Search for computing posterior probabilities in Bayesian networks

by
David Poole

Technical Report 92-24
September 1992

Department of Computer Science
University of British Columbia
Rm 333 - 6356 Agricultural Road
Vancouver, B.C.
CANADA V6T 1Z2

Search for computing posterior probabilities in Bayesian networks

David Poole
Department of Computer Science,
University of British Columbia,
Vancouver, B.C., Canada V6T 1Z2
poole@cs.ubc.ca
telephone: (604) 822 6254
fax: (604) 822 5485

September 22, 1992

Abstract

This paper provides a general purpose search-based technique for computing posterior probabilities in arbitrary discrete Bayesian Networks. This is an "anytime" algorithm, that at any stage can estimate prior and posterior probabilities with a known error bound. It is shown how well it works for systems that have normality conditions that dominate the probabilities, as is the case in many diagnostic situations where we are diagnosing systems that work most of the time, and for commonsense reasoning tasks where normality assumptions (allegedly) dominate. We give a characterisation of those cases where it works well, and discuss how well it can be expected to work on average. Finally we give a discussion on a range of implementations, and discuss why some promising approaches do not work as well as may be expected.

1 Introduction

This paper provides a general purpose search-based technique for computing posterior probabilities in arbitrary discrete[1] Bayesian networks.

Implementations of Bayesian networks have been placed into three classes [Pearl, 1988; Henrion, 1990]:

1. Exact methods that exploit the structure of the network to allow efficient propagation of evidence [Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Jensen *et al.*, 1990].

2. Stochastic simulation methods that give estimates of probabilities by generating samples of instantiations of the network, using for example a Monte Carlo technique [Henrion, 1990].

3. Search-based approximation techniques that search through a space of possible values to estimate probabilities.

At one level, the method in this paper falls into the exact class; if it is allowed to run to completion, it will have computed the exact conditional probability in a Bayesian network. It, however has the extra feature that it can be stopped before completion to give an answer, with a known error. Under certain probabilistic assumptions (Section 6) it is shown that convergence to a small error is quick.

While the efficient exact methods exploit aspects of the network structure, we instead exploit extreme probabilities to gain efficiency. The method in the paper uses no information on the structure of the network, but rather has a niche for classes of problems where there are "normality"[2] conditions that dominate the probability tables (see Section 6).

There are a number of advances over previous search methods for Bayesian networks [Shimony and Charniak, 1990; Henrion, 1991]. The main advances over [Shimony and Charniak, 1990] are not necessarily in the algorithm itself

[1] i.e. all of the variables have a discrete set of possible values. We do not consider cases where there are continuous variables.

[2] This should not be confused with "normal" as used for Gaussian distributions. We consider systems that have normal operating conditions and only rarely deviate from this normality (i.e., we are assuming abnormality [McCarthy, 1986] is rare). As we are only considering discrete variables, there should be no confusion.

(but see Section 8), but in what we are doing with the algorithm (computing prior and posterior probabilities, rather than just enumerating composite beliefs), the error estimates we obtain and in the average case analysis of the algorithm. This paper should be seen as a dual to the TOP-N algorithm of Henrion [1991]. We have a different niche. We take no account of the noisy-OR distribution that Henrion concentrates on.

Unlike [Shimony and Charniak, 1990] and [Poole, 1991], the algorithm in this paper directly uses the Bayesian network representation rather than transforming it into another representation.

We have actually experimented with a number of algorithms. The one presented here works quite well and is simple to describe and analyse. We were surprised by some of the results we obtained. For example, the ATMS-based implementation and the Prolog-based implementation turn out to not work very well at all. I explain why in Section 8.

2 Background

2.1 Probability

In this section we give a semantic view of probability theory[3] and describe the general idea behind the search method. In some sense the idea of this method has nothing to do with Bayesian networks — we just have to commit to some independence assumptions to make the algorithm more concrete.

We assume we have a set of **random variables** (written in upper case). Each random variable has an associated set of **values**. $vals(X)$ is the set of all possible values of variable X. Values are written in lower case. Both random variables and values will be primitive, not defined in terms of other concepts. An **atomic proposition** is an assignment of a value to a random variable; variable X having value c is written as $X = c$. Each assignment of a value to every random variable is associated with a **possible world**. Let Ω be the set of all possible worlds. Associated with a possible world ω is a

[3]This could have also been presented as joint distributions, with probabilistic assignments to the possible worlds corresponding to joint distributions. If that view suits you, then please read possible worlds as elementary events in a joint distribution [Pearl, 1988, p. 33].

measure $\mu(\omega)$, with the constraint that $\forall\omega,\ 0 \leq \mu(\omega) \leq 1$ and

$$1 = \sum_{\omega \in \Omega} \mu(\omega).$$

We write $\omega \models X = c$ (read "X has value c in world ω") if variable X is assigned value c in world ω. A **proposition** is a logical formula made up of atomic propositions and the usual logical connectives. We define the truth of propositions in a world using the normal truth tables (e.g., $\omega \models \alpha \wedge \beta$ if and only if $\omega \models \alpha$ and $\omega \models \beta$).

Probability is a function from propositions to $[0, 1]$, defined by

$$P(\alpha) = \sum_{\omega \in \Omega : \omega \models \alpha} \mu(\omega).$$

We can define the conditional probability by

$$P(\alpha|\beta) = \frac{P(\alpha \wedge \beta)}{P(\beta)}$$

If β is a conjunction of observations, then $P(\alpha|\beta)$ is the **posterior probability** of α. $P(\alpha)$ is the **prior probability** of formula α.

2.2 Searching possible worlds

For a finite number of variables with a finite number of values, we can compute the probabilities directly, by enumerating the possible worlds. This is however computationally expensive as there are exponentially many of these (the product of the sizes of the domains of the variables).

The idea behind the search method presented in this paper can be obtained by considering the questions:

- Can we estimate the probabilities by only enumerating a few of the possible worlds?
- How can we enumerate just a few of the most probable possible worlds?
- Can we estimate the error in our estimates?
- For what cases does the error get small quickly?
- How fast does it converge to a small error?

This paper sets out to answer these questions, for the case where the distribution is given in terms of Bayesian networks.

2.3 Bayesian Networks

A *Bayesian network* [Pearl, 1988] is a graphical representation of (in)dependence amongst random variables. A Bayesian network is a directed acyclic graph where the nodes represent random variables. If there is an arc from variable B to variable A, B is said to be a parent of A. The independence assumption of a Bayesian network says that each variable is independent of its non-descendents given its parents.

Suppose we have a Bayesian network with random variables $X_1, ..., X_n$. The parents of X_i are written as $\Pi_{X_i} = \langle X_{i_1}, \cdots, X_{i_{k_i}} \rangle$. Suppose $vals(X_i)$ is the set of possible values of random variable X_i .

