

**Computer Science Department Technical Report
Cognitive Systems Laboratory
University of California
Los Angeles, CA 90024-1596**

d-SEPARATION: FROM THEOREMS TO ALGORITHMS

**Dan Geiger
Thomas Verma
Judea Pearl**

**June 1989
CSD-890040**

d-SEPARATION: FROM THEOREMS TO ALGORITHMS

Dan Geiger, Thomas Verma & Judea Pearl
Cognitive Systems Laboratory, Computer Science Department
University of California Los Angeles, CA 90024

ABSTRACT

An algorithm that identifies all independencies implied by the topology of a Bayesian network is developed. Its correctness and maximality stems from the soundness and completeness of *d*-separation with respect to probability theory.

1. INTRODUCTION

Bayesian networks encode properties of a probability distribution using directed acyclic graphs (dags). Their usage is spread among many disciplines such as: Artificial Intelligence [Pearl 1988], Decision Analysis [Howard and Matheson 1981; Shachter 1988], Economics [Wold 1964], Genetics [Wright 1934], Philosophy [Glymour et al. 1987] and Statistics [Lauritzen and Spiegelhalter 1988; Smith 1987]. A Bayesian network is a pair (D, P) where D is a dag and P is a probability distribution called the *underlying* distribution. Each node i in D corresponds to a variable X_i in P , a set of nodes I correspond to a set of variables \mathbf{X}_I and x_i, \mathbf{x}_I denotes values drawn from the domain of X_i and from the (cross product) domain of \mathbf{X}_I , respectively.⁽¹⁾ Each node in the network is regarded as a storage cell for the distribution $P(x_i | \mathbf{x}_{\pi(i)})$ where $\mathbf{X}_{\pi(i)}$ is a set of variables that correspond to the parent nodes $\pi(i)$ of i . The underlying distribution represented by a Bayesian network is composed via

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \mathbf{x}_{\pi(i)}), \quad (1)$$

(when i has no parents, then $\mathbf{X}_{\pi(i)} = \emptyset$). The role of a Bayesian network is to record a state of knowledge P , to provide means for updating the knowledge as new information is accumulated and to facilitate query answering mechanisms for knowledge retrieval [Lauritzen and Spiegelhalter 1988; Pearl 1988]. A standard query for a Bayesian network is to find the current belief distribution of a hypothesis variable X_I , given a composed evidence set $\mathbf{X}_J = \mathbf{x}_J$ i.e., to compute $P(x_i | \mathbf{x}_J)$ for each value of X_I and for a given combination of values of \mathbf{X}_J . The answer to such queries can, in principal, be computed directly from equation (1) because this equation defines a full probability distribution. However, it would be very inefficient both in time and space requirements, because it does not exploit independence relationships encoded in the network and because it treats the underlying distribution as a large table instead of a composition of several small ones. To better understand the improvements and limitations that more efficient algorithms can achieve, the following two problems must be examined: Given a variable X_k , a Bayesian network D and the task of computing $P(x_i | \mathbf{x}_J)$ determine, without resorting to numeric calculations: 1)

*This work was partially supported by the National Science Foundation Grant #IRI-8610155. "Graphoids: A Computer Representation for Dependencies and Relevance in Automated Reasoning (Computer Information Science)".

(1) Note that bolds letters denote sets of variables.

whether the answer to the query is sensitive to the value of X_k , and 2) whether the answer to the query is sensitive to the parameters $p_k = P(x_k | \mathbf{x}_{\pi(k)})$ stored at node k . The answer to both questions will be given in terms of conditional independence. The value of X_k does not affect this query if $P(x_i | \mathbf{x}_J) = P(x_i | \mathbf{x}_J, x_k)$ for all values of x_i, x_k and \mathbf{x}_J , or equivalently, if X_i and X_k are *conditionally independent* given \mathbf{X}_J , denoted by $I(X_i, X_k | \mathbf{X}_J)_P$. Similarly, whether the parameters p_k stored at node k would not affect the query $P(x_i | \mathbf{x}_J)$ will also be reduced to a simple test of conditional independence.

