none) - When a node receives the *update-node* message, it executes the following sequence of steps:

   - compute and send all λ messages to its descendants;
   - compute and send all π messages to its ancestors;
   - update its own belief vector.

2. **send-pi**, and **send-lambda** (parameters: $\langle dest \rangle$) - The parameter $\langle dest \rangle$ indicates who is going to receive the corresponding π or λ. These methods would compute the message to send to $\langle dest \rangle$ and would send the message *catch-pi* or *catch-lambda* to $\langle dest \rangle$ with the corresponding updated message as parameter.

3. **catch-pi**, and **catch-lambda** (parameters: sender, v) - The parameter *sender* is a pointer to the originator of the message, and v is the actual message (π or λ). The node that receives the π or λ compares it with the last message received from the same source. If the difference between the two is very small, the message is disregarded. If the message is not disregarded, then the node updates its parameters (rendering *excited?* true).

B.5 Computing and Updating the Sensitivity Matrices

B.5.1 New Attributes of *pnode*

In this section we describe the methods defined to compute and update the sensitivity matrices of every node in the network. The following attributes have been added to the *pnode* flavor:

1. S, the sensitivity matrix of the node with respect to the target node.

2. *path-node*, a pointer to the path-neighbor of the node. It is initialized as nil.

3. *summary*, a matrix introduced to simplify the computation of the sensitivity matrix. It will be described later on.

4. LP, lambda product, is the component-by-component product of the lambda messages received from the sons (see equation B.5).

The summary matrix just introduced is used to perform a more efficient computation of the sensitivity matrices. In the current version of the program, it is also used to compute the λ and π parameters that each node sends to its direct ancestors and descendants.

The summary matrix for node A in the example of figure 3.1 is defined as:

$$U_{ijk} = P(A_i|B_j, E_k)\lambda_W(A_i)\lambda_G(A_i)\lambda_C(A_i)\pi_A(B_j)\pi_A(E_k). \tag{B.3}$$

If we have defined the target to be node B, then we may compute:

$$S_{ij}^{A,B} = \alpha_i \sum_k U_{ijk}/LP_i, \tag{B.4}$$

where:

$$LP_i = \lambda_W(A_i)\lambda_G(A_i)\lambda_C(A_i). \tag{B.5}$$

Also, in node A we need to compute and update the sensitivity matrix of node E with respect to the target (given by equation 4.24). That can be computed as:

$$S_{ij}^{E,B} = \sum_k U_{ijk}/\pi_A(B_j). \tag{B.6}$$

So, whenever a new π or λ message is received by the object A, the summary matrix, along with the sensitivity matrices $S^{A,B}$ and $S^{E,B}$, is updated. These equations are not valid when a node receives a π or λ that makes the denominator of these equations equal to zero. In these cases, the sensitivity matrices are computed directly from equations 4.22 and 4.24.

B.5.2 Initialization

After the target node has been chosen, we need to initialize the parameters at each node. The initialization is performed by the *target* method. The process starts when the *target* message is received by the target node. Thus, in our example, the message:

(send ^B 'tensor)

will initialize node B as the tensor and will trigger the initialization process. In order to describe this process, let us assume that we pull the target node up, ignoring the directionality of the links. By doing so, we obtain a tree rooted at the target node. Figure B.7 shows the tree obtained by pulling up node B of figure 3.1.

Once node B receives the *tensor* message, it starts a propagation on breadth-first order (on the tree of figure B.7). This propagation consists of two steps: (1) *identification phase*, where every node receives a message from its path-neighbor to identify itself and (2) initial sensitivity matrix computation. For the second step, it is important to notice that the breadth-first order of propagation is essential since the sensitivity matrix of a node will depend only on the sensitivity matrices of nodes above it on the tree.

