A Heuristic Algorithm to Update Probabilities in Influence Diagrams

영향도에서의 확률개선을 위한 휴리스틱 알고리즘

Bae, Ki Woong**
배 기 웅

Abstract

The objective of this paper is to develop an algorithm to update probabilities in influence diagrams. Pearl has designed the method of conditioning. This paper applies his algorithm for probability updates by selecting a loop-cutset for the diagram and instantiating these loop-cutset nodes. It is discussed the conditions that need to be satisfied by the selected nodes. And a heuristic algorithm is presented for finding a loop-cutset that satisfies these conditions.

1. Introduction

Influence diagrams have proven to be a useful structure for representing knowledge in a probabilistic and decision analysis models. An influence diagram is an acyclic, directed graph consisting of decision nodes, chance nodes and value nodes. Some researchers distinguish probabilistic influence diagram with influence diagram whether it is composed of probabilistic nodes alone or not [5]. But in this paper, influence diagram is used in any case. In medical domains, for example, a chance node could represent the presence vs. absence of a disease, or the possible values of a laboratory result. The arcs in an influence diagram represent the probabilistic influence between nodes. If a disease generally cause a symptom, then this could be represented by an arc from the disease to the symptom. An arc implies that it is possible to characterize the relationship between the connected nodes by a (conditional) probability distribution.

Pearl suggests a well-known algorithm for updating probabilities in singly-connected belief network [4]. A singly-connected belief network, also known as a casual polytree, has only a single pathway (in the undirected sense) from any node to any other node. An influence diagram, on the other hand, can have more than one pathway between nodes [1, 5]. The main limitation of the pearl's algorithm is that its performance of belief updates in linear time is limited to singly-connected belief networks. We have applied his algorithm into updating the probabilities in the influence diagrams in this paper.

There are several ways to apply Pearl's algorithm to multiply-connected network such as an influence diagram [2, 3]. One, the method of conditioning provides a reasonable solution, provided that the network is not highly connected [3]. The method of conditioning is based on selection of a

* 본 연구는 1993년도 대전산업대학교 기성회 학술연구조성비 지원에 의해 이루어졌음
** 대전산업대학교 산업공학과

set of nodes, the loop-cutset, from the influence diagram and considering separately all possible combinations of values that these nodes can have. In other words, each possible combination of values of the nodes of the loop-cutset is treated as a separate case. In order to minimize the number of cases to consider, we are interested in finding the minimal loop-cutset: the set of nodes satisfying the requirements of the method of conditioning(described in Section 2) such that the product of the number of values of these nodes is minimal. The problem of finding the minimal loop-cutset is NP-hard; however, it is possible to find rapidly a small set of nodes such that, in many cases, it will be the minimal loop-cutset. In this paper, it is described a heuristic algorithm that will find a set of nodes that can be used for method of conditioning.

2. Notation

When we refer to a node as a variable, a capital letter(X) will be used; when we refer to a value of that node X, a lower case x will be used. For an arc from node A to node B, node A is referred as the *direct predecessor* of node B, and node B as the *direct successor* of node A. The direct predecessors of a node are all nodes from which there is a direct arc to that node; the set of *direct successors* is defined similarly. The term *direct predecessor* is not to be confused with *predecessor*. Node A is an *predecessor* of node B if there is a directed path form node A to node B. Similarly the term *successor* means the converse of *predecessor*: if node A is an predecessor of node B, then node B is a successor of node A. We define the set of *neighbors* of a node as the union of the set of *direct predecessors* of that node and the set of *direct successors* of the node. Initially, the values of each node in the influence diagram are unspecified. When we receive information about a node that allows us to determine the node's value, that node becomes an *evidence node*. The terms *evidence node* and *observed node* are used interchangeably. After observing a node, it is said that that node has been *instantiated*.