The main achievement of this paper is the development of an efficient algorithm that detects these independencies directly from the topology of the network, by merely examining the paths along which i, k and J are connected. The proposed algorithm is based on a graphical criteria, called *d-separation*, that associates the topology of the network to independencies encoded in the underlying distribution. The main property of *d-separation* is that it detects only genuine independencies of the underlying distribution and that it can not be sharpened to reveal more independencies [Verma and Pearl 1988; Geiger and Pearl 1988]. An additional achievement of the paper is the solution of two distinct problems, sensitivity to parameter values and sensitivity to variable instantiations, in a uniform approach.

2. SOUNDNESS AND COMPLETENESS OF *d*-SEPARATION

In this section we review the definition of *d-separation*; a graphical criteria that identifies conditional independencies in Bayesian networks. This criteria is both *sound* and *complete (maximal)* i.e., it identifies only independencies that hold in the underlying distribution and it cannot be sharpened to reveal more independencies. A preliminary definition is needed.

Definition: A *trail* in a dag is a sequence of links that form a path in the underlying undirected graph. A node β is called a *head-to-head node* with respect to a trail t if there are two consecutive links $\alpha \rightarrow \beta$ and $\gamma \rightarrow \beta$ on t . (note that nodes that start and end a trail t are not head-to-head nodes wrt to t).

Definition [Pearl 1988]: If J, K , and L are three disjoint subsets of nodes in a dag D , then L is said to *d-separate* J from K , denoted $I(J, K | L)_D$, iff there is no trail t between a node in J and a node in K along which every head-to-head node (wrt t) either is or has a descendent in L and every other node on t is outside L . A trail satisfying the conditions above is said to be *active*, otherwise it is said to be *blocked* (by L).

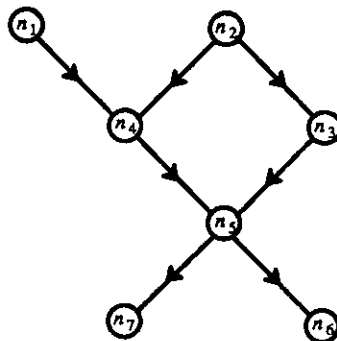


Figure 1

In Figure 1, for example, $J = \{n_4\}$ and $K = \{n_3\}$ are d -separated by $L = \{n_2\}$; the path $n_4 \leftarrow n_2 \rightarrow n_3$ is blocked by $n_2 \in L$ while the path $n_4 \rightarrow n_5 \leftarrow n_3$ is blocked because n_5 and all its descendants are outside L . Thus $I(n_4, n_2, n_3)_D$ holds in D . However, J and K are not d -separated by $L' = \{n_2, n_6\}$ because the path $n_4 \rightarrow n_5 \leftarrow n_3$ is rendered active: learning the value of the consequence n_6 , renders its causes n_3 and n_4 dependent, like opening a pathway along the converging arrows at n_5 . Consequently, $I(n_4, \{n_2, n_6\}, n_3)_D$ does not hold in D .

Note that by our definitions, to check whether L d -separates J and K requires an examination of all trails connecting a node in J and a node in K , including trails that form a cycle in the underlying undirected graph. For example, in Figure 1, $J = \{n_1\}$ and $K = \{n_7\}$ are not d -separated by $L = \{n_6\}$ because the trail along the nodes $n_1, n_4, n_5, n_3, n_2, n_4, n_5$ and n_7 is active; the only head-to-head node on this trail is n_4 and it belongs to L , while all other nodes are outside L . The next lemma shows that a trail that forms a (undirected) loop need not be examined because whenever there is an active trail with a loop there is an active *simple* trail as well, i.e. a trail that forms no cycles in the underlying undirected graph. In the previous example, the trail n_1, n_4, n_5 and n_7 is the simple active trail (by $\{n_6\}$), guaranteed by Lemma 1. The proof of lemma 1, which requires only the definition of d -separation, can be found in [Geiger et al. 1988].

