Probabilistic Inference in Multiply Connected Belief Networks Using Loop Cutsets

H. Jacques Suermondt and Gregory F. Cooper
Stanford University, Stanford, California

ABSTRACT

The method of conditioning permits probabilistic inference in multiply connected belief networks using an algorithm by Pearl. This method uses a select set of nodes, the loop cutset, to render the multiply connected network singly connected. We discuss the function of the nodes of the loop cutset and a condition that must be met by the nodes of the loop cutset. We show that the problem of finding a loop cutset that optimizes probabilistic inference using the method of conditioning is NP-hard. We present a heuristic algorithm for finding a small loop cutset in polynomial time, and we analyze the performance of this heuristic algorithm empirically.

KEYWORDS: ***artificial intelligence, Bayesian methods, expert systems, probabilistic reasoning, belief networks, multiply connected, cutsets, loops***

INTRODUCTION

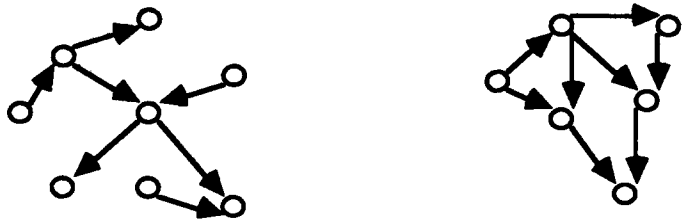
Bayesian belief networks provide an intuitive method for representing knowledge in a probabilistic framework. A belief network is an acyclic directed graph containing nodes that represent chance variables of quantities of interest. Each node assumes a set of values that are mutually exclusive and exhaustive. The arcs in a belief network represent the probabilistic relationships between nodes. These relationships can be causal, correlational, or a combination of the two. Other terms used for the belief-network representation are *probabilistic influence diagrams*, *causal networks*, and *Bayesian nets* (Cooper [1], Horvitz et al. [2], Howard and Matheson [3], Lemmer [4], Pearl [5], Rousseau [6], Spiegelhalter and Knill-Jones [7]).

Once a domain has been represented as a belief network, the probability of any node can be calculated conditioned on the values of any available evidence nodes. An *evidence node*, also called *instantiated node*, *fixed-value node*,

Address correspondence to H. J. Suermondt, M.S.O.B. X-215, Stanford University Medical Center, Stanford, CA 94305-5479.

International Journal of Approximate Reasoning 1990; 4:283–306

655 Avenue of the Americas, New York, NY 10010 0888-613X/90/$3.50



(a) (b)

Figure 1. Examples of belief networks. (a) A singly connected belief network. (b) A multiply connected belief network.

or *observed node*, is a belief-network node for which a single value has been established with certainty. We propagate the observed value of an evidence node throughout the belief network such that each proposition in the network is assigned a new measure of belief consistent with the axioms of probability theory. There are currently several algorithms for this process of probabilistic inference (Lemmer [4], Chavez and Cooper [8], Henrion [9], Kim and Pearl [10], Lauritzen and Spiegelhalter [11], Pearl [5, 12–14], Schachter [15, 16]).

Pearl's algorithm for updating probabilities in singly connected belief networks (SCBNs) is well known [5]. The main limitation of this algorithm is that the performance of belief updates in time that is linear in the size of the network is limited to singly connected belief networks. A singly connected belief network, also known as a *causal polytree*, has at most one path (in the undirected sense) from any node to any other node (see Fig. 1a) (Harary [17], p. 32). A multiply connected belief network, on the other hand, can have more than one path (in the undirected sense) between nodes (see Fig. 1b).

Unfortunately, many belief networks of practical use are multiply connected. Pearl presents several ways to apply the SCBN algorithm to such networks (Pearl [5, 12, 13]. One, the method of conditioning, or reasoning by assumptions, provides a reasonable solution provided that the network is not highly connected (Pearl [12]). The method of conditioning is based on selecting a set of nodes, the *loop cutset*, from the belief network and then considering separately all possible combinations of values that these nodes can have. In other words, we treat each possible combination of values of the nodes of the loop cutset as a separate case. Once the nodes of the loop cutset have been instantiated, the belief network can be treated as though it were singly connected for the purpose of inference. Thus, a loop cutset in the method of conditioning differs from the graph-theoretic concept of a cutset: Graph-theoretically, a cutset is a set of nodes or arcs the removal of which disconnects the graph (Shier and Whited [18]). Our notion of a loop cutset is a set of nodes whose instantiation makes a multiply connected belief network functionally singly connected.

Because each possible combination of values of the nodes of the loop cutset has to be considered separately in inference, we are interested in finding the

Figure 2. Blocking conditions. Because the blocking conditions are in effect, (a) a fixed-value node (shaded) no longer transmits information between parent and child nodes; also, (b) information passing between child nodes is blocked. (c) In nodes that have not been observed and that do not have any observed descendants, no information is passed between parent nodes.

minimal loop cutset, which we define to be the set of nodes satisfying the requirements of the method of conditioning such that the product of the number of values of these nodes is minimal. We shall show that the problem of finding a minimal loop cutset is NP-hard; however, it is possible in many belief networks to find rapidly a small set of nodes such that this set will be a minimal or near-minimal loop cutset. We describe a heuristic algorithm that will locate such a set of nodes, and we discuss the efficacy of this algorithm in finding a minimal loop cutset.

PEARL'S SCBN ALGORITHM

Pearl's SCBN algorithm performs probabilistic inference through a series of local probabilistic propagation operations, in which each node receives information messages from its neighboring nodes and combines these messages to update its measure of probability. Each node determines which of its neighbors need to receive updated information in order to maintain a correct probability distribution. Thus, through entirely local operations, the algorithm updates the probabilities for the nodes in the belief network to incorporate new evidence. See Pearl [5, 13] for a more detailed discussion.

There are situations in which the structure of the network is such that instantiation of one node will have no effect on the probability distribution of some other nodes in the network. To prevent unnecessary calculations for nodes whose belief distributions will not be affected anyway, we can set *blocking conditions* for transmission of information. These blocking conditions, based on the criterion of *d-separation* discussed by Pearl [6, 19] and Pearl and Verma [20], implement the idea that certain portions of the network are independent of other portions given that the values for certain nodes are fixed. The blocking conditions are the following (see also Fig. 2):

- A fixed-value node does not send information from its children to its parents or from its parents to its children.

- A fixed-value node does not send information from one child to any other children.
- A node whose value is not fixed and that does not have any fixed-value descendants does not send information from one parent to any other parents.

The last condition is an illustration of the property of belief networks of *independence except through arcs*. If a node has not been instantiated, then information from one parent will not convey any information about the probabilities of the other parents (if we assume that the only path between the parents is through their common child). For a fixed-value node, however, each parent node functions as a possible explanation for that node value. Therefore, information about the belief distribution of one parent will affect the distributions of the others. The following example will clarify this property.

Consider two possible causes for a severe headache: a subdural hematoma and a brain tumor. Normally, the probability of having a subdural hematoma will not influence the probability of having a brain tumor: The two probability distributions are marginally independent. However, if we observe that a person has a serious headache, the probability of each of these two causes increases. Now, if a CT scan shows that the patient has a subdural hematoma, this observation explains the headache; therefore, the probability of a brain tumor decreases again. Thus, only if the child node (headache) has been observed does one parent node (subdural hematoma) influence the probability distribution of another (brain tumor).