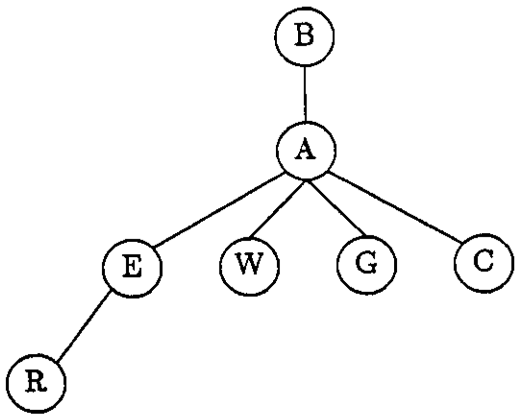


Figure B.7: Tree rooted at the target node

B.5.3 Update

As well as the belief vectors being updated, the sensitivity matrices are also updated whenever new π and λ messages are received by a node. The methods *update-S-pi* (which is invoked when a node receives a π message) and *update-S-lambda* (invoked when the node receives a λ message) update the sensitivity parameters at every node. To update the sensitivity matrix of any node N, we update the sensitivity matrix of N with respect to its closest T-path-cut node (say, M) and then compute $S \leftarrow S^{N,M} \cdot S^{M,T}$. When a node receives a message, the update is performed according to the following rules:

1. N receives a π message, and its path-neighbor is one of its ancestors. In this case, N's object would update: (1) its own sensitivity matrix, according to equation 4.22, and (2) the sensitivity matrices of all its ancestors, excluding the sensitivity matrix of its path-neighbor and the sensitivity matrix of the node that issued the message (which does not need the update).

2. N receives a π, and its path-neighbor is one of its sons. It updates the sensitivity matrices of its immediate ancestors, according to equation 4.23. This update is not done for the node that sent the π message.

3. N receives a λ message, and its path-neighbor is one of its ancestors. Here, N's object updates the sensitivity matrices of its ancestors, except for its path neighbors' sensitivity matrix, according to equation 4.24.

4. N receives a λ message, and its path neighbor is one of its descendants. In this case we update only the sensitivity matrices of N's ancestors, as specified by equation 4.24.

As an example, let us consider node A in figure 3.1. Suppose that Holmes heard the news about an earthquake. In this case, new messages are propagated through the network. When A receives a new π message, it will update its own sensitivity matrix with respect to B (see equation 4.22). If, instead, A had received a λ message from node C, it would have updated only the sensitivity matrix of node E with respect to B.

B.6 The User Interface

GATE provides an environment that includes the capability for graphical interaction with a *menu* driven user interface. Most of the functions and methods that are of common use in BAYNET can be accessed through the menus. As we already mentioned, the system can be in two different states, *edit* and *update*.

The user is allowed to execute a different set of commands in each state; so, different sets of menu items are provided for each state. In BAYNET we have three different menus for each state. These are:

1. *pop-up menu*, which pops up when the user clicks the left button of the mouse in an empty space in the BAYNET window (e.g., figure B.8).

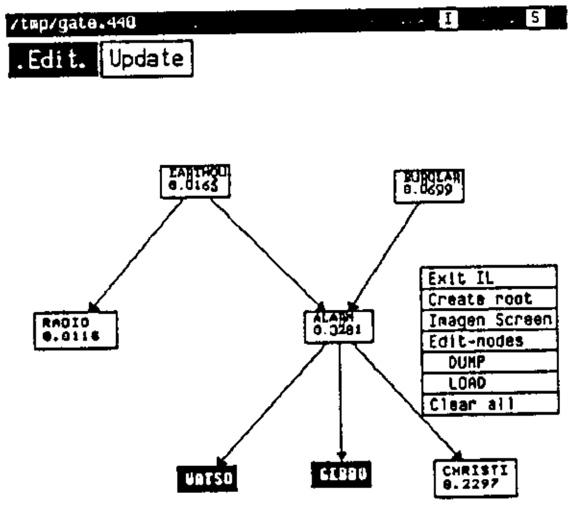


Figure B.8: *Edit* state pop-up menu.

2. *leaf node pop-up menu*, which pops up when the user clicks the left button of the mouse in a leaf node (e.g., figure B.9).

3. *non-leaf node pop-up menu*, pops up when the user clicks the left button of the mouse in any non-leaf node (e.g., figure B.10).

Each menu item is associated with either a function or a method. For example, the *Create Desc.* menu item of figure B.10 is associated with the create-descendant function defined on page 105. So, whenever the user selects the Create Desc.