Lemma 1: Let L be a set of nodes in a dag D , and let $\alpha, \beta \notin L$ be two additional nodes of D . Then α and β are connected via an active trail (by L) iff α and β are connected via a simple active trail (by L).

Definition: If $\mathbf{X}_J, \mathbf{X}_K$, and \mathbf{X}_L are three disjoint subsets of variables of a distribution P , then \mathbf{X}_J and \mathbf{X}_K are said to be conditionally independent given \mathbf{X}_L , denoted $I(\mathbf{X}_J, \mathbf{X}_L, \mathbf{X}_K)_P$ iff $P(\mathbf{x}_J, \mathbf{x}_K \mid \mathbf{x}_L) = P(\mathbf{x}_J \mid \mathbf{x}_L) \cdot P(\mathbf{x}_K \mid \mathbf{x}_L)$ for all possible values of $\mathbf{X}_J, \mathbf{X}_K$ and \mathbf{X}_L for which $P(\mathbf{x}_L) > 0$. $I(\mathbf{X}_J, \mathbf{X}_L, \mathbf{X}_K)_P$ is called a (*conditional independence*) *statement*.

The importance of d -separation stems from the following theorem [Verma and Pearl 1988; Geiger and Pearl 1988].

Theorem 2: Let $\mathbf{P}_D = \{P \mid (D, P) \text{ is a Bayesian network}\}$. Then,

$$I(J, L, K)_D \leftrightarrow I(\mathbf{X}_J, \mathbf{X}_L, \mathbf{X}_K)_P \text{ for all } P \in \mathbf{P}_D$$

The "only if" part (soundness) states that whenever $I(J, L, K)_D$ holds in D , it represents an independency of the underlying distribution and the "if" part (completeness) asserts that any independency that is not detected by d -separation can not be revealed without resorting to numerical parameters.

3. THE MAIN RESULTS

In this section we develop a linear time algorithm for identifying the set of nodes K that are d -separated from J by L . The soundness and completeness of d -separation guarantees that the set of variables X_K corresponding to the set of nodes K is the maximal set of variables that can be identified as being independent of X_J give X_L , without resorting to numerical calculations. The proposed algorithm is a variant of the well known Breath First Search algorithm; it finds all nodes reachable from J through an active trail (by L) hence, identifying the maximal set of nodes K satisfying $I(J, L, K)_D$. This task can be viewed as an instance of a more general task of finding a path in a directed graph for which some specified pairs of links are restricted not to appear consecutively. In this context, d -separation is viewed

as a specification for such restrictions, for example, two links $u \rightarrow v, v \rightarrow w$ cannot appear consecutively in an active trail unless $v \in L$. The following notations are employed: $D = (V, E)$ is a directed graph (not necessarily acyclic) where V is a set of nodes, $E \subseteq V \times V$ is the set of (directed) links and $F \subseteq E \times E$ is a list of pairs of adjacent links that cannot appear consecutively (F -connotes fail). We say that an ordered pair of links (e_1, e_2) is *legal* iff $(e_1, e_2) \notin F$, and that a path is *legal* iff every pair of adjacent links on it is legal. We emphasize that by "path" we mean a directed path, not a trail.

We propose a simple algorithm for the following problem: Given a finite directed graph $D = (V, E)$, a subset $F \subseteq E \times E$ and a set of nodes J , find all nodes reachable from J via a legal path in D . The algorithm and its proof are a slight modification of those found in Even [1979].

Algorithm 1

Input: A directed graph $D = (V, E)$, a set of legal pairs of links F and a set of nodes J .

Output: A labeling of the nodes such that a node is labeled with R iff it is reachable from J via a legal path.

- (i) Add a new node s to V and for each $j \in J$, add the link $s \rightarrow j$ to E and label it 1. Label s and all $j \in J$ with R (connoting Reachable). Label all other nodes and links with "undefined."
- (ii) $i := 1$
- (iii) Find all unlabeled links $v \rightarrow w$ adjacent to at least one link $u \rightarrow v$ labeled i , such that $(u \rightarrow v, v \rightarrow w)$ is a legal pair. If no such link exists, stop.
- (iv) Label each link $v \rightarrow w$ found in Step (iii) with $i+1$ and the corresponding node w with R .
- (v) $i := i+1$, Goto Step (iii).