Associated with the Bayesian network are conditional probability tables which gives the marginal probabilities of the values of X_i depending on the values of its parents Π_{X_i}. This consists of, for each $v_j \in vals(X_j)$, probabilities of the form

$$P(X_i = v_i | X_{i_1} = v_{i_1} \wedge \cdots \wedge X_{i_{k_i}} = v_{i_{k_i}})$$

written as

$$P(X_i = v_i | \Pi_{X_i} = {}^v\Pi_{X_i})$$

where ${}^v\Pi_{X_i} = \langle v_{i_1}, \cdots, v_{i_{k_i}} \rangle$

For any probability distribution, we can compute a joint distribution by

$$P(X_1, \cdots, X_n) = \prod_{i=1}^{n} P(X_i | \Pi_{X_i}).$$

This is often given as the formal definition of a Bayesian network.

This is shorthand for the propositions that the variables have particular values, that is

$$\begin{aligned} & P(X_1 = v_1 \wedge \cdots \wedge X_n = v_n) \\ = & \prod_{i=1}^{n} P(X_i = v_i | \Pi_{X_i} = {}^v\Pi_{X_i}) \end{aligned}$$

An assignment of values to all the variables corresponds with a possible world. For the rest of this paper we will equate a possible world with an assignment of values to all of the variables, and use the two interchangeably (e.g., in using the complete description on the left hand side of the '$\models$').

2.4 Ordering the variables

Suppose we have a Bayesian network with random variables $X_1, ..., X_n$. Assume that the index respects the ordering of the network so that the indices form a total ordering with the parents of a node have a lower index than the node. This can always be done as the nodes in a Bayesian network form a partial ordering. If the parents of X_i are $\Pi_{X_i} = \langle X_{i_1}, \cdots, X_{i_{k_i}} \rangle$, the total ordering preserves $i_j < i$.

Given this ordering, we can use the multiplication rule for probabilities:

$$\begin{aligned} P(X_1, \cdots, X_n) &= P(X_n | X_{n-1}, \cdots, X_1) \\ &\quad \times P(X_{n-1} | X_{n-2} \cdots, X_1) \\ &\quad \vdots \\ &\quad \times P(X_2 | X_1) \\ &\quad \times P(X_1) \end{aligned}$$

The Bayesian network independence assumption says that

$$P(X_i | X_{i-1}, ..., X_1) = P(X_i | \Pi_{X_i})$$

(this relies on the ordering we assumed for the variables).

The reason that we are interested in this ordering is that we can determine the probability of any conjunction of variable given just their predecessors in the total ordering.

3 Enumerating possible worlds in order

3.1 Search Tree

We are now in a position to determine a search tree for Bayesian networks[4].

Definition 3.1 A **partial description** is a tuples of values $\langle v_1, \cdots, v_j \rangle$ where v_i is an element of the domain of variable X_i.

The nodes in the tree are labelled with partial descriptions as follows:

[4]This search tree will correspond to the semantic trees used in theorem proving [Chang and Lee, 1973, Section 4.4], but with random variables instead of complementary literals.

- The root of the tree is the empty tuple $\langle\rangle$.
- The children of node labelled with $\langle v_1, \cdots, v_j \rangle$ are the nodes labelled with $\langle v_1, \cdots, v_j, v \rangle$ for each $v \in vals(X_{j+1})$. In other words the children of a node correspond to the possible values of the next variable in the total ordering.
- The leaves of the tree are tuples of the form $\langle v_1, \cdots, v_n \rangle$.

There is a one to one correspondence between leaves of the tree and possible worlds (or complete assignments to the variables), with the tuple $\langle v_1, \cdots, v_n \rangle$ corresponding to the complete variable assignment

$$X_1 = v_1 \wedge \cdots \wedge X_n = v_n.$$

We associate a probability with each node in the tree. The probability of the node labelled with $\langle v_1, \cdots, v_j \rangle$ is the probability of the corresponding proposition which is

$$P(X_1 = v_1 \wedge \cdots \wedge X_j = v_j) = \prod_{i=1}^{j} P(X_i = v_i | \Pi_{X_i} = {}^v\Pi_{X_i})$$

This is well defined as all of the ancestors of every node have a value in the partial description, by the ordering of the variables.

The following lemma can be trivially proved, and is the basis for the search algorithm.

Lemma 3.2 *The probability of a node is equal to the sum of the probabilities of the leaves that are descendents of the node.*

This lemma lets us bound the probabilities of possible worlds by only generating a few of the possible worlds and placing bounds on the sizes of the possible worlds we have not generated.

3.2 Searching the Search Tree

To implement the computation of probabilities, we carry out a search on the search tree, and generate some of the most likely possible worlds. There are many different search methods that can be used [Pearl, 1984].

$Q := \{\langle\rangle\}$;
$W := \{\}$;
While $Q \neq \{\}$ do
    choose and remove $\langle v_1, \cdots, v_j \rangle$ from Q;
    if $j = n$
      then $W := W \cup \{\langle v_1, \cdots, v_j \rangle\}$
      else $Q := Q \cup \{\langle v_1, \cdots, v_j, v \rangle : v \in vals(X_{j+1})\}$

Figure 1: Basic search algorithm

Note that each node can only be generated once. There is no need to check for multiple paths or loops in the search. This simplifies the search, in that we do not need to keep track of a CLOSED list or check whether nodes are already on the OPEN list (Q in Figure 1) [Pearl, 1984].

Figure 1 gives the generic search algorithms that can be varied by changing which element is chosen from the queue. The idea of the algorithm is that there is a priority queue Q of nodes. We remove one (e.g., the most likely) node at any time, either it is a total description (i.e., where $j = n$) in which case it is added to W, the set of generated worlds, or else its children are added to the queue.

No matter which element is chosen from the queue, this algorithm halts and when it halts W is the set of all tuples corresponding to possible worlds.

We can carry out various search strategies, to enumerate the most likely possible worlds. For example, we can carry out a multiplicative version[5] of A^* search [Pearl, 1984] by choosing the node m from the queue with the highest value of $f(m) = g(m) \times h(m)$. Here $g(m)$ is the probability of the corresponding proposition:

$$\begin{aligned} g(\langle v_1, \cdots, v_j \rangle) &= P(X_1 = v_1 \wedge \cdots \wedge X_j = v_j) \\ &= \prod_{i=1}^{j} P(X_i = v_i | \Pi_{X_i} = {}^{v}\Pi_{X_i}) \end{aligned}$$

[5]This is an instance of Z^* where, instead of adding the costs and choosing the minimum we multiply and choose the maximum. This can be transformed into a more traditional A^* algorithm by taking the negative of the logarithms of the probabilities. We do not do this explicitly as we want the probabilities to add after the search.

$$= P(X_j = v_j | \Pi_{X_j} = {}^v\Pi_{X_j}) \times g(\langle v_1, \cdots, v_{j-1} \rangle)$$

One version of the heuristic function $h(\langle v_1, \cdots, v_j \rangle)$ is the product of the maximum probabilities that can be obtained by variables $X_{j+1} \cdots X_n$ (for any values of the predecessors of these variables). These can be computed by a linear scan (from X_n to X_1) keeping a table of the maximum products.

For most of this paper the heuristic function is not used (i.e. $h(m) = 1$ for all m). Section 7.1, however, discusses how conflicts can be used to generate a heuristic function.