For singly connected networks, the structure of the belief network determines when nodes should stop sending information. Even if no blocking conditions are observed, the worst that can happen is that the new information will be transmitted to parts of the network where it will not change any probability distributions. The fact that a pathway between nodes is blocked means only that beyond that block no beliefs are changed by the new information, so the extra work of recalculating those beliefs is unnecessary. Because information is never sent back down an arc from which it arrived, belief propagation comes to a natural halt for singly connected networks. As we shall show in the following section, blocking conditions play a much more central role in belief updates for multiply connected networks.

MULTIPLY CONNECTED NETWORKS

As noted earlier, most belief networks created for practical purposes are not constrained to the singly connected structure. Superimposing such a constraint could make the structure of the problem inadequate. However, if we allow the network to be multiply connected, we can introduce loops. A loop, also known as an *elementary circuit* (Berge [21], p. 7), is defined in terms of

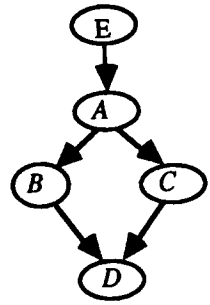


Figure 3. Example of a small belief network. We are interested in the probability of node D, given evidence *E*. Since node D has not been instantiated, it does not pass messages from B to C or vice versa. Therefore, cycling of information in loop A-B-C-D is not a problem in this example. However, messages passed from A through B to D are not independent of messages passed from A through C to D.

undirected paths. An undirected path between two nodes X_1 and X_k is defined as a sequence of nodes $[X_1, \ldots, X_k]$ such that in the belief network there exists an arc from X_i to X_{i+1} or from X_{i+1} to X_i for each $i \in \{1, \ldots, k-1\}$. A *loop* is an undirected path $[X_1, \ldots, X_k]$ in which (1) the initial node X_1 coincides with the terminal node X_k, and (2) apart from the coincidental initial and terminal nodes, every node in $[X_1, \ldots, X_k]$ is distinct. By definition, loops occur only in multiply connected networks. Loops are problematic for Pearl's SCBN algorithm, because the algorithm acts only through local operations on the nodes; when there are loops in the network, propagated information may cycle through the loops indefinitely.

In addition to possible cycling of information, multiply connected belief networks present another problem: Parents of a node may share information from elsewhere in the network; therefore, they may not independently influence the probability distribution of their common child. Due to the local nature of belief propagations in Pearl's SCBN algorithm, this phenomenon can lead to incorrect probability calculations unless an instantiated node prevents passage of shared information. An example will clarify this problem.

Consider the case given in Figure 3, where we are interested in the current probability of node D, given evidence node E, which is a predecessor of node A. Without loss of generality, we assume that the possible values of node A are a and $\neg a$; nodes B, C, and D are also binary-valued. Assume that nodes A–D have not been observed. Because node D has not been observed, it will fulfill the blocking conditions described earlier; therefore, messages will not go around the loop formed by nodes A–D but rather will stop propagation at node D. In this example we shall show that unless node A, B, or C is part of the loop cutset, the results of evidence propagation will not be correct according to the laws of probability, because the effects of evidence node E can reach node D through two paths.

If none of nodes A, B, or C are instantiated, we first use evidence E to calculate the new belief distribution for node A. Next, the evidence is propagated to nodes B and C. Nodes B and C send the information to node D. Node D now reads the messages from B and C and calculates its current belief.

Because of the local nature of belief propagation in Pearl's SCBN algorithm, the probabilities for any node are calculated by considering only the current beliefs for that node's neighbors, as well as the conditional probability tables relating the node to its neighbors. Thus, the probabilities for node D are calculated by the SCBN algorithm as follows:

$$\begin{aligned} P(d|E) = {} & P(d|b, c)P(b|E)P(c|E) \\ & + P(d|b, \neg c)P(b|E)P(\neg c|E) \\ & + P(d|\neg b, c)P(\neg b|E)P(c|E) \\ & + P(d|\neg b, \neg c)P(\neg b|E)P(\neg c|E) \end{aligned} \tag{1}$$

$P(b|E)$ and $P(c|E)$ are calculated locally as follows:

$$P(b|E) = P(b|a)P(a|E) + P(b|\neg a)P(\neg a|E)$$

$$P(c|E) = P(c|a)P(a|E) + P(c|\neg a)P(\neg a|E)$$

Analogously, we can calculate $P(\neg b|E)$ and $P(\neg c|E)$. The problem in calculating the probabilities for node D arises from substituting these probabilities into Eq. (1). If the network were singly connected, the resulting calculation would be equivalent to

$$P(d|E) = \sum_{A,B,C} P(d, A, B, C|E)$$

However, since node A is a common predecessor of both parents of node D, substituting the locally calculated values for $P(b|E)$, $P(\neg b|E)$, $P(c|E)$, and $P(\neg c|E)$ into Eq. (1) does not give us a calculation that is equivalent to this sum. When we collect terms, we obtain an equation that contains several terms that are inconsistent with the axioms of probability. An example of such an incorrect term is

$$P(d|b, c)P(b|a)P(a|E)P(c|\neg a)P(\neg a|E)$$

Note that this term contains the probabilities $P(a|E)$ and $P(\neg a|E)$, which are logically incompatible with each other. The product of these terms is inconsistent with the probability distributions specified by the belief-network representation. Due to the local nature of belief propagation, however, these terms are

unavoidable unless we instantiate node A, B, or C. If we fail to do this, we get cross-multiplications of incompatible terms.

We can prevent this situation by instantiating node A. We will then consider separately the case where node A has value a and that where it has value $\neg a$. The belief distribution for node D is calculated as a combination of these cases:

$$P(d|E) = P(d|a, E)P(a|E) + P(d|\neg a, E)P(\neg a|E) \tag{2}$$

If we use Pearl's SCBN algorithm, we calculate $P(d|a, E)$ as follows:

$$\begin{aligned} P(d|a, E) = {} & P(d|b, c)P(b|a, E)P(c|a, E) \\ & + P(d|b, \neg c)P(b|a, E)P(\neg c|a, E) \\ & + P(d|\neg b, c)P(\neg b|a, E)P(c|a, E) \\ & + P(d|\neg b, \neg c)P(\neg b|a, E)P(\neg c|a, E) \end{aligned} \tag{3}$$

where $P(d|b, c)$ is equal to $P(d|b, c, a)$ because of the blocking conditions; similarly, $P(d|b, \neg c)$, $P(d|\neg b, c)$, and $P(d|\neg b, \neg c)$ are equal to $P(d|b, \neg c, a)$, $P(d|\neg b, c, a)$, and $P(d|\neg b, \neg c, a)$, respectively. We can compute $P(d|\neg a, E)$ in a manner analogous to Eq. (3).