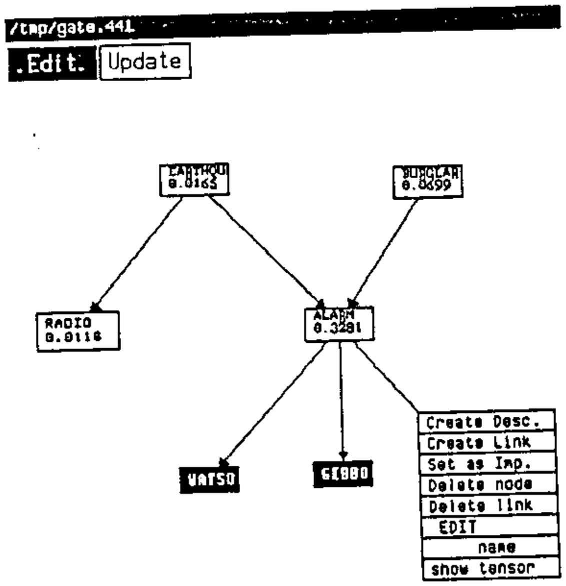


Figure B.9: *Edit* state leaf node pop-up menu.

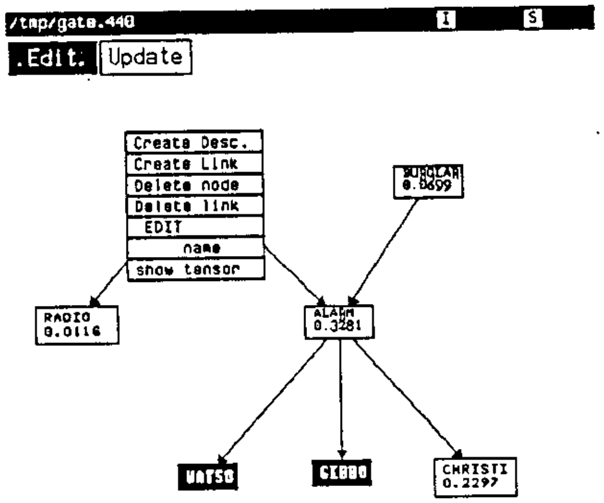


Figure B.10: *Edit* state non-leaf pop-up menu.

item, the function **create-descendant** is automatically invoked, with the node in which the mouse has been clicked as a parameter. The list of menu items that are available is[2]:

1. **Exit IL**: Exits from the interaction loop, returning the control of the program to the user through the T window.
2. **Create root**: Calls the function *create-node* (page 104).
3. **Edit-nodes**: Calls the *edit-node* function (page 106) for every node defined in the network.
4. **DUMP**: This is associated with a "dump" facility. It will preserve a snapshot of the Bayesian network as is at the time of invocation. It will ask for the name of a file.
5. **LOAD**: Will read the contents of a file generated by the dump facility just mentioned.
6. **Clear all**: Deletes all the nodes and links, reinitializing the system.
7. **Create Desc.**: calls the function *create-descendant*, with parameters $\langle n \rangle$ and nil.
8. **Set as Imp.**: Sets $\langle n \rangle$ as implicit. In this case, $\langle n \rangle$ can only be a leaf node.

[2]In this description $\langle n \rangle$ will be the node on which the left button of the mouse has been clicked, this is not used for the pop-up menus.

9. **Delete node:** Calls the function delete-node (page 105) with parameter $\langle n \rangle$.

10. **Delete link.** Calls the function delete-link (page 105) with parameter $\langle n \rangle$

11. **EDIT:** Edits $\langle n \rangle$, asking for the conditional probability matrix of node $\langle n \rangle$ with respect to its parents. This definition, as well as the ones given with the "Edit-nodes" menu item, is deleted whenever a link from one of its parents is deleted or when a parent is either deleted or added to the node.

12. **name:** Permits a name change for $\langle n \rangle$.

13. **show tensor:** Displays the conditional probability matrix of $\langle n \rangle$ in the T window.

Most of the menu items included in the previous list are available only in the *edit state*. In the following list we include all those menu items that are available in the *update state* and those that are not included for the *edit state*. The three menus available in the update state are shown in figures B.11, B.12 and B.13.

1. **Propagate:** Initiates a propagation of outstanding data through the network, it finishes when the network reaches equilibrium.

2. **recency driv:** Initiates a recency driven propagation (page 107).

3. **fixed order:** Initiates a fixed order propagation (page 108). It follows the order given in a list called "*propositions*," which can be modified by the user. It must always contain one pointer for each node in the network.