3. Belief Propagation Algorithm

When new evidence is observed for a proposition, this evidence is propagated throughout the influence diagram in such a way that each proposition in the diagram is assigned a new measure of belief consistent with the axioms of probability theory. Pearl's algorithm performs this task through a series of local belief propagation operations, in which each node receives information messages from its neighboring nodes and combines these messages to update its measure of belief. 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. More details can be found in [3] and [4].

An important advantage of Pearl's algorithm applied to influence diagram is that the effects of observations are propagated to all nodes, rather than to a single node that is the object of a query. As a medical example, consider an influence diagram with several nodes that represent disease and several nodes that represent symptoms, laboratory-test data, and other findings. Some belief-updating algorithms limit the transmission of information to a single query, such as one regarding the effect of observing a certain combination of symptoms on the probability distribution of a specific disease [4]. However, the physician will be rarely interested in the probability of a single disease; rather, he will wish to consider a list of diseases that make up the differential diagnosis. Pearl's algorithm transmits the effects of observation of a given set of symptoms to all

the disease nodes simultaneously, so queries regarding the effects of the same evidence on multiple propositions are efficient.

There are situations in which the structure of the network is such that observations of one node will have no effect on the belief distribution of some other nodes in the network. To prevent unnecessary calculations for the nodes whose belief distributions will be affected anyway, we can set the following blocking conditions for transmission of information:

(1) An evidence node does not send information from its direct successors to its direct predecessors or from its direct predecessor to its direct successors

(2) An evidence node does not send information from one direct successor to any other direct successors

(3) A node that has not been observed and that does not have any successors that have been observed does not send information from one direct predecessor to any other direct predecessors.

The last condition is a property of "independence except through links." If a node has not been observed, then all its direct predecessors will independently influence the probability distribution of that node, and information from one direct predecessor will not convey any information about the probabilities of the others; for an observed node, however, each direct predecessor node functions as a possible explanation for that observation. Therefore, information about the belief distribution (or probability distribution) of one direct predecessor will affect the distributions of the others.

For singly-connected networks, the structure determines when to stop sending information. Even if no blocking conditions are observed, the worst that can happen is that the algorithm will do unnecessary work. The fact that a pathway 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 is arrived, belief propagation comes to a natural halt for singly-connected network. As will be shown in Section 4.1, blocking condition plays a much more central role in belief updates for influence diagrams.

4. Probability Inference on Influence diagrams

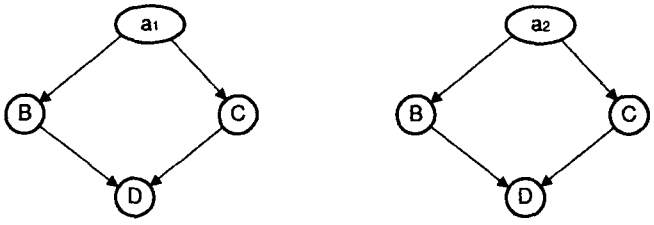
As noted earlier, most influence diagrams created for practical purposes cannot be constrained to the singly-connected structure. Superimposing such a constraint could make the structure of the problem unnatural and counterintuitive. However, the most efficient form of Pearl's algorithm only applies to singly-connected networks. The most generally applicable of these method to the multiply-connected structure is the method of conditioning.

4-1. The Method of Conditioning

The method of conditioning is based on instantiating a small set of nodes to "cut" all loops in an influence diagram. A loop is set of two undirected pathways between two nodes X and Y such that the pathways only intersect at X and Y. Although influence diagram does not allow loops or cycles, here we disregard the direction of an arc, then there may exist several loops in the diagram. It is obvious that loops never occur in the singly-connected networks. Because propagation of information is not centrally managed by Pearl's algorithm, when there are loops in a influence diagram, information may cycle infinitely. The method of conditioning prevents this cycling of messages by assuming that a select set of nodes, the loop-cutset, has been observed [3]. The nodes of the loop-cutset act as though they are evidence nodes, and thus prevent cycling of information by way the blocking conditions described in Section 3.