By instantiating node A, we are effectively treating it as an evidence node; therefore, due to the blocking conditions, $P(b|a, E)$ is equal to $P(b|a)$. Analogously, $P(c|a, E)$ equals $P(c|a)$, and so on. After substituting these simplifications into Eq. (3) (and its analogs), and substituting Eq. (3) (and its analogs) into Eq. (2), we have the following result:

$$\begin{aligned} P(d|E) = {} & P(d|b, c)P(b|a)P(c|a)P(a|E) \\ & + P(d|b, \neg c)P(b|a)P(\neg c|a)P(a|E) \\ & + P(d|\neg b, c)P(\neg b|a)P(c|a)P(a|E) \\ & + P(d|\neg b, \neg c)P(\neg b|a)P(\neg c|a)P(a|E) \\ & + P(d|b, c)P(b|\neg a)P(c|\neg a)P(\neg a|E) \\ & + P(d|b, \neg c)P(b|\neg a)P(\neg c|\neg a)P(\neg a|E) \\ & + P(d|\neg b, c)P(\neg b|\neg a)P(c|\neg a)P(\neg a|E) \\ & + P(d|\neg b, \neg c)P(\neg b|\neg a)P(\neg c|\neg a)P(\neg a|E) \end{aligned} \tag{4}$$

or, equivalently,

$$P(d|E) = \sum_{A,B,C} P(d, A, B, C|E)$$

Equation (4) is consistent with the axioms of probability theory. Thus, by instantiating node A to each of its possible values and combining the results, we can use Pearl's SCBN algorithm to calculate $P(d|E)$.

We can obtain the same result by instantiating either node B or node C, rather than node A, to each of its values. Instantiating one of these nodes prevents inconsistent products of conditional probabilities.

In summary, the instantiated nodes must not only stop infinite cycling of information, but also enable the local probability calculations to achieve probabilistically correct results. In general, to get belief calculations consistent with the axioms of probability when using Pearl's SCBN algorithm in multiply connected belief networks, we must satisfy the following *loop-cutset condition*:

> Instantiate at least one node from every loop in the belief network such that this node is a child to no more than one other node in the same loop.

If we do not instantiate at least one node from every loop in the network, we may fail to prevent cycling of information. To obtain consistent probability calculations, as described in this section, we also need to prevent the situation in which information can reach a node along multiple pathways; since only those instantiated nodes in a loop that are a child to no more than one other node in that loop will block an information pathway within the loop, we must instantiate at least one such node. Therefore, we need to satisfy the loop-cutset condition.

THE METHOD OF CONDITIONING

Pearl's SCBN algorithm to propagate messages by means of exclusively local operations on nodes applies only to singly connected networks; for multiply connected networks, there are several methods based on modification of this algorithm (Pearl [13]), all of which may suffer from combinatorial explosion because probabilistic inference using belief networks is known to be NP-hard (Cooper [22]). One such technique is the *method of conditioning* (Pearl [12, 13]).

The method of conditioning is based on instantiating a small set of nodes, the loop cutset, to "cut" all loops in a multiply connected belief network. The nodes of the loop cutset act as though they were evidence nodes and thus prevent cycling of information by way of the blocking conditions described earlier (see Fig. 4). By instantiating the nodes of the loop cutset, we satisfy the loop-cutset condition.

Thanks to the blocking conditions, the effects of new evidence can be calculated using Pearl's SCBN algorithm. The results of these calculations are then weighted by the joint probability of the nodes in the loop cutset, given the observed evidence. This joint probability of the loop-cutset nodes can be obtained during initialization of the network by sequentially instantiating these

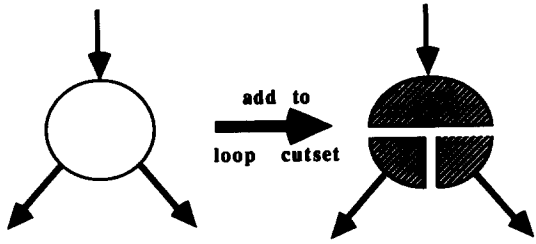


Figure 4. Function of the loop cutset. Loop-cutset nodes use the blocking conditions to separate ancestor nodes from descendant nodes, and descendant nodes from one another. Thus, loops in a multiply connected network are cut to leave a singly connected network. The node on the right is darkened to indicate that it has been instantiated.

nodes (Suermondt and Cooper [23]); when evidence is introduced, the joint probability is updated dynamically, as described by Pearl [12] and Suermondt and Cooper [23].

The correctness of the approach of conditioning is based on the following equation. Given evidence E and a loop cutset consisting of nodes $C_1, \ldots, C_n$, then, for any node X,

$$P(x|E) = \sum_{c_1,\ldots,c_n} P(x|E, c_1, \ldots, c_n) P(c_1, \ldots, c_n|E) \tag{5}$$

In the calculation for the new belief $P(x|E)$, the probability of x given a certain instantiation of the loop-cutset nodes, $P(x|E, c_1, \ldots, c_n)$, and the joint probability of that loop-cutset instantiation, $P(c_1, \ldots, c_n|E)$, can both be calculated by Pearl's SCBN algorithm [5].

A good way to look at belief updates using the method of conditioning is to act as though, rather than there being a single belief network, there is a collection of networks. The number of possible instantiations such that we cover every possible combination of value assignments to the members of the loop cutset determines the number of copies of the belief network. These copies, which we have thus far called instantiations, must all be processed when a new piece of evidence arrives.

In Figure 5, if we assume that our loop cutset consists of node A, then the copies of the network will be instantiation 1, in which we assume that node A has value a, and instantiation 2, in which we assume that node A is assigned $\neg a$. When we observe a new piece of evidence, the information in each network must be updated independently.

Intuitively, this method makes sense. When one is confronted with a problem that is too complex to handle, then an obvious strategy is to divide the problem into cases. We consider what would happen if we made certain simplifying assumptions; subsequently, we adjust the solution to take into account our original

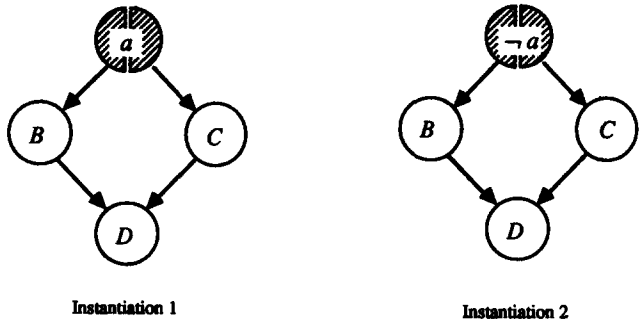


Figure 5. How multiple instantiations are considered. Node A has been assigned a different value in each instantiation, effectively stopping messages from B to A to C, and vice versa.

assumptions. Unfortunately, this method may require a significant amount of computation time. For example, if our loop cutset consists of just 10 binary nodes, then we need $2^{10} = 1024$ terms in the sum of Eq. (5). The time complexity of this approach is exponential in the number of nodes in the loop cutset. We can imagine highly connected belief networks for which this method would not be practical owing to the large size of the loop cutset. However, provided that we can find a small loop cutset for the multiply connected network, the method of conditioning provides a workable and intuitively clear solution.

The remainder of this article addresses the problem of finding a small loop cutset.

FINDING THE MINIMAL LOOP CUTSET

In general, probabilistic inference using belief networks is NP-hard (Cooper [22]). The method of conditioning is a generally applicable inference algorithm for belief networks; therefore, we can expect that in the worst cases inference using this method will be of exponential time complexity with respect to the number of nodes in the network. In particular, the time complexity of a probability update of a single node using the method of conditioning will be proportional to the product of the number of values of the loop-cutset nodes.