Another alternative is an iterative-deepening A^* [Korf, 1985]. As we are not concerned with finding the most likely possible world, but a set of most likely worlds, we can carry out depth-bounded depth-first searches (not generating nodes whose probability is below a threshold), without worrying too much about decreasing the threshold to the maximum value it could obtain. Appendix B gives an implementation of the depth-bounded depth-first search.

4 Estimating the Probabilities

If we let the above algorithm run to completion we have an exponential algorithm for enumerating the possible worlds that can be used for computing the prior probability of any proposition or conjunction of propositions. This is not, however, the point of this algorithm. The idea is that we want to stop the algorithm part way through, and determine any probability we want to compute.

We use W, at the start of an iteration of the while loop, as an approximation to the set of all possible worlds.

4.1 Prior Probabilities

Suppose we want to compute $P(g)$. At any stage, the possible worlds can be divided into those that are in W and those that will be generated from Q.

$$P(g) = \sum_{w \in \Omega \wedge w \models g} P(w)$$

$$= \left(\sum_{w \in W \wedge w \models g} P(w) \right) + \left(\sum_{w \text{ to be generated from } Q : w \models g} P(w) \right)$$

We can easily compute the first of these sums, and can put both an upper and lower bound on the second (using Lemma 3.2). This means that we can put an bound on the range of probabilities of an goal based on finding just some of the explanations of the goal. Let

$$P_W^g = \sum_{w \in W \wedge w \models g} P(w)$$

$$P_Q = \sum_{t \in Q} P(t)$$

We then have

Lemma 4.1

$$P_W^g \leq P(g) \leq P_W^g + P_Q$$

As the computation progresses, the probability mass in the queue P_Q approaches zero and we get a better refinements on the value of $P(g)$. Note that P_Q is monotonically non-increasing through the loop (i.e P_Q stays the same or gets smaller through the loop). This thus forms the basis of an "anytime" algorithm for Bayesian networks.

4.2 Posterior Probabilities

The above analysis was for finding the prior probability of any proposition. If we want to compute the posterior probability of some g given some observations *obs*, we can use the definition of conditional probability, and use

$$P(g|obs) = \frac{P(g \wedge obs)}{P(obs)}$$

Thus given our estimates of $P(g \wedge obs)$ and $P(obs)$, (namely $P_W^{g \wedge obs}$ and P_W^{obs}) we can consider what happens to elements of the queue:

- They can go towards implying $obs \wedge \neg g$. If all of them go towards implying this, the resultant conditional probability is

$$\frac{P_W^{g \wedge obs}}{P_W^{obs} + P_Q}$$

- They can go towards implying $obs \wedge g$. If all of them go towards implying this, the resultant conditional probability is

$$\frac{P_W^{g \wedge obs} + P_Q}{P_W^{obs} + P_Q}$$

- They can go towards implying $\neg obs$. If all of them go towards implying this, the resultant conditional probability is

$$\frac{P_W^{g \wedge obs}}{P_W^{obs}}$$

The correct answer is a combination of the three cases. As we can easily prove the inequality,

Lemma 4.2

$$\frac{P_W^{g \wedge obs}}{P_W^{obs} + P_Q} \leq \frac{P_W^{g \wedge obs}}{P_W^{obs}} \leq \frac{P_W^{g \wedge obs} + P_Q}{P_W^{obs} + P_Q}$$

It can be proved that $P(g|obs)$ has the following bound:

Theorem 4.3

$$\frac{P_W^{g \wedge obs}}{P_W^{obs} + P_Q} \leq P(g|obs) \leq \frac{P_W^{g \wedge obs} + P_Q}{P_W^{obs} + P_Q}$$

If we choose the midpoint as an estimate, the maximum error is

$$\begin{aligned} & \frac{1}{2}\left(\frac{P_W^{g \wedge obs} + P_Q}{P_W^{obs} + P_Q} - \frac{P_W^{g \wedge obs}}{P_W^{obs} + P_Q}\right) \\ = & \frac{P_Q}{2(P_W^{obs} + P_Q)} \end{aligned}$$

What is interesting about this is that the error is independent of g. Thus when we are generating possible worlds for some observations, and want to have posterior estimates within some error, we can generate the required possible worlds independently of the proposition that we want to compute the probability of.

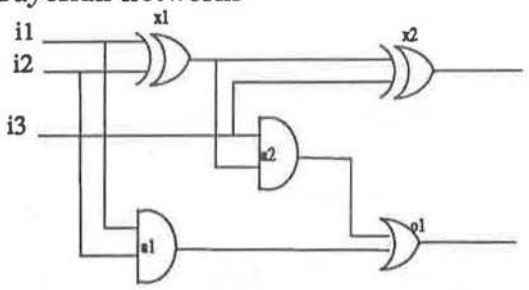

Figure 2: 1 bit adder

5 Diagnosis Example

The preceding section gave an implementation for arbitrary Bayesian networks. In this section we describe how this idea can be applied to diagnosis problems (as in [de Kleer, 1991]). The translation of the circuit into a Bayesian network will follow that of [Pearl, 1988, Section 5.4].

We will do this for a one-bit adder, that can be cascaded to form a multiple-bit adder[6]

5.1 Representation

Figure 2 shows a one bit adder. Figure 3 shows the corresponding Bayesian network.

In this Bayesian network the random variable *out-a2* is a binary random variable that has two values *on* meaning that the output of gate *a2* is on, and *off* meaning the output of the gate *a2* is off. The random variable *a2ok* has 4 values: *ok* meaning that the gate *a2* is working correctly, *stuck1* meaning

[6]There is actually an efficient algorithm for such an example using a clique hypertree representation [Lauritzen and Spiegelhalter, 1988; Jensen *et al.*, 1990]. This exploits the local nature of the propagation, which we do not exploit. These would not work so well when the structure cannot be exploited as well as for the cascaded adders, for example, if we add to the circuit another circuit to find the parity of the resulting bits. We chose this example as it is simple to extend to large systems and also because it was used in [de Kleer, 1991].