A good way to look at the probability updates using the method of conditioning is to act as though,

rather than a single influence diagram, there is a collection of diagrams. 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 influence diagram. These copies, which we have thus far called instantiations, all need to be processed when a new piece of information or evidence arrives.



(a) Instantiation 1 (b) Instantiation 2

Figure 1. Considering multiple instantiation

In Figure 1, if we assume that our loop-cutset consists of node A, then the copies of our network will be instantiation 1, in which we assume that node A has value a_1 and instantiation a_2, in which we assume that node A is assigned a_2. When we observe a new piece of evidence, the information in each diagram must be updated independently.

Thanks to the blocking conditions, the effects of new evidence can be calculated using Pearl's algorithm for singly-connected networks. The results of these calculations are then weighted by the joint probability of the nodes in the loop-cutset, given the observed evidence. The correctness of this approach is based on the rule of conditional probability: given evidence E and a loop-cutset consisting of nodes $C_1 \cdots C_n$, then for any node X,

$$P(x \mid E) = \sum_{c_1 \cdots c_n} P(x \mid E, c_1 \cdots c_n) P(c_1 \cdots c_n \mid E) \quad (1)$$

In the calculation for the new belief P(x|E), the probability of x given a certain instantiation of the loop-cutset nodes, $P(x|E, c_1 \cdots c_n)$, and the joint probability of that loop-cutset instantiation, $P(c_1 \cdots c_n|E)$, can both be calculated by Pearl's algorithm for singly-connected network [4].

Intuitively, this method makes sense. When one is confronted with a problem that is too complex to handle, then a obvious strategy is to devide the problem into cases. We consider what would happen if we made certain simplifying assumption; subsequently, we adjust the solution to take into account our original assumptions. Unfortunately, this method may require a significant amount of computation time. For example, if our loop-cutset consists of just ten nodes and each node has only two possible probabilities, then we need 2^{10}=1024 terms in the above sum. The time complexity of this approach is exponential in the number of nodes in the loop-cutset. We can imagine highly connected influence diagrams, for which this method would not be practical due to the large size of the loop-cutset. However, provided that we can find a small loop-cutset for the influence diagram, the method of conditioning provides a workable and intuitively clear solution for influence diagrams with few loops.

4-2. Additional Problems in Influence Diagrams

In addition to possible cycling of information, influence diagrams present another problem: direct

predecessors of a node may share information from elsewhere in the diagram; therefore, they may not independently influence the probability distribution of their common direct successor. Due to the local nature of belief propagations in Pearl's algorithm, this can lead to incorrect probability calculations unless the information shared by two direct predecessors is intercepted by a member of the loop-cutset. An example will clarify this problem.

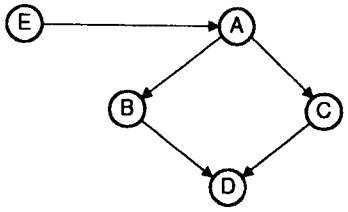


Figure 2. Example of a simple influence diagram

Consider the case given in Figure 2, where we are interested in the current probability of node D, given some information E that arrives at node A. We assume that nodes A through D are binary; for some node X, the possible values will be x_1 and x_2, Assume nodes A through D have not been observed. Because node D has not been observed, it will fulfill the blocking conditions; therefore, messages will not go around the loop formed by node A through D, but rather, will stop propagation at node D. However, this example will show that unless node A, B or C is part of the loop-cutset, the results of top-down evidence propagation will not be correct according to the laws of probability, because evidence from node E can reach node D through multiple pathways.

If none of nodes A, B and C are in the loop-cutset, when some new evidence E is observed, we first use E to calculate the new probability distribution for node A. Next, the evidence is propagated to node B and C. Node B and node C send the information to node D. Node D now reads the messages from B and C and calculates its current probability.