The exponential nature of the method of conditioning makes finding a small loop cutset important; if possible, this loop cutset should be minimal. As mentioned earlier, however, finding the minimal loop cutset is also an NP-hard problem. We can show that the minimal loop cutset (MLC) problem is NP-hard by reducing the minimal vertex cover (MVC) problem to the MLC problem. The MVC problem, which is known to be NP-hard (Garey and Johnson [24], p. 190), is defined as follows: Given a finite undirected graph (V, E) with vertices V and edges E, find a subset V' of V that is of smallest cardinality such that

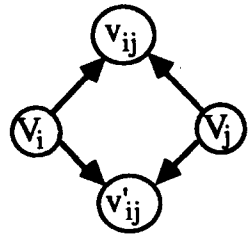


Figure 6. Loop L_{ij}. Each edge between nodes V_i and V_j in the minimal vertex cover problem corresponds to loop L_{ij} in the minimal loop cutset problem.

for each edge e in E, e has an endpoint in V'. Informally, the reduction from the MVC problem to the MLC problem is as follows. Each edge (V_i, V_j) in an instance of the MVC problem induces the structure shown in Figure 6 in the corresponding instance of the MLC problem.

In Figure 6, V_i, V_j, v_{ij}, v'_{ij} are propositional variables, and v_{ij} and v'_{ij} are instantiated to either T or F. The reduction thus constructs a loop L_{ij} in the MLC problem instance for each edge in the MVC problem instance. Each edge in the MVC problem instance induces four nodes in the MLC problem instance, each of which has no more than two parents. Thus, the reduction is polynomial in the size of the MVC problem instance. By the loop-cutset condition given earlier (see Multiply Connected Networks), for each loop L_{ij} in the MLC problem instance, we must include one of $\{V_i\}$, $\{V_j\}$, or $\{V_i, V_j\}$ in the loop cutset. Furthermore, by the loop-cutset condition, neither node v_{ij} nor node v'_{ij} can serve to cut the loop L_{ij}, so neither node would appear in a minimal loop cutset of the MLC problem instance. Selecting one of $\{V_i\}$, $\{V_j\}$, or $\{V_i, V_j\}$ as a cutset for loop L_{ij} in the MLC problem instance corresponds to selecting one of $\{V_i\}$, $\{V_j\}$, or $\{V_i, V_j\}$ to cover edge (V_i, V_j) in the MVC problem instance. Conversely, selecting one of $\{V_i\}$, $\{V_j\}$, or $\{V_i, V_j\}$ to cover edge (V_i, V_j) in the MVC problem instance corresponds to selecting one of $\{V_i\}$, $\{V_j\}$, or $\{V_i, V_j\}$ as a cutset for loop L_{ij} in the MLC problem instance. Thus, there is a one-to-one correspondence between finding the minimal loop cutset and finding the minimal vertex cover. Therefore, finding the minimal loop cutset is NP-hard.

Figure 7 shows an example. Here, $\{V_1, V_3\}$ is a minimal loop cutset in the MLC problem and a minimal vertex cover in the MVC problem. However, $\{V_1, V_2\}$ is neither because it leaves edge (V_3, V_4) in MVC uncovered and it leaves loop $L_{3,4}$ in MLC uncut.

Because of the computational complexity of the MLC problem, we have developed the heuristic algorithm described in the next section to find a loop cutset that is generally small but that is not guaranteed to be minimal. The worst-case time complexity for finding a loop cutset using this algorithm is $O(n^2)$ for a belief network of n nodes.

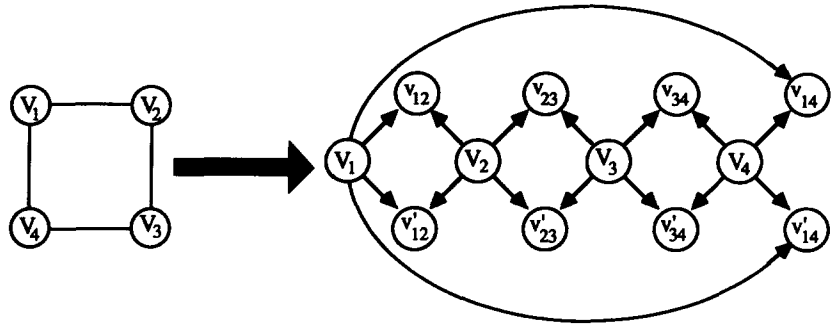


Figure 7. Reduction of a simple instance of the MVC problem in an undirected graph to an instance of the MLC problem in a belief network.

A HEURISTIC ALGORITHM FOR FINDING THE LOOP CUTSET

It is important to have as small a loop cutset as possible in order to minimize the number of possible instantiations of the loop-cutset nodes. The following algorithm creates a loop cutset that satisfies the loop-cutset condition. It is heuristic in the sense that it attempts to find a small loop cutset, but it does not guarantee that the minimal set will always be found. Its main steps are as follows:

1. Remove all parts of the network that are not in any loop.
2. If there are any nodes left, find a good loop-cutset candidate. A *good loop-cutset candidate* is defined as a node that satisfies the loop-cutset condition and the heuristic criteria described in the following subsections. Add this node to the loop cutset, and then remove it from the network.
3. If no nodes remain, halt; else, return to step 1.

A More Detailed Description of the Heuristic Algorithm

Step 1 is based on the fact that we want to add nodes to our loop cutset only if these nodes are part of at least one loop. No singly connected nodes of the belief network will be members of the loop cutset; therefore, all singly connected parts can be deleted. A *singly connected branch* of a belief network is defined as a subgraph S of the network such that (1) there is exactly one path between any two nodes of S and (2) there is at most one arc in the belief network such that this arc is shared by a node of S and any node of the belief network not in S. If a set of n nodes is a singly connected branch, there are exactly $n - 1$ arcs connecting the nodes (Harary [17], p. 33).

We start the process of removing singly connected branches by finding any nodes that have a single parent and no children or that have a single child and no parents; in other words, we find nodes that have only a single neighbor. These

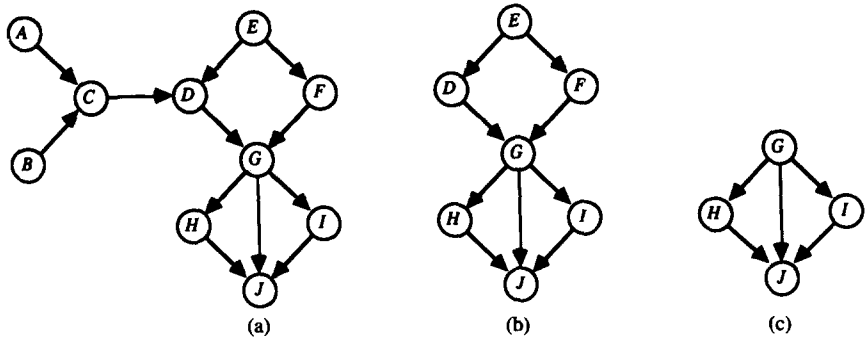


Figure 8. A belief network that illustrates the heuristic algorithm. (a) The original network; (b) the network after removal of all singly connected branches; (c) what is left of the network after a loop has been removed.

nodes are not part of any loop, so we can remove them from our network, because they do not need to be part of the loop cutset. We also remove the arcs that connect each pruned node to its single neighbor. After removing a node and its arc, we consider its neighbor. If this neighbor now also meets the condition for removal (that is, it has a single neighbor), then we can repeat the process. We continue until no nodes with a single neighbor remain in the network.