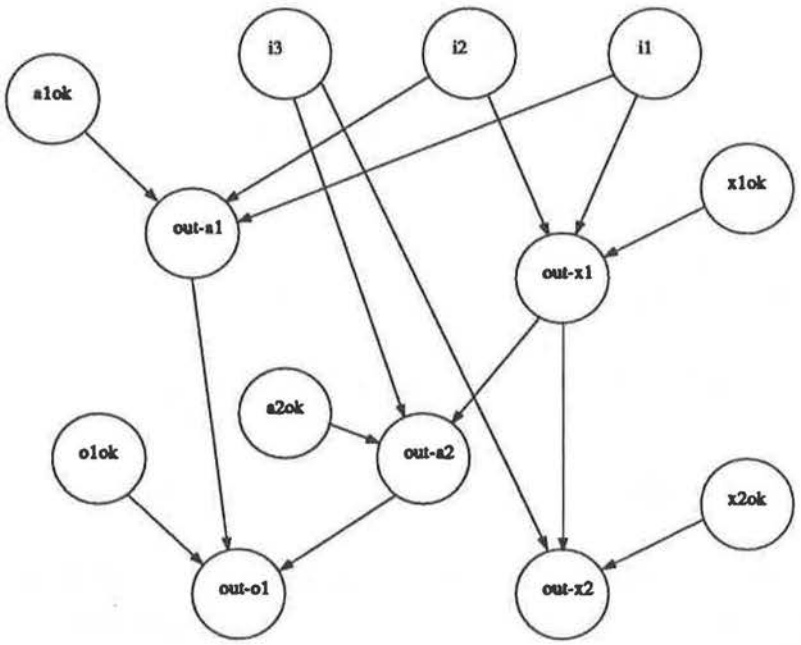

Figure 3: Bayesian network for a 1 bit adder

a2ok	*i3*	*out-x1*	*out-a2*	
			on	*off*
ok	*on*	*on*	1	0
ok	*on*	*off*	0	1
ok	*off*	*on*	0	1
ok	*off*	*off*	0	1
stuck1	–	–	1	0
stuck0	–	–	0	1
unknown	–	–	0.5	0.5

Figure 4: Conditional probability table for variable *out-a2*.

a2ok			
ok	*stuck1*	*stuck0*	*unknown*
0.99999	0.0000049	0.0000049	0.0000002

Figure 5: Conditional probability table for variable *out-a2*.

the gate *a2* is broken, and always has output on, *stuck0* meaning the gate *a2* is broken, and always has output off and *unknown* meaning that it is broken in some different way.

The value of *out-a2* depends on the values of three other variables, *i3*, *out-x1*, and *a2ok*. The values for the variable *out-a2* follow the table in Figure 4. The tables for the other outputs of gates is similar.

The value of *a2ok* does not depend on any other variables. The values for the variable follow the table in Figure 5[7]. The tables for the status of other gates is similar.

These one-bit adders can be cascaded for form multiple bit adders, by connecting the output of gate *o1* in one adder to input *i3* of the following adder. In the Bayesian network, this is done by having multiple instances

[7]The numbers are purely made up. It may seem as though these probabilities are very extreme, but a 1000 bit adder (with 5000 components), is only 95% reliable, if all of the gates are as reliable as that given in this table.

$out\text{-}o1_{k-1}$	$i3_k$	
	on	*off*
on	1	0
off	0	1

Figure 6: Conditional probability table for input 3 of adder k.

of the network for the one-bit adder with the value of $i3$ depending on the variable *out-o*1 for the previous instance of the adder. The table for the probabilities is given in figure 6

5.2 Computation

When we apply the algorithm of Figure 1 to our cascaded adder example, first the world with all gates being *ok* is generated followed by the worlds with single (stuck-at) faults. Most of these can be pruned quickly (see Section 7). Then the *unknown* faults and then the double stuck-at faults, then other mixes of faults in order of likelihood are generated. The probability in the queue converges very quickly. Each of the elements of the queue can be characterized by what errors are in the partial description. We typically only generate the partial descriptions with only a few of the errors.

This is essentially the candidate generator phase of [de Kleer, 1991]. From this candidate generation, we can compute all of the probabilities that we need to. Note that we do not need to find conflicts (but see Section 7.1). There is no need for the ATMS part of [de Kleer, 1991].

6 Complexity

The problem of finding the posterior probability of a proposition in a Bayesian network is NP hard [Cooper, 1990]. Thus we should not expect that our algorithms will be good in the worst case. Our algorithm, when run to completion, is exponential in computing the exact prior and posterior probability of a hypothesis.

Because of the the "anytime" nature of our algorithm, which trades search

time for accuracy, we cannot consider run time independently of error. It is interesting to estimate (or put bounds on) how long it takes to get within some average error, or how accurate we can expect (or guarantee as as asymptotic behaviour) to be within a certain run time.

Because we have probabilities it is possible to carry out an average case complexity analysis of our algorithm.

If we make no assumptions about the probability distributions, the average case of finding the most likely explanation or prior probability within some error ϵ is exponential in the size n of the Bayesian network. This can be seen by noticing that the size of complete descriptions are linear in n and so the probability of explanations is exponentially small in n. This means that we need to consider exponentially many explanations to cover any fixed proportion of the probability mass.

This is not always a reasonable distribution assumption, for example, when using this for diagnosis of a system that basically works we would like to assume that the underlying distribution is such that there is one assignment of values (the "normal values") that dominates the probability mass.

For our analysis we assume that we have extreme probabilities for each conditional probability given. For each value of the parents of variable X_i, we assume that one of the values for X_i is close to one, and the other values are thus close to zero.

When we are computing the probability of a node in the search tree,

$$P(X_1 = v_1 \wedge \cdots \wedge X_j = v_j) = \prod_{i=1}^{j} P(X_i = v_i | \Pi_{X_i} = {}^{v}\Pi_{X_i})$$

we call those elements $P(X_i = v_i | \Pi_{X_i} = {}^{v}\Pi_{X_i})$ that have probability close to zero "*faults*". The others we call *normality* conditions.

Definition 6.1 If we have extreme probabilities (i.e., all of the conditional probabilities are close to one or zero), then the *faults* of node $\langle v_1, \cdots, v_k \rangle$ are those elements v_i such that $P(X_i = v_i | \Pi_{X_i} = {}^{v}\Pi_{X_i})$ is close to zero.

What is a fault is thus context dependent and depends on the values of the parent variables. The term "fault" is derived from its use in diagnosis, where deviation from normality is a fault. The non-monotonic reasoning community has called such things *abnormalities* [McCarthy, 1986]. The algorithm given here can be seen as considering the most normal possible worlds.

Each element of the queue consists of a set of normality conditions and a set of faults. For each set of faults there can be at most one element of Q or W whose faults are exactly that set.

For our complexity analysis, let p ($p \approx 1$) be the probability of the least likely normality condition. We can interpret our analysis as being for the case where all of the normality conditions have value p or as a bound for the cases where all normality conditions are greater than p.

Let $f = 1 - p$. f is a bound on the probability of the faults.

Let b be the average number of fault assumptions for each hypothesis (i.e., $b + 1$ is the number of values for each variable). We assume that b is fixed, and is not a function of the number of variables in the network (n).

6.1 Error convergence

Suppose we have n variables in our Bayesian network. Consider the case where we are about to choose the first k-fault partial description from the queue. We assume for this analysis that $k \ll n$.

At this time there can be no $(k+1)$-fault node on the queue (as we can only generate $(k+1)$-fault nodes from k-fault nodes and we have not yet expanded out first k-fault node), so there are at most $\binom{bn}{k}$ possible combinations of faults on the priority queue (choosing k faults from the set of bn fault assumptions). Each explanation on the queue has a probability less than f^k (not because they all have k faults, but because something with probability less than f^k has been chosen from the priority queue and it was the element with highest probability). Thus the probability mass in the queue can be bounded by $\binom{bn}{k} f^k$. So

$$\begin{aligned} P_Q &\leq \binom{bn}{k} f^k \\ &\leq \frac{(bn)^k}{k!} f^k \\ &= \frac{b^k (nf)^k}{k!}. \end{aligned}$$

Thus, we have convergence $bnf < 1$.