The main difference between this algorithm and BFS, a change which has been proposed by Gafni [1988], is the traversal of the graph according to a labeling of the links and not according to a labeling of nodes (as in the traditional BFS algorithm). This change is essential as the next example shows (Figure 2); Let $F = \{(a, c)\}$, then the path from 1 to 3 through links a, b and c is legal while the path not traversing b is not legal because $(a, c) \in F$. However, BFS which labels nodes, would not traverse the link b since when link b is considered, its end point has already been labeled. Thus, BFS with node labeling, would not reveal the legal path connecting nodes 1 and 3.

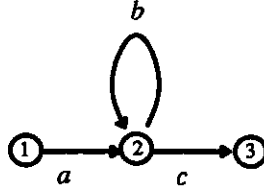


Figure 2

Lemma 3: Algorithm 1 labels with R all nodes that are reachable from s (and thus from J) via a legal path and only these nodes are labeled with R .

Proof: First, we show that if a node w_l is labeled with R , then there exists a legal path from s to w_l . Let $w_{l-1} \rightarrow w_l$ be a link through which w_l has been labeled. We induct on the label l of the link $w_{l-1} \rightarrow w_l$. If $l = 1$ then $w_l \in J$ and is therefore reachable from s . If $l > 1$, then by step (3), there exists a link $w_{l-2} \rightarrow w_{l-1}$ labeled with $l-1$ such that $(w_{l-2} \rightarrow w_{l-1}, w_{l-1} \rightarrow w_l)$ is a legal pair. Repeatedly applying this argument for $i = l \dots 2$ yields a legal path $w_0 \rightarrow w_1 \rightarrow \dots \rightarrow w_l$, where $w_0 \rightarrow w_1$ is labeled with 1. However, the only links labeled 1 emanate from s , hence the above path is the required legal path from s to w_l .

It remains to show that each node that is reachable from s via a legal path is labeled with R by the algorithm. Instead, we show that every link $\alpha \rightarrow v_m$ that is reachable from s via a legal path (i.e., it participates in a legal path emanating from s) is eventually labeled by the algorithm ⁽²⁾. The latter claim is stronger than the former because for every reachable node v_m there exists a reachable link $\alpha \rightarrow v_m$ and by Step (iv), whenever $\alpha \rightarrow v_m$ is labeled with some integer, v_m is labeled with R . We continue by contradiction. Let $l_m = v_{m-1} \rightarrow v_m$ be the closest link to s via a legal path that remains unlabeled. Let $p = s \rightarrow v_1 \rightarrow \dots \rightarrow v_{m-1} \rightarrow v_m$ be the path emanating from s and terminating with the link l_m . The portion of this path that reaches the link $l_{m-1} = v_{m-2} \rightarrow v_{m-1}$ is shorter than p . Thus, by the induction hypothesis, l_{m-1} is labeled by the algorithm. Hence, the link l_m is labeled as well (by the next application of step (iv)), contradicting our assumption that it remains unlabeled. \square

The complexity of the algorithm is $O(|E| \cdot |V|)$ because each link e is labeled at most once, and for each such link the algorithm requires at most $O(|V|)$ operations to find another link that is legal after e . This analysis assumes that for each link we keep the full list of all links that are legal after it, however, when the test for a legal pair requires a constant time (as it does for d -separation), the complexity of the algorithm reduces to $O(|E|)$.

Next we employ the above algorithm to solve the problem of identifying the set of nodes that are d -separated from J by L . For this aim, we construct a directed graph D' and define the set of legal pairs in D' such that a node is reachable from J via an **active trail** (by L) in D iff v is reachable from J via a **legal path** in D' . The following observations are the basis of our algorithm. First, any link on a trail can be traversed both ways. Therefore, to ensure that every active trail in D corresponds to a legal (directed)

(2) By labeled we mean that the algorithm attaches an integer label to that link and not an "undefined" label.

path in D' , D' must consist of all links of D in their forward and reverse direction. Second, an array that indicates for each node, whether it is in Z or has a descendent in Z , facilitates a test for legal pairs in D' that can be done in constant time, regardless of $|V|$.