Let us illustrate how we remove the singly connected branches of the network in Figure 8a. Node A has only one neighbor; it is therefore not part of any loop, so we can remove it. We follow its arc to node C, which now has two neighbors, since node A has been deleted. If a node has more than one neighbor, we do not know whether it is part of a loop; for example, both nodes C and E have two neighbors; node C is not part of any loop, but node E is. As we cannot yet tell whether node C is a member of any loop, we leave this node; first, we see whether there is another node we can remove. In this case, node B also has only a single neighbor. Therefore, we delete node B and follow its arc, returning to node C. This time, node C has only one neighbor left, because nodes A and B have been deleted, so we now know that node C is not part of a loop and we can delete it. We follow its arc to node D. Node D has two neighbors, so we cannot continue. There are no other nodes with only a single neighbor; therefore, we have completed step 1. At this point, no nodes with fewer than two neighbors remain in the network. Since each singly connected branch of the network contains at least one node with fewer than two neighbors (Harary [17], p. 34), we conclude that all singly connected branches of the belief network have been removed; all nodes remaining in the network are therefore members of one or more loops.

The goal of step 2 is to find a node that satisfies the loop-cutset condition for as many loops as possible, in order to minimize the number of possible instan-

tiations of the loop-cutset nodes. Therefore, in step 2, we employ a heuristic strategy that has the following three components.

1. We consider only those nodes that have one or no parents. Thus, we avoid nodes that violate the loop-cutset condition by having more than one parent in the same loop.
2. Of the nodes that remain under consideration, we select the node that has the most neighbors. Since all nodes remaining in our network are members of one or more loops, the number of loops of which a node is a member will increase with the number of neighbors that that node has. By adding the node with the most neighbors, we hope to minimize the number of nodes that must be added to the loop cutset to satisfy the loop-cutset condition.
3. If there is a choice among multiple nodes that each have the same number of neighbors, then we select the node with the lowest number of possible values. This strategy will help us to minimize the number of possible value assignments to the loop-cutset nodes.

In the next section we shall discuss this heuristic in more detail.

After heuristically deciding which node is a good loop-cutset member, we add this node to our set and remove it from the network. By removing a node that is in a loop, we potentially make the remainder of that loop singly connected. Because we may thus create new singly connected branches in the network, we need to return to step 1 and remove these branches.

In Figure 8b, nodes G and J have more than one parent, so they cannot be considered for the loop cutset. The nodes remaining under consideration all have two neighbors and are therefore equally attractive candidates by the first two criteria. If, for example, node E has fewer possible values (e.g., two) than any other node (e.g., all others have three or more values), by the fewest-values criterion we decide to add it to our loop cutset. After removing node E, we can prune its neighbors—nodes D and F—because these nodes each have only one remaining neighbor. After these nodes have been removed, once more all remaining nodes have more than one neighbor, so all nodes are members of one or more loops. Therefore, we need to look for the best loop-cutset candidate again.

With what is left of the network (see Fig. 8c), we repeat the same process. Node G has the most neighbors of all nodes with one or no parents, so we select it next. After adding node G to the set and deleting it from our network, we follow its arcs to nodes H and I. These nodes now have only one neighbor left, so we can remove them. For either one of them, we follow the arc to node J, which has no remaining neighbors. We remove node J, and we are finished. Our final loop cutset consists of nodes E and G.

Alternative Heuristics for Selecting a Loop-Cutset Node

Selecting a good loop-cutset candidate in step 2 has a great influence on the eventual size of the loop cutset. In determining which node to make our

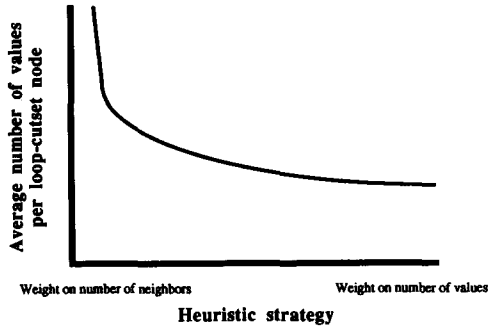


Figure 9. The effect of different heuristic criteria on the average number of values per loop-cutset node. In this graph, we characterize the expected effect of varying the heuristic strategy from considering only the number of neighbors to considering only the number of values of each candidate.

next candidate, we pay attention to two factors: (1) the number of values of the node, because the number of instantiations of the loop cutset grows by the product of the numbers of values of the loop-cutset nodes, and (2) the number of neighbors of the node. The latter factor is important because a node with many neighbors usually will be in more loops than will a node with few neighbors. By selecting a node that is a member of multiple loops, we may be able to cut all those loops with a single node, rather than having to cut every loop separately.

If we consider the spectrum of strategies that vary from considering only the number of neighbors to considering only the number of values of the loop-cutset candidate, we find that the two criteria have independent effects. In Figure 9, we characterize the effects of the two criteria on the average number of values of all the loop-cutset nodes. The curve in Figure 9 is based on our experience with combinations of the two heuristics in randomly generated networks, as described in the Evaluation section. If we fail to consider the number of values of each candidate in selecting the loop cutset, the average number of values per loop-cutset node grows quickly. The resulting large average number of values per loop-cutset node will have an adverse effect on the total number of instantiations of the loop-cutset nodes.

As shown in Figure 10, however, it also is necessary to consider the number of neighbors of the candidate in order to restrict the number of nodes in the loop cutset. If we consider only the number of values of each candidate, we get a very large loop cutset, albeit one with a low average number of values per loop-cutset member.

In Figure 11, we summarize the combined effect of the two strategies on the variable of interest, the total number of instantiations of the loop-cutset nodes. If we ignore the number of values of each loop-cutset candidate, we get a loop cutset that is suboptimal in that it has an unnecessarily high average number of values per loop-cutset node. On the other hand, if we focus on only the number

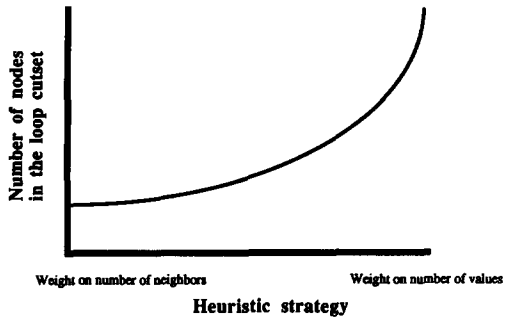


Figure 10. The effect of different heuristic criteria on the number of nodes in the loop cutset. If we consider exclusively the number of values of the nodes to be included in the loop cutset, the total number of loop-cutset members becomes very large.

of values of the loop-cutset candidates, ignoring the number of neighbors of the candidates, we also obtain a suboptimal loop cutset—it has too many nodes.