In order to interpret the value of nf, we use the following lemma:

Lemma 6.2 if $(1-f)^n = 1-\delta$, for small δ then $nf \approx \delta$.

See Appendix A for proof of this lemma.

$(1-f)^n = p^n$ is the probability that there are no faults. δ is then the prior probability of some fault in the system. This is low when we are diagnosing a system which works most of the time, and has rare errors.

6.2 How fast is convergence?

To see how fast we reach convergence, consider how long it takes to reach the stage where we are taking the first k-fault hypothesis off the queue. We assume for this analysis that $k \ll n$.

There are at most $(n+1)^{k-1}$ ways to generate less than k faults. For each of these combinations of faults, we go through the while-loop of figure 1, at most n times (for the other normality assumptions). The only thing in the while loop that depends on the size of n, is removing adding and removing elements from the priority queue, this can be done in $\log |Q|$ time.

$$\log |Q| \leq \log \left(\frac{(bn)^k}{k!} \right) = k(\log n + \log b) - \log k! = O(\log n)$$

(for fixed b and k). Thus we can reach this stage in $O(n^k \log n)$ time.

We can combine the above results to give us:

Theorem 6.3 For a fixed k, to attain an accuracy of $\frac{(b\delta)^k}{k!}$ in the estimate of prior probabilities, we require $O(n^k \log n)$ time.

For example, if $b = 1$ (there is only one fault mode) we can attain an accuracy of $\frac{\delta^2}{2}$ in $O(n^2 \log n)$ time. We can attain an accuracy of $\frac{\delta^3}{6}$ in $O(n^3 \log n)$ time.

Theorem 6.4 If δ is small, we can obtain an accuracy of ϵ in time

$$O\left(n^{\frac{\log \epsilon}{\log b\delta}} \log n \right)$$

See Appendix A for the proof.

Thus for fixed ϵ, b and δ, the solution time is polynomial in n.

6.3 Posterior Probabilities

When estimating posterior probabilities, the relative error depends on the relative sizes of P_Q and P_W^{obs}. When $P_W^{obs} \gg P_Q$ the estimate is good.

The error in posterior probability is the difference between the upper and lower estimates for the posterior probability for any g given *obs*. This is

$$\begin{aligned}\frac{1}{2}\left(\frac{P_W^{g\wedge obs}+P_Q}{P_W^{obs}+P_Q}-\frac{P_W^{g\wedge obs}}{P_W^{obs}+P_Q}\right)\\ =\frac{P_Q}{2(P_W^{obs}+P_Q)}\end{aligned}$$

This value is independent of g, and only depends on the observations.

To make the posterior error less than ϵ, we require

$$\frac{P_Q}{P_W^{obs}+P_Q}<2\epsilon$$

this occurs when

$$P_Q<\frac{2\epsilon P_W^{obs}}{1-\epsilon}$$

which can be ensured if we make sure that

$$P_Q<2\epsilon P_W^{obs}$$

Thus we can use the analysis for the prior probability, but multiplying the error bound by a factor that is an estimate of $P(obs)$. As it is unlikely that the observations have a low probability, it is unlikely to have a situation where the error term required is dominated by the probability of the observation.

The following gives a PAC (probably approximately correct) characterization of the complexity[8].

Theorem 6.5 In the space of all systems, to compute the posterior probability of any proposition (of bounded size) given observation *obs*, we can guarantee an error of less than ϵ ($\epsilon<\frac{1}{2}$), at least $1-\psi$ of the cases in time

$$O\left(n^{\frac{\log\epsilon\psi}{\log b\delta}}\log n\right)$$

[8]This has the extra property that we know when we are in a case for which we cannot guarantee the error.

See Appendix A for a proof of this theorem.

Note that in this theorem we are considering "the space of all systems". This means that we consider a random artifact. Most of these have no faults; and presumably would not be the subject of diagnosis. Thus the space of all systems is probably not the space of all systems that we are likely to encounter. A more realistic space of systems by which to judge our average-time behavious is the space of all broken systems, that is those that have at least one fault[9]. We are thus excluding all but δ of the systems. To exclude all but ψ of the broken systems we have to exclude all but $\delta\psi$ of the total number of systems. We thus have the following corollary.

Corollary 6.6 In the space of all broken systems (with at least one fault), to compute the posterior probability of any proposition (of bounded size) given observation *obs*, we can guarantee an error of less than ϵ ($\epsilon < \frac{1}{2}$), at least $1 - \psi$ of the cases in time

$$O\left(n^{\frac{\log \epsilon\delta\psi}{\log \delta\delta}} \log n\right)$$

7 Refinements

There are a number of refinements that can be carried out to the algorithm of Figure 1. Some of these are straightforward, and work well. The most straightforward refinements are:

- We only need to consider the ancestors of the variables we are interested in. If we know our query and the conditioning variables, we don't need to enumerate the values of variables that are not ancestors of these variables.
- If we are trying to determine the value of $P(\alpha)$, then we can stop enumerating the partial descriptions once it can be determined whether α is true in that partial description. In particular when conditioning on our observations we can prune any partial description that is inconsistent with the observations.

[9]It could also be argued that this is also inappropriate; we would rather consider the space of systems that exhibit faulty behaviour. This would be much harder to analyse here, as we have no notion of the observable variables developed in this paper. The space of broken devices seems like a reasonable approximation.

- We do not really require that we find the most likely possible worlds in order, as we are just summing over them anyway. One way to improve the algorithm is to carry out a depth-first depth-bounded search. We can guess the probability of the least likely possible world we will need to generate, use this as a threshold and carry out a depth-first search pruning any partial description with probability less that this threshold. If the answer is not accurate enough, we decrease the threshold and try again. This is reminiscent of iterative deepening A^* [Korf, 1985], but we can decrease the bound in larger ratios as we do not have to find the most likely possible world.

Appendix A gives an implementation in Prolog that incorporates the last two refinements.

7.1 Conflicts

The consistency-based diagnostic approaches have been based on the use of conflicts [de Kleer and Williams, 1987; Reiter, 1987]. We did not use conflicts in the algorithm given above. This turns out to be to the detriment of the algorithm. For example, in diagnosing a large adder, if the error is in the last bits of the adder, the above algorithm will generate k-fault nodes for previous components, even though there cannot be a k-fault error involving only previous components. These earlier values cannot be consistently combined with normality assumptions on latter components, and so get pruned later in the search than they need be.

Conflicts can be incorporated into the heuristic function (see Section 3.2). A conflict here is a set of normality conditions that cannot consistently co-incide. Conflicts discovered in the search can be used to prune the search earlier than it could be pruned without the conflict. If we know that the most likely values of variables $X_{c_1} \cdots X_{c_n}$ cannot co-occur, then the heuristic function for previous variables (in the total ordering; see Section 3.2) should incorporate the maximum value of these variables that can co-occur. I have not implemented this, but it should solve the problem that occurs in the algorithm when we have a conflict in latter variables in the total ordering.