Algorithm 2

Input: A Bayesian network $D = (V, E)$ and two disjoint sets of nodes J and L .

Data Structure: A list of incoming links (in-list) for each node $v \in V$.

Output: A set of nodes K where $K = \{\alpha \mid I(J, L, \alpha)_D\}$.

(i) Construct the following table:

$$\text{descendent}[v] = \begin{cases} \text{true} & \text{if } v \text{ is or has a descendent in } L \\ \text{false} & \text{otherwise} \end{cases}$$

(ii) Construct a directed graph $D' = (V, E')$ where $E' = E \cup \{(u \rightarrow v) \mid (v \rightarrow u) \in E\}$

(iii) Using algorithm 1, find the set of all nodes K' which have a legal path from J in D' , where a pair of links $(u \rightarrow v, v \rightarrow w)$ is legal iff $u \neq w$ and either 1) v is a head-to-head node on the trail $u \rightarrow v \rightarrow w$ in D and $\text{descendent}[v] = \text{true}$ or 2) v is not a head-to-head node on the trail $u \rightarrow v \rightarrow w$ in D and $v \notin Z$.

(iv) $K = V - (K' \cup J \cup L)$

Return (K) .

Lemma 4: For every node $\alpha \notin J \cup L$, α is reachable from J via a legal trail in D' iff there is an active path by L from J to α in D .

Proof: For $\alpha \notin J \cup L$ and $x_0 \in J$, if $(x_0 - x_1 \dots \alpha)$ is an active trail (by L) in D , then the directed path $(x_0 \rightarrow x_1 \rightarrow \dots \alpha)$ is a legal path in D' , and vice versa. (We have eliminated the case $\alpha \in J \cup L$ for technical convenience; the trail $(x_0 - x_1 \dots \alpha)$ is not active nor non-active because, by our definition, J, L , and $\{\alpha\}$ must be disjoint. \square)

Theorem 5: The set K returned by the algorithm is exactly $\{\alpha \mid I(J, L, \alpha)_D\}$.

Proof: The set K' constructed in Step (iii) contains all nodes reachable from J via a legal path in D' . Thus, by lemma 4, K' contains all nodes not in $J \cup L$ that are reachable from J via an active trail (by L) in D . However, $I(J, L, \alpha)_D$, holds iff $\alpha \notin J \cup L$ and α is not reachable from J (by an active path by L), therefore, $K = V - (K' \cup J \cup L)$ is exactly the set $\{\alpha \mid I(J, L, \alpha)_D\}$. \square

Next, we show that the complexity of the algorithm is $O(|E|)$ we analyze the algorithm step by step. The first step is implemented as follows: Initially mark all nodes of Z with true. Follow the incoming links of the nodes in Z to their parents and then to their parents and so on. This way, each link is examined at most once, hence the entire step requires $O(|E|)$ operations. The second step requires the construction of a list for each node that specifies all the links that emanate from v in D (out-list). The in-list and the out-list completely and explicitly specify the topology of D' . This step also requires $O(|E|)$ steps. Using the two lists the task of finding a legal pair in step (iii) of algorithm 2 requires only constant time; if $e_i = u \rightarrow v$ is labeled i then depending upon the direction of $u - v$ in D and whether v is or has a descendent in Z , either all links of the out-list of v , or all links of the in-list of v , or both are selected. Thus, one operation per each encountered link is performed. Hence, Step (iii) requires no more than $O(|E|)$ operation which is therefore the upper bound (assuming $|E| \geq |V|$) for the entire algorithm.