The optimal strategy is one that considers both the number of values and the number of neighbors of each loop-cutset candidate. Our experiments have shown that we obtain the best results when we consider foremost the number of neighbors of the loop-cutset candidate, breaking ties by selecting the node with the lower number of values (see under Evaluation). If, in some belief networks, there are large discrepancies between the numbers of values of cutset candidates, then we might consider giving the number of values priority over the neighbor-maximization criterion; we might also combine the two criteria into a single weight function. Both criteria attempt to minimize the number of instantiations of the loop-cutset nodes, but if the nodes are similar in number of values, the neighbor-maximization criterion is likely to be preferable.

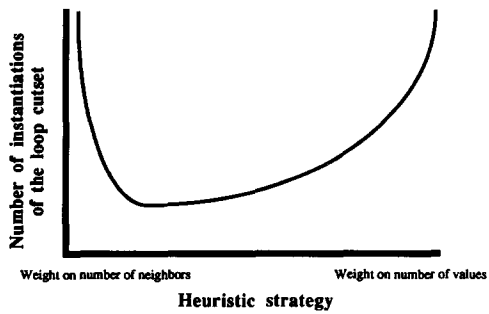


Figure 11. The effect of different heuristic criteria on the number of instantiations of the nodes in the loop cutset. In selecting the members of the loop cutset, we obtain the smallest number of loop-cutset instantiations if we consider both the number of neighbors and the number of values of each node under consideration.

Computational Time Complexity

The worst-case time complexity for finding a loop cutset using this algorithm is $O(n^2)$, where n is the number of nodes in the belief network. The algorithm can be described in terms of the following procedures. The first procedure is to visit all nodes in the network and select a node with fewer than two neighbors. Given that we know the number of neighbors (the *degree*) of each node, this procedure is $O(n)$ time. The second procedure is to determine the best loop-cutset candidate. We find the best loop-cutset candidate by visiting all nodes and rating the nodes on the basis of heuristic criteria. Since rating the nodes can be done in constant time given that we have available the degree of each node, this procedure also is $O(n)$. Finally, there is a procedure to remove a selected node. Given that we need to visit each neighbor of a removed node to update the degree of the neighbor, removal of a node also is $O(n)$.

We can summarize the algorithm as follows:

```
WHILE any nodes remain
  IF there is a node with fewer than two neighbors
    THEN select a node with fewer than two neighbors O(n)
  ELSE select a loop-cutset candidate O(n)
  Remove the selected node O(n)
END WHILE
```

Finding the heuristic loop cutset is done by an outer loop, which we repeat until no nodes remain in the network. During each cycle through the outer loop, we execute the first procedure, selecting a singly connected node of the network. If no node can be selected by the first procedure, no singly connected portions of the network remain; therefore, we execute the second procedure, selecting a loop-cutset candidate, as described in the preceding subsection. After selecting either a singly connected node or a loop-cutset candidate, we remove the selected node. Thus, during each cycle of the outer loop, exactly one node is removed from the network; therefore, the outer loop cannot be executed more than n times.

Thus, in the worst case, our algorithm repeats n times three procedures, each of $O(n)$. Therefore, the worst-case time complexity of our algorithm is $O(n^2)$.

EVALUATION

We have implemented the loop-cutset algorithm discussed in the preceding section for a general belief-network tool called KNET (Chavez and Cooper [25]). We have tested the heuristic algorithm on randomly generated belief networks, each of which we created by the following procedure. First, we specified the number of nodes and arcs that we wanted the network to have; this

gave us an indication of the connectivity of the network. For example, a network with 20 nodes and 19 arcs would be singly connected, but a network with 20 nodes and 25 arcs might contain six loops. After specifying the number of nodes and arcs, we generated a belief network of the desired number of nodes that was maximally connected; that is, there were as many arcs as there could be without the graph being cyclic. We created a maximally connected graph by numbering all nodes and creating arcs from each node to all higher-numbered nodes in the graph. After generating such a maximally connected graph, we eliminated arcs at random until the desired number of arcs was achieved; before eliminating any arc, we made sure that the belief network would remain connected. After completing these arc eliminations, we had a belief network of random topology given the specified number of nodes and arcs.

The principal bias of this method is that the method does not create graphs that are not connected. We introduced this bias because our purpose was to test our algorithm in networks resembling real belief networks, and, in our experience, these networks are generally connected. Because of this selection bias, our method of creating random belief networks does not guarantee that all members of the class of networks with the specified number of arcs and nodes are equally likely to be generated, since all arcs are not equally likely to be removed. However, the generated networks did provide us with an indication of the performance of our algorithm for a broad range of networks.

Using this mechanism, we generated a total of 600 random networks to use as testbeds for our heuristic algorithm to find the loop cutset. In the Introduction, we defined the minimal loop cutset as the set of nodes that satisfies the loop-cutset condition described under Multiply Connected Networks, such that the product of the numbers of values of these nodes is less than or equal to that of any other set of nodes also satisfying the loop-cutset condition. Because the task of finding the minimal loop cutset is NP-hard, we decided to limit our test cases to small networks (20–30 nodes, no more than 35 arcs per network). For these small networks, we could determine the minimal loop-cutset size by exhaustive search. For the test cases, we varied the number of nodes, the number of values per node, and the connectivity of the networks. Thus, we arrived at the following test cases:

I. One hundred networks containing 20 nodes, 25 arcs; each node is assigned randomly between two and six values.

II. One hundred networks containing 20 nodes, 25 arcs; each node is assigned randomly between two and ten values.

III. One hundred networks containing 20 nodes, 30 arcs; each node is assigned randomly between two and six values.

IV. One hundred networks containing 20 nodes, 30 arcs; each node is assigned randomly between two and ten values.

V. One hundred networks containing 30 nodes, 35 arcs; each node is assigned randomly between two and six values.

Table 1. Summary of Evaluation Data for Six Sets of Test Networks

	Set I	Set II	Set III	Set IV	Set V	Set VI
Number of networks	100	100	100	100	100	100
Number of nodes in each network	20	20	20	20	30	30
Number of arcs in each network	25	25	30	30	35	35
Possible number of values of each node	2–6	2–10	2–6	2–10	2–6	2–10
Percentage of networks in which minimal loop cutset was found	77	75	64	61	75	67
Average ratio of loop-cutset sizes (heuristic/minimal) in networks for which the minimal loop cutset was not found ($\overline{R_N}$)	1.72	2.38	2.06	2.72	1.96	2.29

VI. One hundred networks containing 30 nodes, 35 arcs; each node is assigned randomly between two and ten values.

The first two sets of networks are the smallest, with 20 nodes and 25 arcs each. The base case consists of set I. In set II we consider the change in performance as we increase the average cardinality of the nodes. The test networks of sets III and IV are more highly connected than are those of the first two sets, thus posing a more difficult task of finding a minimal loop cutset. The average cardinality of the nodes in set IV is greater than that in set III. By comparing test networks V and VI to sets I and II, we hope to get an indication of what happens when we increase the number of nodes of the networks. Set VI has a greater average cardinality than does set V, and the connectivity of the networks of sets V and VI is approximately the same as that of the networks of sets I and II.

Results of the Evaluation

For each of these sets of test networks, we compared the loop cutset generated by our heuristic algorithm with the minimal loop cutset generated by exhaustive search. Table 1 shows our results.