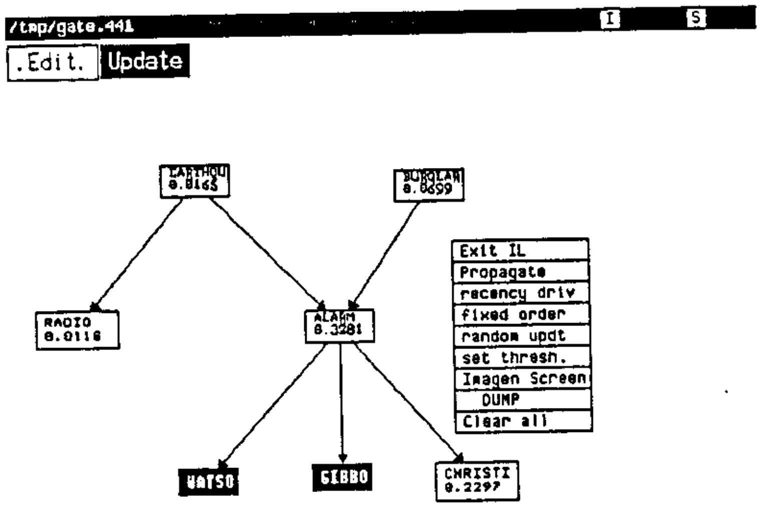


Figure B.11: *Update* state pop-up menu.

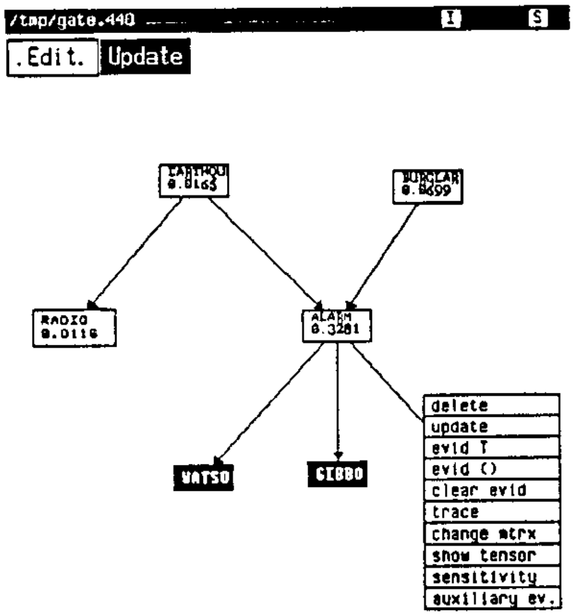


Figure B.12: *Update* state leaf pop-up menu.

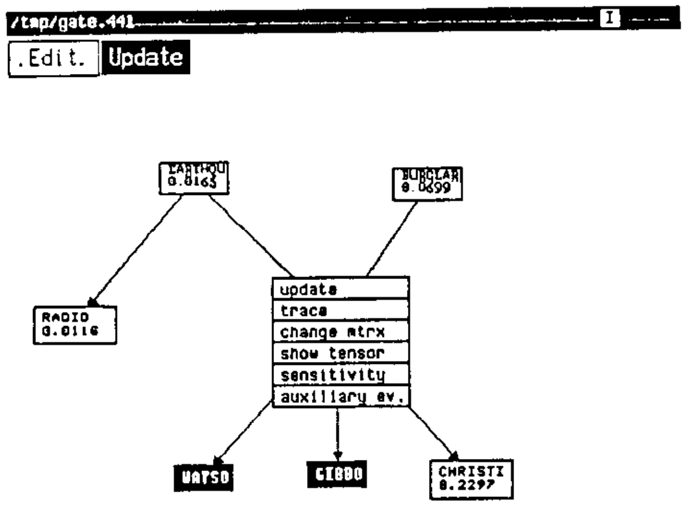


Figure B.13: *Update* state non-leaf pop-up menu.

4. **random updt.** Calls for a random update (page 108).

5. **set thresh..** Compares a new message received by a node with the last message it received from the same source. If the difference between the two does not surpass a certain threshold t, it ignores the message. This item allows redefinition of the threshold t.

6. **delete:** Permits deletion of a leaf node (only), avoiding a change of state from *update* to *edit.*

7. **evid T and evid ().** Are equivalent to the calls "(set-data $\langle n \rangle$ t)" and "(set-data $\langle n \rangle$ nil)" respectively (page 106).

8. **clear evid:** Is equivalent to the call "(clear-evidence-data $\langle n \rangle$)" (page 106).

9. **trace:** BAYNET permits control of an extra window on which we put textual information to identify the sources of changes in the belief vector of selected nodes in the network. To select a node we use this trace item which, in the current implementation, is also used to debug the system.