The above algorithm can also be employed to verify whether a specific statement $I(J, L, K)_D$ holds in a dag D . Simply find the set K_{\max} of all nodes that are d -separated from J given L and observe that $I(J, L, K)_D$ holds in D iff $K \subseteq K_{\max}$. In fact, for this task, algorithm 2 can slightly be improved by forcing termination once the condition $K \subseteq K_{\max}$ has been detected. Lauritzen et al [1988] have recently proposed another algorithm for the same task sub max. Their algorithm consists of the following steps. First, form a dag D' by removing from D all nodes which are not ancestors of any node in $J \cup K \cup L$ (and removing their incident links). Second, form an undirected graph G , called the *moral graph*, by shielding the directionality of the links of D' and connecting any two nodes that have a common child (in D') which is or has a descendent in L . Third, they show that $I(J, L, K)_D$ holds iff all (undirected) paths between J and K in G are blocked by L .

The complexity of the moral graph algorithm is $O(|V|^2)$ because the moral graph G may contain up to $|V|^2$ links. Hence, checking separation in G could require $O(|V|^2)$ steps. Thus, our algorithm is a moderate improvement as it only requires $O(|E|)$ steps. The gain is significant mainly in sparse graphs, where $|E| = O(|V|)$. We note that if the maximal number of parents of each node is bounded by a constant, then the two algorithms display the same asymptotic behavior i.e. linear in $|E|$. On the other hand, when the task is to find all nodes d -separated from J by L , and not merely validating a given independence, then a straight forward application of the moral graph algorithm requires $O(|V|^3)$ steps, because for each node not in $J \cup L$ the algorithm must construct a new moral graph, the construction of which requires $O(|V|^2)$ steps. Hence, for this task, our algorithm offers a considerable improvement.

The inference engine of Bayesian networks has also been used for decision analysis; an analyst consults an expert to elicit information about a decision problem, formulates the appropriate network and then by an automated sequence of graphical and probabilistic manipulations an optimal decision is obtained [Howard and Matheson 1981; Olmsted 1984; Shachter 1988]. When such a network is constructed it is important to determine the information needed to answer a given query $P(x_J | x_L)$ (where $\{j\} \cup L$ is an arbitrary set of nodes in the network), because some nodes might contain no relevant information to the decision problem and eliciting their numerical parameters is a waste of effort [Shachter 1988]. Assuming that each node X_i stores the conditional distribution $P(x_i | \pi(x_i))$, the task is to identify the set M of nodes that must be consulted in the process of computing $P(x_j | x_L)$ or, alternatively, the set of nodes that can be assigned arbitrary conditional distributions without affecting the quantity $P(x_j | x_L)$. The required set can be identified by the d -separation criterion. We represent the parameters of the distribution $P(x_i | \pi(x_i))$ as a dummy parent π_i of node i . From Theorem 1, all dummy nodes that are d -separated from J by L represent variables that are conditionally independent of J given L and so, the information stored in these nodes can be ignored. Thus, the information required to compute $P(x_j | x_L)$ resides in the set of dummy nodes which are not d -separated from J given L . Moreover, the completeness of d -

separation further implies that M is minimal; no node in M can be excluded on purely topological grounds (i.e., without considering the numerical values of the probabilities involved). The algorithm below summarizes the discussion.

Algorithm 3

- Input:** A Bayesian network, two sets of nodes J and L .
- Output:** A set of nodes M that contains sufficient information to compute $P(x_j | x_L)$
- (i) Construct a dag D' by augmenting D with a dummy node v' for every node v in D and adding a link $v' \rightarrow v$.
 - (ii) Use algorithm 2 to compute the set K' of nodes not d -separated from J by L .
 - (iii) Let M be the set of all dummy nodes v' that are included in K' .

We conclude with an example. Consider the network D of Figure 3 and a query $P(x_3)$.



Figure 3

The computation of $P(x_3)$ requires only to multiply the matrices $P(x_3 | x_1)$ and $P(x_1)$ and to sum over the values of X_1 . These two matrices are stored at the dummy nodes $1'$ and $3'$, which are the only dummy nodes not d -separated from node 3 (given \emptyset). Thus, algorithm 3 reveals the fact that the parameters represented by node $2'$ and $4'$ ($P(x_2), P(x_4 | x_1, x_2)$) are not needed for the computation of $P(x_3)$. Note, however, that knowing the value of X_4 might influence the computation of $P(x_3)$, because X_3 and X_4 could be dependent. The value of X_2 , on the other hand, never affects this computation because X_2 is independent of X_3 . This example shows that the question of whether a variable influences the computation of a query and the question of whether the parameters stored at that node influence the same query are two different questions. Algorithm 3 exploits the fact that parameters can be represented as (dummy) variables; it solves the latter problem by transforming it to an instance of the former.