In approximately 70% of the networks studied, our heuristic algorithm found the minimal loop cutset. Let us call the percentage of networks in which our heuristic algorithm found the minimal loop cutset the *success rate S*. For set I, the heuristic algorithm located the minimal loop cutset in 77% of the cases (95% confidence interval: 68.8–85.2%). For set II, in which the average cardinality per node is increased, the heuristic algorithm found the minimal loop cutset in

75% of the cases (95% confidence interval: 66.5–83.5%). In set III, which has a much greater connectivity than do sets I and II, the heuristic algorithm found the minimal loop cutset in 64% of the cases (95% confidence interval: 54.6–73.4%). In set IV, where the average number of values per node is increased compared to set III, the success rate is 61% (95% confidence interval: 51.4–70.6%). In the remaining networks, which, compared to sets I and II, have the same connectivity but a greater number of nodes, the success rate was higher: For set V, the heuristic algorithm found the minimal loop cutset in 75% of the networks (95% confidence interval: 66.5–83.5%); in set VI, the success rate was 67% (95% confidence interval: 57.8–76.2%).

In the cases where the heuristically generated loop cutset was suboptimal, it is clear from the ratios of numbers of instantiations that the price of a suboptimal loop cutset is high: Having too many nodes can easily double the number of possible instantiations of the loop-cutset nodes. Let us define R as the ratio of the number of instantiations of the heuristically generated loop cutset to that of the minimal loop cutset; let $\overline{R_N}$ be the mean value of R for the networks in which a minimal loop cutset was not found. The values of $\overline{R_N}$ are shown in Table 1. For the networks of set I, the value of $\overline{R_N}$ is 1.72. For the networks of set II, the value of $\overline{R_N}$ is slightly higher: 2.375. Set III has an $\overline{R_N}$ of 2.06. For set IV, $\overline{R_N}$ is 2.72, the greatest value of all our sets of test networks. Set V has an $\overline{R_N}$ of 1.96, and for set VI, $\overline{R_N}$ is 2.29. In Figure 12, we display the range of R for each of our six sets of test networks.

When we compare sets of test networks that differ in only the average number of values of the nodes (set I versus set II; set III versus set IV; and set V versus set VI), we find that the difference in success rate S for each of these pairs of sets is not very great. For set I versus set II, $\chi^2 = 0.1028$; for set III versus set IV, $\chi^2 = 0.192$, and for set V versus set VI, $\chi^2 = 1.554$. None of these values are statistically significant at the 5% level. This finding lends support to the conclusion that the effect of the number of values of each node on the ability of the algorithm to locate the minimal loop cutset is relatively minor. However, we notice that for each of these pairs of sets the value of $\overline{R_N}$ is greater in the set with the higher average number of values per node, although this increase was not statistically significant for any of the pairs.

If we increase the connectivity of the network, locating a minimal loop cutset becomes progressively more difficult. As we see when we compare the test networks of set I to those of set III, the success rate of the heuristic algorithm decreases from 77% to 64% ($\chi^2 = 4.063$, $p < 0.05$). Similarly, when we compare set II to set IV, we see a decrease in S from 75% to 61% ($\chi^2 = 4.503$, $p < 0.05$). This decrease, which is statistically significant in both cases, leads us to believe that when we extrapolate to highly connected belief networks, the performance of the heuristic algorithm will not be as good as it is in sparsely connected networks. However, the method of conditioning, for which our heuristic algorithm finds a loop cutset, is not particularly well suited

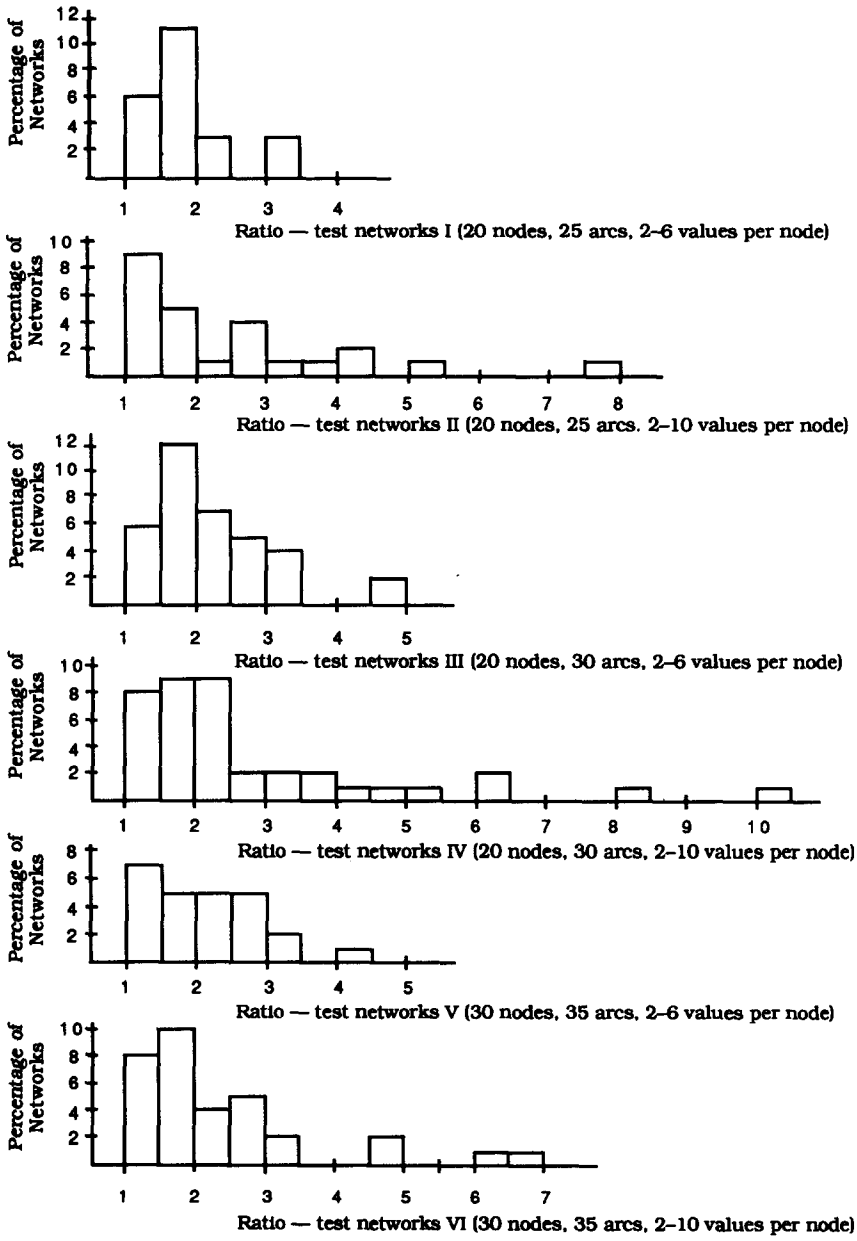


Figure 12. The ratio R of heuristic loop-cutset size to minimal loop-cutset size for the test networks for which our heuristic algorithm did not locate a minimal loop cutset.

for use in highly connected belief networks, since in such networks even the minimal loop-cutset size will be prohibitively large.