Because of the local nature of belief propagation in Pearl's algorithm, the probabilities for node D are calculated as follows;

$$\begin{aligned} P(d_1|E) = \; & P(d_1|b_1, c_1)P(b_1|E)P(c_1|E) \\ & + P(d_1|b_1, c_2)P(b_1|E)P(c_2|E) \\ & + P(d_1|b_2, c_1)P(b_2|E)P(c_1|E) \\ & + P(d_1|b_2, c_2)P(b_2|E)P(c_2|E) \end{aligned} \qquad (2)$$

$P(b_1|E)$ and $P(c_1|E)$ are calculated locally, however, as follows:

$$P(b_1|E) = P(b_1|a_1)P(a_1|E)+P(b_1|a_2)P(a_2|E)$$
$$P(c_1|E) = P(c_1|a_1)P(a_1|E)+P(c_1|a_2)P(a_2|E)$$

Analogously, we can calculate $P(b_2|E)$ and $P(c_2|E)$. After substituting these into Equation (2), and collecting terms, we get an equation that contains several terms that would not be part of the equation if we were to follow the axioms of probability. An example of such an incorrect term is

$$P(d_1|b_1,c_1)P(b_1|a_1)P(a_1|E)P(c_1|a_2)P(a_2|E)$$

Note that this term contains the probabilities $P(a_1|E)$ and $P(a_2|E)$ which are logically inconsistent with one another. Due to the local nature of probability propagation, however, these terms are

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

On the other hand, if we make node A part of our loop-cutset, then we will consider separately the case where node A has value a_1 and the case where node A has value a_2. The probability distribution for node D would be calculated as a combination of these cases:

$$P(d_1|E) = P(d_1|a_1,E)P(a_1|E)+P(d_1|a_2,E)P(a_2,E) \quad (3)$$

In this equation, $P(d_1|a_1,E)$ and $P(d_1|a_2,E)$ are the calculated probabilities for each instantiation of the loop-cutset; $P(a_1|E)$ and $P(a_2|E)$ are the cutset weights for these instantiation. To simulate this calculation using Pearl's algorithm, we calculate $P(d_1|a_1,E)$ and $P(d_1|a_2,E)$ as follows:

$$\begin{aligned} P(d_1|a_1,E) = {} & P(d_1|b_1, c_1)P(b_1|a_1,E)P(c_1|a_1,E) \\ & + P(d_1|b_1, c_2)P(b_1|a_1,E)P(c_2|a_1,E) \\ & + P(d_1|b_2, c_1)P(b_2|a_1,E)P(c_1|a_1,E) \\ & + P(d_1|b_2, c_2)P(b_2|a_1,E)P(c_2|a_1,E) \end{aligned} \quad (4)$$

We can calculate $P(d_1|a_2,E)$ in an analogous manner. We assume that node A, as a number of the loop-cutset, is an evidence node; therefore, due to the blocking conditions, $P(b_1|a_1,E) = P(b_1|a_1)$. Analogously, $P(c_1|a_1,E) = P(c_1|a_1)$, etc. After substituting these simplifications into Equation(4) (and its analogs), and substituting Equation(4) (and its analogs) into Equation(3), we have the following results:

$$\begin{aligned} P(d_1|E) = {} & P(d_1|b_1, c_1)P(b_1|a_1)P(c_1|a_1)P(a_1|E) \\ & + P(d_1|b_1, c_1)P(b_1|a_2)P(c_1|a_2)P(a_2|E) \\ & + P(d_1|b_1, c_2)P(b_1|a_1)P(c_2|a_1)P(a_1|E) \\ & + P(d_1|b_1, c_2)P(b_1|a_2)P(c_2|a_2)P(a_2|E) \\ & + P(d_1|b_2, c_1)P(b_2|a_1)P(c_1|a_1)P(a_1|E) \\ & + P(d_1|b_2, c_1)P(b_2|a_2)P(c_1|a_2)P(a_2|E) \\ & + P(d_1|b_2, c_2)P(b_2|a_1)P(c_2|a_1)P(a_1|E) \\ & + P(d_1|b_2, c_2)P(b_2|a_2)P(c_2|a_2)P(a_2|E) \end{aligned} \quad (5)$$

Equation (5) is the results of calculating the probability for D given E according to Pearl's algorithm provided we add node A to our loop-cutset.. Equation (5) is consistent with the axioms of probability theory. We reach the same result if we use node B or C as our cutset node, instead of node A.