10. **change mtrx.** Permits change of the conditional probability matrix of $\langle n \rangle$.

11. **auxiliary ev:** Sends $\langle n \rangle$ the message "create-aux-evid-node" (see page 106).

The menu items described in this section are dynamically changed. There are also some items that are slightly modified under certain conditions. For example, if a node has been put on trace, then the menu item *trace* will be changed to *untrace* so that we can remove that node from the trace list. This types of modifications are straighforward, and their use should be evident on the context.

B.7 Tensor Package

In this section we describe the implementation of the *tensor* package, which has been used to represent and manipulate the conditional probability matrices used by the Bayesian network approach. We will also introduce the notation used for the definition and access of *flavors*.

The definition of any object type involves the settings of attributes that each instance of the object type inherits. For example, one attribute of a node object may be the name of the concept for which it stands. Separately, we also specify a set of *methods*, or procedures, that operate on these objects by manipulating its attributes.

Let us take the example of figure 3.1 and see how the conditional probability matrix $P(A|B,E)$ has been represented. From equation 3.10 we have:

$$\begin{aligned} P(A|B,E) &= 1.0 \qquad P(A|\neg B,E) = 0.949 \\ P(A|\neg B,E) &= 0.2 \qquad P(A|\neg B,\neg E) = 0.01 \end{aligned} \tag{B.7}$$

where, $P(\neg A|B_i,E_j) = 1 - P(A|B_i,E_j)$.

To represent this kind of matrix, we use a special object type, called *tensor*, which has been defined as[3]:

```
(define-flavor tensor
               ((value nil)
                (shape nil)
                (index nil)
                (participants nil)))
```

where we have specified that the tensor object has the attributes "value," "shape," "index" and "participants," which are all initialized as nil. The meaning of each one of the attributes is:

1. *shape*: Order and dimension of the matrix being represented. In the example, the matrix has dimension three and order $2 \times 2 \times 2$. Hence, the

[3]This definition is not rigorously correct, but a precise definition will unnecessarily complicate the exposition.

proper value for this attribute is (2 2 2)[4] - where the dimension is implicitly represented as the length of the list.

2. *value*: List containing all the components of the matrix. In our example it will have $2 \times 2 \times 2 = 8$ elements.

3. *index*: Indicates how each component of the matrix has to be accessed. For example, if index is (4 2 1), then the element $M[1,0,1]$ of a matrix M would be in the position $(4\ 2\ 1) \cdot (1\ 0\ 1) = 5$ in the *value* list.

4. *participants*, is an optional list of names that can be assigned at each dimension of the matrix. For example, the conditional probability matrix $P(A|B,E)$ may be implemented with this attribute equal to $(A\ B\ E)$. In our system this list contains pointers to the objects that represent the nodes A, B and E respectively.

Therefore, in the example, the values of the attributes are:

1. shape = (2 2 2), only binary variables
2. value = (0.99 0.8 0.051 0 0.01 0.2 0.949 1)
3. index = (4 2 1)
4. participants = $(\hat{}A\ \hat{}B\ \hat{}E)$

[4]The implementation of the tensor package has not been constrained to binary matrices, hence we could have matrices of any order and dimension.

Since we are considering only binary variables, we decided to access the elements of the matrix by utilizing lists of binary numbers. For example, the index (0 0 0) corresponds to the element $P(\neg A|\neg B, \neg E)$, and the index (1 1 0) is mapped to $P(A|B, \neg E)$. The mapping function that permits to retrieval of an element of the matrix is simply the dot product between the index of the variable and the *index* attribute of the tensor. Thus, to retrieve the element $P(A|B, \neg E)$ from the matrix, we compute the dot product

$$(1 \;\; 1 \;\; 0) \cdot index = 5,$$

mapping to the fifth element (starting from zero) of the *value* list, i.e., to the number 0.949.