Shachter was the first to address the problem of identifying irrelevant parameters [Shachter 1988]⁽³⁾. Our formulation provides several advantages. First, we distinguish between sensitivity to variable instantiations and sensitivity to parameter values, and the algorithm we provide can be tailored to solve either one of these problems. Shachter's algorithm handles the second problem and, therefore, it does not reveal all the independencies that are implied by the topology of the dag. For example, in Figure 3, Shachter's algorithm would correctly conclude that nodes 2 and 4 both contain no relevant information

(3) He also considers deterministic variables which we treat in [Geiger et al. 1989].

for the computation of $P(x_3)$. Yet, X_2 is independent of X_3 , while X_4 and X_3 might be dependent, a distinction not addressed in Shachter's algorithm. Second, our method is comprised of two components, 1) declarative characterization of the independencies encoded in the network (i.e., the d -separation criterion) and 2) procedural implementation of criterion defined in 1). This approach facilitates a clear proof of the validity and maximality of the graphical criterion, independent of the details of the algorithm, followed by proofs of the algorithm's correctness and optimality (it requires only $O(|E|)$ steps). In Shachter's treatment the characterization of the needed parameters is inseparable from the algorithm, hence, it is harder to establish proofs of correctness and maximality.

ACKNOWLEDGEMENT

We thank Eli Gafni for his help in developing algorithm 1, and to Azaria Paz and Ross Shachter for many stimulating discussions.

REFERENCES

- S. Even. 1979. *Graph Algorithms*, Computer Science Press.
- E. Gafni. Personal communication, 1988.
- D. Geiger & J. Pearl. August 1988. "On The Logic of Causal Models," *Proc. of the 4th Workshop on Uncertainty in AI*, St. Paul, Minnesota, pp. 136-147.
- D. Geiger, T. S. Verma and J. Pearl. 1989. "Identifying independence in Bayesian networks", Technical report R-116-I, UCLA Cognitive Systems Laboratory. In preparation.
- C. Glymour, R. Scheines, P. Spirtes and K. Kelly. 1987. *Discovering Causal Structure*, New York: Academic Press.
- R. A. Howard & J. E. Matheson. 1981. "Influence Diagrams," In, *Principles and Applications of Decision Analysis*, Menlo Park, CA: Strategic Decisions Group.
- S.L. Lauritzen, D. Spiegelhalter. 1989. "Local computations with probabilities on graphical structures and their applications to expert systems." *J. Royal Statist. Soc.*, ser. B.
- S.L. Lauritzen, A.P. Dawid, B.N. Larsen and H.G. Leimer. October 1988. "Independence Properties of Directed Markov Fields," Technical Report R 88-32, Aalborg Universitetscenter, Aalborg Denmark.
- S. M. Olmsted. 1983. "On Representing and Solving Decision Problems," Ph.D. Thesis, EES Dept., Stanford University.
- J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo, CA: Morgan Kaufmann.
- R.D. Shachter. 1988. "Probabilistic Inference and Influence Diagrams," *Operations Research*, Vol. 36, pp. 589-604.
- J.Q. Smith. June 1987. "Influence Diagrams for Statistical Modeling," Technical report #117, department of Statistics, University of Warwick, Coventry, England.
- T. Verma & J. Pearl. August 1988. "Causal Networks: Semantics and Expressiveness," *Proceedings of the 4th Workshop on Uncertainty in AI*, St. Paul, Minnesota, pp. 352-359.
- H. Wold. 1964. *Econometric Model Building*, Amsterdam: North-Holland Publishing Co.
- S. Wright. 1934. "The Method of Path Coefficients," *Ann. Math. Statist.*, Vol. 5, pp. 161-215.