4-3. A Condition for the Loop-Cutset Nodes

The previous example shows that nodes of the loop-cutset must not only stop infinite cycling of information, but also enable the local probability calculations to achieve correct results. We thus conclude that when we adjust Pearl's algorithm for influence diagrams, in order to get probability calculations consistent with the axioms of probability, the loop-cutset must satisfy the following Loop-Cutset Condition:

The loop-cutset must contain at least one node from every loop in the influence diagram such that this node is a direct successor to no more than one other node in the same loop.

If we do not add at least one node from every loop in the diagram to our loop-cutset, we may

fail to prevent cycling of information. If the only loop-cutset node in a certain loop is a direct successor to more than one other node in that loop, then it receives top-down information more than once, leading to the incorrect probability updates demonstrated in Section 4.2. The algorithm described in Section 5 finds a loop-cutset that satisfies these conditions.

5. An 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 instantiation 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:

Step 1. Remove all parts of the network that are not in any loop.
Step 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 below. Add this node to the loop-cutset, then remove it from the network; return to step 1. Terminate when no nodes remain in the diagram.
Step 1 is based on the fact that we only want to add nodes to our loop-cutset if those nodes are part of at least one loop. No singly-connected nodes of the influence diagram will be members of the loop-cutset; therefor, all singly-connected parts can be deleted. We start this process by finding any nodes that have a single direct predecessor and no direct successors, or single direct successor and no direct predecessors; in other words, we find nodes that have only a single neighbor. After removing each node and its arc, we consider its neighbor. If this neighbor now also meet the condition for removal (i, e. , 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 this for the network in Figure 3(A). 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 node C and node E have two neighbors; node C is not part of any loop, but node E is. Therefore, we leave node C ; 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 node A and node B have been deleted, so now we know 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, all singly-connected parts of the influence diagram have been removed; all nodes remaining in the network are members of one or more loops.

The goal of step 2 is to find a node that satisfies the loop-cutset condition(see Section 4.3) for as many loops as possible in order to minimize the number of possible instantiations of the loop-cutset nodes. Therefore, in step 2 we employ a heuristic strategy that has the following three components. First, we only consider nodes with one or fewer direct predecessors. Thus, we avoid nodes that violate the loop-cutset condition by having more than one direct predecessor in the same loop. Second, of the nodes that remain under consideration we select the node that has the most neighbors. Since all nodes remaining in our diagram are members of one or more loops, the number of loops that a node is part of 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 need to

be added to the loop-cutset to satisfy the loop-cutset condition. Third, if there is a choice between multiple nodes that each have the same maximum number of neighbors, then we select the node with the fewest values. This will minimize the number of possible value assignments to the loop-cutset nodes. If there are large discrepancies between the number of values of cutset candidates, then one might consider giving this 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.