To properly set a *tensor* object to represent the matrix of equation B.7, we use the function "make-instance" (provided by the T-*Flavors* package), and assign its result to some variable. For instance, our example matrix would be created as a *tensor* object as:

```
(set tsr (make-instance 'tensor
                        'shape '(2 2 2)
                        'participants '(^A ^B ^E))
```

In addition to defining the attributes of the tensor object, we also define a set of methods or procedures that implement diverse functions for handling multidimensional matrices. These methods manipulate the attributes defined above. The definition of a method is similar to the definition of a normal function;

its main difference is the way in which the method is invoked. Let us assume that we have defined the method "met," and that we want to apply this method to the object "tsr." To do so, we *send* the *message* "met" to the object "tsr":

$$\text{(send tsr 'met . } \langle\text{parameter-list}\rangle\text{)}$$

where ⟨parameter-list⟩ is a list of the parameters to the method if there are any. Among the set of methods defined for the tensor package, we have:

1. *init*: The method executed whenever a tensor instance is created with the "make-instance" call. It is not explicitly invoked from any place in the system. This method automatically initializes the *value* and *index* attributes.

2. *ask-tensor*: The method called when a tensor is going to be read from the terminal. It has been defined only for binary probability matrices.

3. *tensor-get*: By calling "(send tsr 'tensor-get '(0 0 1))," would return the element $P(\neg A|\neg B, E)$.

4. *tensor-set*: By calling "(send tsr 'tensor-set '(0 1 1) $\langle v \rangle$)," would modify the value of the component (0 1 1) of the matrix, assigning it the value $\langle v \rangle$.

5. *tranpose*: Transposes the matrix represented by the tensor. For example, after the call "(send tsr 'transpose 'E 'B)," the call mentioned in 3 would return the element $P(\neg A|\neg E, B)$ instead of $P(\neg A|\neg B, E)$. This is easily achieved by modifying the elements of the *index* attribute.

6. *vxt*, a vector by matrix multiplication. For example, "(send tsr 'vxt '(0.9 0.1) 'E)" eliminates the component E of the tensor, making

$$P(A|B) = 0.9 \times P(A|B, \neg E) + 0.1 \times P(A|B, E),$$

After this call, the tensor is modified and would be bi-dimensional (the call of 3 would now be illegal since it specifies a three dimensional index).

7. *vxt-no-sum*, same as *vxt*, but without reducing the dimensionality of the matrix. "(send tsr 'vxt-no-sum '(0.9 0.1) 'E)" would make $P(A|B, \neg E) \leftarrow 0.9 \times P(A|B, \neg E)$, and $P(A|B, E) \leftarrow 0.1 \times P(A|B, E)$.

8. *copy*. Returns a copy of the tensor object which receives the call. This method is used to save a matrix before applying any destructive operations to it (e.g. *"vxt"*).

There are some other methods that have been implemented for a variety of different functions but which are not relevant to the present discussion.

Now, let us assume that the following λ and π messages have been received by the node representing A:

$$\underline{\lambda_C}(A) = (1\ 1) \tag{B.8}$$

$$\underline{\lambda_G}(A) = (0.2\ 0.8) \tag{B.9}$$

$$\underline{\lambda_W}(A) = (0.1\ 0.9) \tag{B.10}$$

$$\underline{\pi_A}(B) = (0.997\ 0.003) \tag{B.11}$$

$$\underline{\pi_A}(E) = (0\ 1) \tag{B.12}$$

which correspond to the situation in which the radio broadcast was transmitted, and Christie has not called up her father.

Using the *tensor* object defined above, we want to determine an expression that would return the belief vector of node A. According to equation 3.9,

$$Bel(A_i) = \alpha\lambda_W(A_i)\lambda_G(A_i)\lambda_C(A_i)\left[\sum_{jk} P(A_i|B_j, E_k)\pi_A(B_j)\pi_A(E_k)\right]. \qquad \text{(B.13)}$$

Now, let us assume that we have the lists:

1. $\Lambda \leftarrow ((\underline{\lambda_C}(A)\ .\ \hat{}C)\quad (\underline{\lambda_G}(A)\ .\ \hat{}G)\quad (\underline{\lambda_W}(A)\ .\ \hat{}W))$,

2. $\Pi \leftarrow ((\underline{\pi_A}(B)\ .\ \hat{}B)\quad (\underline{\pi_A}(E)\ .\ \hat{}E))$.