8 Other Approaches Tried

Some of the approaches that I thought would work well are not as good as I had anticipated. We tried an ATMS-inspired approach, and a top-down approach, which were less satisfactory than anticipated.

8.1 ATMS-inspired Approach

The approach based on the ATMS [de Kleer, 1986] has a partial description associated with each node (corresponding to an ATMS environment). The node-ancestor relationship corresponds to a rule that adds a justification (a variable having a value is justified by the ancestors having particular variables, and the conditional probabilities associated with that part of the conditional probability table).

Instead of making label-updating demand driven (as in [de Kleer, 1986]), we make the label-updating in a branch and bound manner, choosing the justification with the highest probability at any time to forward chain on. This approach has been implemented, and while it works, there are a number of problems with it:

We cannot estimate the probability of the queue from summing over the elements of the queue. The reason for this is that the partial descriptions in the labels are not disjoint. This occurs when there are multiple children for any node. It also occurs initially, when each of the topmost ancestors has a label that corresponds to a normality assumption. The sum of these values will typically be greater than one. While this could be solved by not counting the mass on the queue (e.g., by considering the mass of the possible worlds generated and those pruned by inconsistency), there is another, more serious, problem.

The major problem is that the size of the queue grows larger than in the algorithm in Figure 1. Elements of the queue can have much lower probability than those generated by the algorithm in Figure 1 at the corresponding stage, and there are many more elements in the queue. For example, if node X has parents Y_1 and Y_2, and we are forward chaining on k-fault environments. Then a k-fault environment supporting Y_1 and a k-fault environment supporting Y_2 will produce a $(2k)$-fault environment supporting X. The algorithm in Figure 1, only ever produces $(k+1)$-fault "environments" when considering k-faults.

It seems as though the algorithm is fatally flawed because of the last point. It is unclear to me how to solve this problem.

8.2 Prolog-inspired approach

An alternate method is to do some goal-directed search, as in [Shimony and Charniak, 1990; Poole, 1992a]. Here we backward chain from the queries generating as subgoals the set of values that need to be assigned. These are goal-directed, and so can have the focusing advantage of this. They also do not suffer from the problem of having more that $(k+1)$-fault complexes on the queue, when considering k-faults.

They still have the problem of having the elements of the queue being able to sum to more than one. This does not seem to be a severe problem in practice, however. There is also a standard problem with backward chaining, and this is in remembering not to recompute what has been computed for some other subgoal. This is tricky (combining the lemmatization with the branch and bound search), but it can be done.

The main problem is having subgoals that have a number of possible normality conditions that are applicable. This can be most easily seen if we consider generating the 0-fault possible-world. The algorithm in Figure 1 can solve this in linear time; just choosing the normality condition at each time. The backward chaining system cannot do this. The reason is that one value of a variable may be the normality condition for more than one state of the ancestors. We have to search this space. This is like finding a path from the root of a tree to a leaf; it is much more efficient searching from the leaf to the root, as there are no choices involved, rather than searching from the root to find the leaf.

The backward chaining approach offers the advantage of being able to handle more expressive non-propositional languages [Poole, 1992b], for which the algorithm of Figure 1, is not really suited.

9 Comparison with other systems

The branch and bound search, this is very similar to the candidate enumeration of de Kleer's focusing mechanism [de Kleer, 1991]. This similarity to a single step in de Kleer's efficient method indicates the potential of the search

method. He has also been considering circuits with thousands of components, which correspond to Bayesian networks with thousands of nodes.

Shimony and Charniak [Shimony and Charniak, 1990] have an algorithm that is a backward chaining approach to finding the most likely possible world. The algorithm is not as simple as the one presented here, and has worse asymptotic behaviour (as it is a top-down approach — see Section 8.2). It has not been used to find prior or posterior probabilities.

Peng and Reggia [Peng and Reggia, 1990] and Henrion [Henrion, 1991] also consider an abductive definition of diagnosis and incorporate probabilities, and best-first search. They however consider different classes of networks. They fill different niches to the algorithm presented in this paper.

10 Conclusion

This paper has considered a simple search strategy for computing prior and posterior probabilities in Bayesian networks. It is a general purpose algorithm, that is always correct, and has a niche where it works very well. We have characterised this niche, and have given bounds on how badly it can be expected to perform. How common this niche is, is, of course, an open question, but the work in diagnosis and nonmonotonic reasoning would suggest that reasoning about normality is a common task.

The performance results of this algorithm are similar to that of [de Kleer, 1991] (but it is hard to compare due to the different technologies used). Our Prolog implementation (Appendix A) can find the leading diagnoses of a 1000-bit adder (5000 components, 13000 node Bayesian network), in about 73 seconds running Sicstus Prolog on a Next. This is for a similar experiment to [de Kleer, 1991], where all of the inputs were off, and one output bit (in this example bit 5) was on and all of the others were off. These possible worlds can then be used to compute arbitrary posterior probabilities with an error of less than 9% (N.B. the circuit has approximately 5% failure rate). The experimental performance is that the time is linear with the size of the circuit (for a fixed bit that fails). The algorithm is n^2 in the position of the error bit (see Section 7.1, for a description of why and how to fix it).

A Proofs

Lemma 6.2 if $(1-f)^n = 1-\delta$, for small δ then $nf \approx \delta$.

Proof: In particular we prove that $nf \approx \delta$ as $\delta \to 0$.

$$\begin{aligned} 1-\delta &= (1-f)^n \\ &= ((1-f)^{\frac{1}{f}})^{fn} \end{aligned}$$

$f \to 0$ as $\delta \to 0$ (for fixed n). We use the fact that

$$(1-f)^{\frac{1}{f}} \to \frac{1}{e} \quad \text{as} \quad f \to 0$$

So we have, when δ is very small

$$(1-\delta) \approx e^{-nf}$$

taking logarithms of both sides, and using the Taylor expansion at $\delta = 0$, gives us

$$\begin{aligned} -nf &\approx \log(1-\delta) \\ &= 0 - \delta + c\delta^2 + \cdots \\ &\approx -\delta \end{aligned}$$

From which we derive $nf \approx \delta$. □

Theorem 6.4 If δ is small, we can obtain an accuracy of ϵ in time

$$O\left(n^{\frac{\log \epsilon}{\log b\delta}} \log n\right)$$

Proof: If we require an accuracy of ϵ, we ensure $\frac{(b\delta)^k}{k!} < \epsilon$. This can be obtained if we ensure $(b\delta)^k = \epsilon$. Solving for k, we get

$$\begin{aligned} k \log b\delta &= \log \epsilon \\ k &= \frac{\log \epsilon}{\log b\delta} \end{aligned}$$

This requires $O(n^k \log n)$ time. Substituting the value for k, the theorem follows. □

Theorem 6.5 In the space of all systems, to compute the posterior probability of any proposition (of bounded size) given observation *obs*, we can guarantee an error of less than ϵ ($\epsilon < \frac{1}{2}$), at least $1 - \psi$ of the cases in time

$$O\left(n^{\frac{\log\epsilon\psi}{\log b\delta}} \log n\right)$$

Proof: We can assume that the $P(obs) \geq \psi$. This will be wrong less than ψ of the cases (by definition). If we make sure that $P_Q < \frac{\psi}{2}$, then

$$\frac{\psi}{2} < P(obs) - P_Q \leq P_W^{obs}$$

To achieve error ϵ, we make sure $P_Q < \epsilon\psi$. Then, as $\epsilon\psi < 2\epsilon P_W^{obs}$, we will have $P_Q < 2\epsilon P_W^{obs}$, which, as described above implies that the error will be less than ϵ. Thus we have to ensure that $P_Q < min(\frac{\psi}{2}, \epsilon\psi) = \epsilon\psi$.