When we increase the number of nodes in the network while holding the connectivity of the network constant, we do not see a significant deterioration in the heuristic algorithm's performance. The networks of sets V and VI have a connectivity and average number of values similar to those of sets I and II, respectively. The networks of sets V and VI, however, have 30 nodes each, whereas those of sets I and II have 20 nodes each. When we compare the results for set V to those for set I, we see that there is only a minor decrease in success rate (from 77% to 75%, $\chi^2 = 0.110$, not significant). The success rate in set VI is 67%, a decrease from the success rate of 75% in set II ($\chi^2 = 1.55$, not significant). Also, the values of $\overline{R_N}$ are not significantly different in sets V and VI compared to sets I and II. Thus, these results suggest that increasing the number of nodes per network does not significantly affect the performance of the heuristic algorithm.

Discussion

The most noticeable effect of increasing the average number of values of the nodes in our test networks is that for those networks in which a minimal loop cutset is not found the ratio R becomes larger on average. When the average number of values per node increases, the effects of including unnecessary nodes in the loop cutset become more serious. In set IV, for example, there was one network in which the number of instances of the heuristically located loop cutset was 10 times greater than the size of the minimal loop cutset (see Fig. 12). The success rate, however, is not significantly affected by the average number of values of the nodes.

Increasing the number of nodes did not significantly affect the success rate or the value of $\overline{R_N}$ in our test cases. When we increased the connectivity of the networks, however, we saw a significant decrease in success rate. We determined that the principal reason for not finding a minimal loop cutset in these networks was that our algorithm does not consider nodes with multiple parents as loop-cutset candidates; therefore, when there is a structure of *adjacent loops*, as shown in Figure 13, the heuristic algorithm will often fail to find the optimal loop cutset. In Figure 13, the network consists of two adjacent loops, A-B-C-D and D-E-F-G. The minimal loop cutset, if we assume that all nodes are binary, would consist of node D. However, the heuristic algorithm will not consider node D as a candidate, because this node has two parents that are members of a loop. Thus, we will obtain a suboptimal loop cutset, for example, nodes A and E. Adjacent-loop structures such as this example accounted for all the failures in our test cases. Unfortunately, testing for such structures and correcting the loop cutset whenever an adjacent-loop structure arises makes the task of finding the loop cutset more complex.

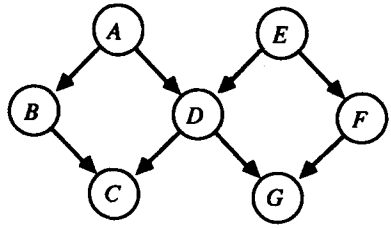


Figure 13. Adjacent-loop structure. This belief-network structure caused our heuristic algorithm to fail to find the minimal loop cutset in approximately 30% of our test cases.

Our preliminary results indicate that the heuristic loop-cutset algorithm that we have described performs well at finding a small loop cutset in polynomial time. Finding a small loop cutset is necessary if we want to use Pearl's method of conditioning. Therefore, the loop-cutset algorithm can assist in applying Pearl's method of conditioning to probabilistic inference for a select class of belief networks.

ACKNOWLEDGMENTS

We thank Lyn Dupre, Eric Horvitz, Ross Shachter, and Ramesh Patil for helpful comments on earlier versions of this paper. Support for this work was provided by the National Science Foundation under grant IRI-8703710, by the U.S. Army Research Office under grant P-25514-EL, and by the National Library of Medicine under grant LM-07033. Computer facilities were provided by the SUMEX-AIM resource under grant LM-05208 from the National Institutes of Health.

References

1. Cooper, G. F., NESTOR: a computer-based medical diagnostic aid that integrates causal and diagnostic knowledge, PhD Thesis, Stanford Univ., Stanford, Cal., 1984.
2. Horvitz, E. J., Breese, J. S., and Henrion, M., Decision theory in expert systems and artificial intelligence, *Int. J. Approximate Reasoning* 2, 247–302, 1988.
3. Howard, R. A., and Matheson, J. E., *Readings on the Principles and Applications of Decision Analysis*, Strategic Decisions Group, Menlo Park, Cal., 1984.
4. Lemmer, J. F., Generalized Bayesian updating of incompletely specified distributions, *Large Scale Syst.*, 5, 55–68, 1983.
5. Pearl, J., Fusion, propagation and structuring in belief networks, *AI* 29, 241–288, 1986.
6. Rousseau, W. F., A method for computing probabilities in complex situations, Tech. Report 6252-2, Center for Systems Research, Stanford Univ., Stanford, Cal., 1968.
7. Spiegelhalter, D. J., and Knill-Jones, R. P., Statistical and knowledge-based ap-

proaches to clinical decision-support systems, with an application in gastroenterology, *J. R. Stat. Soc. A* 147, 35–77, 1984.
8. Chavez, R. M., and Cooper, G. F., A fully polynomial randomized approximation scheme for the Bayesian inference problem, Report KSL-88-72, Knowledge Systems Laboratory, Stanford Univ., Stanford, Cal., 1988.
9. Henrion, M., Propagation of uncertainty by probabilistic logic sampling in Bayes' networks, in *Uncertainty in Artificial Intelligence 2* (J. F. Lemmer and L. N. Kanal, Eds.), Elsevier, New York, 149–164, 1988.
10. Kim, J. H., and Pearl, J., A computational model for causal and diagnostic reasoning in inference engines, *Proceedings of the 8th International Conference on AI*, Karlsruhe, West Germany, 190–193, 1983.
11. Lauritzen, S. L., and Spiegelhalter, D. J., Local computations with probabilities on graphical structures and their application to expert systems, *J. R. Stat. Soc. B* 50(2), 157–224, 1988.
12. Pearl, J., A constraint-propagation approach to probabilistic reasoning, in *Uncertainty in Artificial Intelligence* (L. N. Kanal and J. F. Lemmer, Eds.), Elsevier, New York, 357–369, 1986.
13. Pearl, J., Distributed revision of composite beliefs, *AI* 33, 173–215, 1987.
14. Pearl, J., Evidential reasoning using stochastic simulation of causal models, *AI* 32, 245–257, 1987.
15. Shachter, R. D., Evaluating influence diagrams, *Op. Res.* 34, 871–882, 1986.
16. Shachter, R. D., Probabilistic inference, *Op. Res.* 36, 589–604, 1988.
17. Harary, F., *Graph Theory*, Addison-Wesley, Menlo Park, Cal., 1972.
18. Shier, D. R., and Whited, D. E., Iterative algorithms for generating minimal cutsets in directed graphs, *Networks* 16, 133–147, 1986.
19. Pearl, J., *Probabilistic Reasoning in Expert Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, Cal., 1988.
20. Pearl, J., and Verma, T. S., The logic of representing dependencies by directed graphs, *Proceedings of the AAAI-87 Sixth National Conference on AI*, Seattle, Wash., 374–379, 1987.
21. Berge, C., *The Theory of Graphs and its Applications*, Wiley, New York, 1962.
22. Cooper, G. F., The computational complexity of probabilistic inference using belief networks, *AI* (in press).
23. Suermondt, H. J., and Cooper, G. F., Initialization for the method of conditioning in Bayesian belief networks, Report KSL-89-61, Knowledge Systems Laboratory, Stanford Univ., Stanford, Cal., 1989.
24. Garey, M. R., and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, Cal., 1979.
25. Chavez, R. M., and Cooper, G. F., KNET: integrating hypermedia and normative Bayesian modeling, *Proceedings of the 4th Workshop on Uncertainty in AI*, Minneapolis, Minn., 49–54, 1988.