Then, using the methods we have described, equation B.13 may be computed thus:

```
(define-method (tensor compute-bel) ()
  (walk (lambda (s) (send self 'vxt (car s) (cdr s)))
              Λ)
  (walk (lambda (f) (send self 'vxt-no-sum (car f) (cdr f)))
              Π)
  (normalize (list (send self 'tensor-get '(0))
                   (send self 'tensor-get '(1)))))
```

In order to invoke this method, we issue the message:

```
(send tsr 'compute-bel)
```

which will return a list B, in which:

$$\begin{aligned} Bel(A) &= (cadr\ \ B) \\ Bel(\neg A) &= (car\ \ B) \end{aligned}$$

When this message gets issued, the object to which it is sent (tsr, in this case) executes the corresponding procedures. Thus, references to *self* will be *bound* to the same object (tsr in the example) that received the message. Therefore, the first *walk* sentence would compute the summation in the equation B.13 (as a sequence of vector by matrix products), while the second *walk* would multiply the lambda vectors of A's sons. The result is then normalized to account for the constant α of equation B.13.

Bibliography

[Char83] E. Charniak, The Bayesian Basis of Common Sense Medical Diagnosis (1983).

[Cohe83] P.R. Cohen and M.R. Grinberg, A Theory of Heuristic Reasoning About Uncertainty, *The AI Magazine*, 4(2?) (Summer 1983).

[Coop84] G.F. Cooper, *NESTOR: A Computer-Based Medical Diagnostic Aid that Integrates Causal and Probabilistic Knowledge*, PhD dissertation, Department of Computer Science, Stanford University. (1984).

[Dalk85] N. Dalkey, *Inductive Inference and the Maximum Entropy Principle*, pp. 351–364, D. Reidel Publishing Company (1985).

[Doyl82] J. Doyle, *Some Theories of Reasoned Assumptions: An Essay in Rational Psychology*, Technical Report, Department of Computer Science, Carnegie Mellon University (1982).

[Doyl83] J. Doyle, Methodological Simplicity in Expert System Construction: The Case of Judgements and Reasoned Assumptions, *AI Magazine* (Summer 1983).

[Duda76] R. Duda, P. Hart, and N. Nilsson, Subjective Bayesian methods for rule-based inference systems., pp. 1075–1082, in *Proc. AFIPS Nat. Compt. Conf.* (1976).

[Duda79] R. Duda, J. Gashnig, and P. Hart, Model Design in the Prospector Consultant System for Mineral Exploration, in D. Michie, editor, *Expert Systems in the Microelectronic Age*, Edinburgh University Press (1979).

[Howa84] R. Howard and J. Matheson, Influence Diagrams, in R. Howard and J. Matheson, editors, *The Principles and Applications of Decision Analysis*, Strategic Decisions Group, Menlo Park, California (1984).

[Kim84] J.H. Kim, *CONVINCE: A CONVersational INference Consolidation Engine*, PhD dissertation, Computer Science Department, U.C.L.A. (1984).

[Muel84] E. Mueller and U. Zernik, *GATE Reference Manual*, Technical Report UCLA-AI-84-5, Computer Science Department, U.C.L.A. (October 1984).

[Pear85a] J. Pearl, *Bayes Decision Methods*, Technical Report CSD-850023, R-45., U.C.L.A. Computer Science Deparment (June, 1985.), To appear in Wiley's Encyclopedia of AI.

[Pear85b] J. Pearl, A Constraint-Propagation Approach to Probabilistic Reasoning, pp. 31–42, in *Proceedings of Workshop on Uncertainty and Probability in AI*, U.C.L.A (August 14-16, 1985.), Also in *Uncertainty in AI*, North Holland, 1986.

[Pear85c] J. Pearl, *Fusion, Propagation, and Structuring in Bayesian Networks*, Technical Report CSD-850022, R-42., U.C.L.A. Computer Science Department (April, 1985.), To appear in *Artificial Intelligence*, 1986.

[Rees84] J. Rees, N. Adams, and J. Meehan, *The T Manual*, Computer Science Department, Yale University (January 1984).

[Spie85] D. Spiegelhalter, A Statistical View of Uncertainty in Expert Systems, in *Proc. Workshop on AI and Statistics, Bell Labs.* (April 11-12, 1985).

[Szol78] P. Szolovits and S. Pauker, Categorical and Probabilistic Reasoning in Medical Diagnosis, *Artificial Intelligence*, 11(1-2):115–144 (1978).

[Wein81] D. Weinreb and D. Moon, *Lisp Machine Manual*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, fourth edition edition (1981).

[Weis78] S. Weiss, C. Kulikowski, S. Amarel, and A. Safir, A Model-Based Method for Computer-Aided Medical Decision-Making, *Artificial Intelligence*, 11(1-2):145–172 (1978).