By Theorem 6.4, this can be done in time

$$O\left(n^{\frac{\log\epsilon\psi}{\log b\delta}} \log n\right)$$

□

B A Prolog Implementation

This code implements a bottom-up depth-first depth-bounded search to find the most likely possible worlds that are consistent with the observations. Here we exploit the pattern-matching of Prolog, without utilizing the declarative nature or the search of the Language. All of the code here is deterministic; this program does not backtrack. This is done here by the use of the Prolog cut (!). N.B. we could also write a program that uses Prolog search (and no cuts) to do the depth-first search. This was not done as the following code is more efficient (by our tests), and can be more directly translated into committed-choice parallel logic programming languages or Lisp.

B.1 Network representation

The network is described using the relations:

nextvar$(I1, I2)$ means that variable $I2$ is the next variable after $I1$ in the total order of variables. The first (dummy) variable is *init*.

vals(I, V) is true if V is the list of values of variable I.

parents(I, A, F) is true if F is the set of values of parents of variable I, given the list A of values of the predecessors of I in reverse order (so the first element of A is the value of the variable before I).

prob(E, F, P) is true if proposition E given parents F has probability P. That is if $p(E|F) = P$. A proposition E is written as an equality *Variable=Value*.

inconsis_obs(E) is true if proposition E of the form *Variable=Value* is inconsistent with the observations.

miniscule(P) is true if P is below the threshold for searching in the depth-bounded search.

B.2 Representing possible worlds

A possible world is represented as $pw(P, Vs)$ which represents the possible world with values given by the list Vs and probability P. Vs is the list of the values of the variables in reverse order (so the first element of the list is the value of the last variable in the total ordering).

The found possible worlds that are consistent with the observations are represented as $wf(IM, FM, FND)$ where IM is the mass of the possible worlds pruned by inconsistency, FM is the sum of the probabilities of the possible worlds found (these are all consistent with the observations) and FND is a list of these worlds.

B.3 Search Procedure

The top level procedure is

$$chain(I, VIs, VAs, PAs, D0, D1)$$

where

I is the current variable;

VIs is the list remaining values of I to test;

VAs is the list of the values of the ancestors of I (in reverse order, i.e., the first element of the list is the value of the immediate predecessor of I);

$PAs = P(VAs)$;

$D0$ is the worlds found previously in the search;

$D1$ is the final worlds found.

```
chain(_,[],_,_,D,D).
chain(I,[V1|VIs],VAs,PAs,D0,D1) :-
   parents(I,VAs,Parents),
   prob(I=V1,Parents,PV1),
   P1 is PAs*PV1, !,
   test(I,V1,P1,VIs,VAs,PAs,D0,D1).
```

The procedure *test* is to test one of the values of the variable being tested, and then recurse to find the rest of the possible worlds.

$$test(I, V1, P1, VIs, VAs, PAs, D0, D1)$$

is true when

I is the current variable;

$V1$ is a possible value for variable I;

$P1$ is the probability I having value V and all other variables having values given by VAs;

VIs is the list remaining values of I to test (after $V1$);

VAs is the list of the values of the ancestors of I;

$PAs = P(VAs)$;

$D0$ is the worlds found previously;

$D1$ is the final worlds found.

```
test(I,V1,P1,VIs,VAs,PAs,wf(IM0,FM,FND),D2) :-
   inconsis_obs(I=V1), !,
   IM1 is IM0 + P1,
   chain(I,VIs,VAs,PAs,wf(IM1,FM,FND),D2).
test(I,_,P1,VIs,VAs,PAs,D0,D1) :-
   miniscule(P1), !,
   chain(I,VIs,VAs,PAs,D0,D1).
test(I,V1,P1,VIs,VAs,PAs,D0,D2) :-
   nextvar(I,I1), !,
   vals(I1,VI1), !,
   chain(I1,VI1,[V1|VAs],P1,D0,D1), !,
   chain(I,VIs,VAs,PAs,D1,D2).
test(I,V1,P1,VIs,VAs,PAs,wf(IM0,FM,FND),D2) :-
   FM1 is FM+P1, !,
   chain(I,VIs,VAs,PAs,
         wf(IM0,FM1,[pw(P1,[V1|VAs])|FND]),D2).
```

B.4 Information Seeking

make_worlds($Wlds$) searches and returns all the consistent worlds with probability above the threshold.

```
make_worlds(Wlds) :-
   nextvar(init,I0),
   vals(I0,VI0),
   chain(I0,VI0,[],1,wf(0,0,[]),Wlds).
```