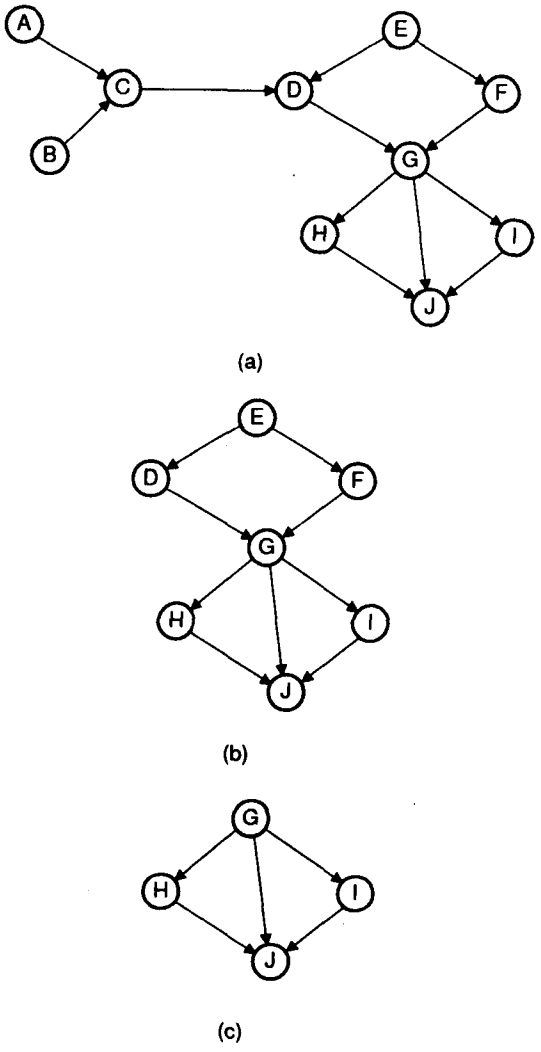


Figure 3. Influence diagram to illustrate the heuristic algorithm

After heuristically deciding which node would be a good loop-cutset member, we add this node to our set and remove it from the diagram. 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 parts of the diagram, we need to return to step 1 and remove these parts.
In Figure 3(B), nodes G and J have more than one direct predecessors 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 criteria. If, for example, node E has fewer values than any other node, by the fewest-values criterion we decide to add it to our loop-cutset. After removing node E, we can prune its neighbors, node D and node, since these nodes both have only one remaining neighbor. After removing these nodes, 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 diagram(Figure 3(C)), we repeat the same process. It is now clear that node G has the most neighbors of all nodes with one or fewer direct predecessors, so we select it next. After adding node G to the set and deleting it from our diagram, 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 are finished. Our final loop-cutset consists of node E and G.

6. Conclusions and Discussions

It is known that general probabilistic inference using influence diagrams is NP-hard [5, 6]. Since the method of conditioning is a generally applicable inference algorithm for influence diagrams, it is to be expected that inference using this method is exponential time complexity with respect to the number of nodes in the network. In particular, the average time complexity of a single probability update using the method of conditioning is proportional to the product of the number of values of the nodes in the loop-cutset.

The exponential nature of the method of conditioning makes it important that we find a small loop-cutset; if possible, we would like this loop-cutset to be minimal. As mentioned earlier, however, finding the minimal loop-cutset is also an NP-hard problem.

Because of the computational complexity of the minimal loop cutset problem, we have developed the heuristic algorithm described in Section 5 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 $O(n^2)$. Our preliminary results indicate that our algorithm performs well at finding a small loop-cutset which, in turn, allows us to apply Pearl's method of conditioning to probabilistic inference for a select class of influence diagrams.

References

[1] Olmsted, S.M.. *On representing and solving decision problems*, Ph.D. Thesis, Dep. of EES, Stanford University, Stanford, CA., 1883

[2] Pearl, J., *Fusion, propagation and structuring in belief networks*, Artificial Intelligence, Vol.29, pp241-288, 1986.

[3] Pearl, J., *Distributed revision of composite beliefs*, Artificial Intelligence, Vol.33, pp.173-215, 1987.

[4] Pearl, J., *A constraint-propagation approach to probabilistic reasoning* in Uncertainty in Artificial Intelligence, edited by Kanal, L.N. and lemmers, J.F., Elsevier Science Publishers, pp.357-369, 1986.

[5] Shachter, R.D., *Probabilistic Inference and Influence Diagrams*, Operational Research, VOL. 36, pp.589-604, 1988.

[6] Cooper, G.F., *The computational complexity of probabilistic inference using belief networks*, Memo KSL-87-27, Knowledge Systems Laboratory, Stanford University, Stanford, CA, 1988.