B.5 Representation of a one thousand bit adder

The following represents a 1000 bit cascaded ripple adder. See Section B.1 for a description of the relations used. The input to the circuit is every bit is *on*. The observation is that every output bit, except bit 5, is *on* and output bit 5 is *off*.

```
nextvar(init,i1(1)).
nextvar(i1(N),i2(N)).
nextvar(i2(N),i3(N)).
nextvar(i3(N),x1ok(N)).
nextvar(x1ok(N),x1(N)).
nextvar(x1(N),x2ok(N)).
nextvar(x2ok(N),x2(N)).
nextvar(x2(N),a1ok(N)).
nextvar(a1ok(N),a1(N)).
nextvar(a1(N),a2ok(N)).
nextvar(a2ok(N),a2(N)).
nextvar(a2(N),o1ok(N)).
nextvar(o1ok(N),o1(N)).
nextvar(o1(N),i1(N1)) :-
   N < 1000,
   N1 is N+1.

vals(i1(_),[on,off]).
vals(i2(_),[on,off]).
vals(i3(_),[on,off]).
vals(x1ok(_),[ok,stuck1,stuck0,ab]).
vals(x1(_),[on,off]).
vals(x2ok(_),[ok,stuck1,stuck0,ab]).
vals(x2(_),[on,off]).
vals(a1ok(_),[ok,stuck1,stuck0,ab]).
vals(a1(_),[on,off]).
vals(a2ok(_),[ok,stuck1,stuck0,ab]).
vals(a2(_),[on,off]).
vals(o1ok(_),[ok,stuck1,stuck0,ab]).
vals(o1(_),[on,off]).

parents(i1(_),_,[]).
parents(i2(_),_,[]).
parents(i3(1),_,[]).
parents(i3(_),[_,_,Vo1|_],[Vo1]).
parents(x1ok(_),_,[]).
parents(x1(_),[X1ok,_,Vi2,Vi1|_],
```

```
                [X1ok,Vi2,Vi1]).
parents(x2ok(_),_,[]).
parents(x2(_),[X2ok,Vx1,_,Vi3|_],
                [X2ok,Vx1,Vi3]).
parents(a1ok(_),_,[]).
parents(a1(_),[A1ok,_,_,_,_,_,_,Vi2,Vi1|_],
                [A1ok,Vi2,Vi1]).
parents(a2ok(_),_,[]).
parents(a2(_),[A2ok,_,_,_,_,_,Vx1,_,Vi3|_],
                [A2ok,Vx1,Vi3]).
parents(o1ok(_),_,[]).
parents(o1(_),[O1ok,Va2,_,Va1|_],
                [O1ok,Va2,Va1]).

prob(i1(_)=on,_,0).
prob(i1(_)=off,_,1).
prob(i2(_)=on,_,0).
prob(i2(_)=off,_,1).
prob(i3(1)=on,_,0).
prob(i3(1)=off,_,1).
prob(i3(_)=V,[V],1).
prob(i3(_)=on,[off],0).
prob(i3(_)=off,[on],0).

prob(x1ok(_)=V,_,P) :-
   prob_ok(V,P).
prob_ok(ok,0.99999).
prob_ok(stuck1,0.0000049).
prob_ok(stuck0,0.0000049).
prob_ok(ab,0.0000002).

prob(x1(_)=V,Par,Prob) :-
   prob_xorgate(V,Par,Prob).
prob(x2ok(_)=V,_,P) :-
   prob_ok(V,P).
prob(x2(_)=V,Par,Prob) :-
   prob_xorgate(V,Par,Prob).
```

```
prob(a1ok(_)=V,_,P) :-
   prob_ok(V,P).
prob(a1(_)=V,Par,Prob) :-
   prob_andgate(V,Par,Prob).
prob(a2ok(_)=V,_,P) :-
   prob_ok(V,P).
prob(a2(_)=V,Par,Prob) :-
   prob_andgate(V,Par,Prob).
prob(o1ok(_)=V,_,P) :-
   prob_ok(V,P).
prob(o1(_)=V,Par,Prob) :-
   prob_orgate(V,Par,Prob).

prob_xorgate(on,[ok,on,on],0).
prob_xorgate(off,[ok,on,on],1).
prob_xorgate(on,[ok,on,off],1).
prob_xorgate(off,[ok,on,off],0).
prob_xorgate(on,[ok,off,on],1).
prob_xorgate(off,[ok,off,on],0).
prob_xorgate(on,[ok,off,off],0).
prob_xorgate(off,[ok,off,off],1).
prob_xorgate(on,[stuck1|_],1).
prob_xorgate(off,[stuck1|_],0).
prob_xorgate(on,[stuck0|_],0).
prob_xorgate(off,[stuck0|_],1).
prob_xorgate(on,[ab|_],0.5).
prob_xorgate(off,[ab|_],0.5).

prob_andgate(on,[ok,on,on],1).
prob_andgate(off,[ok,on,on],0).
prob_andgate(on,[ok,on,off],0).
prob_andgate(off,[ok,on,off],1).
prob_andgate(on,[ok,off,_],0).
prob_andgate(off,[ok,off,_],1).
prob_andgate(on,[stuck1|_],1).
prob_andgate(off,[stuck1|_],0).
prob_andgate(on,[stuck0|_],0).
```

```
prob_andgate(off,[stuck0|_],1).
prob_andgate(on,[ab|_],0.5).
prob_andgate(off,[ab|_],0.5).

prob_orgate(on,[ok,on,_],1).
prob_orgate(off,[ok,on,_],0).
prob_orgate(on,[ok,off,on],1).
prob_orgate(off,[ok,off,on],0).
prob_orgate(on,[ok,off,off],0).
prob_orgate(off,[ok,off,off],1).
prob_orgate(on,[stuck1|_],1).
prob_orgate(off,[stuck1|_],0).
prob_orgate(on,[stuck0|_],0).
prob_orgate(off,[stuck0|_],1).
prob_orgate(on,[ab|_],0.5).
prob_orgate(off,[ab|_],0.5).

inconsis_obs(x2(N)=on) :- N =\= 5.
inconsis_obs(x2(5)=off).

miniscule(P) :- P < 0.0000001 .
```

Acknowledgements

Thanks to Xianchang Wang and Nevin Zhang for valuable comments on this paper. This research was supported under NSERC grant OGPOO44121, and under Project B5 of the Institute for Robotics and Intelligent Systems.

References

[Chang and Lee, 1973] C-L Chang and R. C-T Lee. *Symbolic Logical and Mechanical Theorem Proving.* Computer Science and Applied Mathematics. Academic Press, New York, 1973.

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

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987.

[de Kleer, 1986] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, March 1986.

[de Kleer, 1991] J. de Kleer. Focusing on probable diagnoses. In *Proc. 9th National Conference on Artificial Intelligence*, pages 842–848, Anahiem, Cal., July 1991.

[Henrion, 1990] M. Henrion. An introduction to algorithms for inference in belief nets. In M. Henrion, et. al., editor, *Uncertainty in Artificial Intelligence 5*, pages 129–138. North Holland, 1990.

[Henrion, 1991] M. Henrion. Search-based methods to bound diagnostic probabilities in very large belief networks. In *Proc. Seventh Conf. on Uncertainty in Artificial Intelligence*, pages 142–150, Los Angeles, Cal., July 1991.

[Jensen *et al.*, 1990] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly*, 4:269–282, 1990.

[Korf, 1985] K. E. Korf. Depth-first iterative deepening: an optimal admissable tree search. *Artificial Intelligence*, 27(1):97–109, September 1985.

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

[McCarthy, 1986] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116, February 1986.

[Pearl, 1984] J. Pearl. *Heuristics.* Addison-Wesley, Reading, MA, 1984.

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

[Peng and Reggia, 1990] Y. Peng and J. A. Reggia. *Abductive Inference Models for Diagnostic Problem-Solving.* Symbolic Computation – AI Series. Springer-Verlag, New York, 1990.

[Poole, 1991] D. Poole. Representing Bayesian networks within probabilistic Horn abduction. In *Proc. Seventh Conf. on Uncertainty in Artificial Intelligence*, pages 271–278, Los Angeles, July 1991.

[Poole, 1992a] D. Poole. Logic programming, abduction and probability. In *International Conference on Fifth Generation Computer Systems (FGCS-92)*, Tokyo, June 1992.

[Poole, 1992b] D. Poole. Probabilistic Horn abduction and Bayesian networks. Technical Report 92-2, Department of Computer Science, University of British Columbia, January 1992.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, April 1987.

[Shimony and Charniak, 1990] S. E. Shimony and E. Charniak. A new algorithm for finding MAP assignments to belief networks. In *Proc. Sixth Conf. on Uncertainty in Artificial Intelligence*, pages 98–103, Cambridge, Mass., July 1990.