AN ABSTRACT OF THE THESIS OF

Zhaoyu Li for the degree of Master of Science in Computer Science presented on November 16, 1990.

Title: Complexity of Probabilistic Inference in Belief Nets - An Experimental Study

Abstract approved: Redacted for Privacy

Bruce D. D'Ambrosio

There are three families of exact methods used for probabilistic inference in belief nets. It is necessary to compare them and analyze the advantages and the disadvantages of each algorithm, and know the time cost of making inferences in a given belief network. This paper discusses the factors that influence the computation time of each algorithm, presents the predictive model of the time complexity for each algorithm and shows the statistical results of testing the algorithms with randomly generated belief networks.

Complexity of Probabilistic Inference in Belief Nets
-- An Experimental Study

by

Zhaoyu Li

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Completed November 16, 1990

Commencement June 1991

APPROVED:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented November 16, 1990

Typed by Zhaoyu Li for Zhaoyu Li

ACKNOWLEDGEMENT

The implementation of Conditioning , Clustering and Reduction , called 'IDEAL' system, is by Jack Breese and Sampath Srinivas at Rockwell. The implementation of SPI is by Bruce D'Ambrosio at Oregon State University. I thank Bruce D'Ambrosio who gave me great help and direction in my work. I also thank Walter Rudd, Thomas Dietterich and Prasad Tadepalli for their help.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Pages

COMPLEXITY OF PROBABILISTIC INFERENCE IN BELIEF NETS -- AN EXPERIMENTAL STUDY

1 INTRODUCTION *

A belief network [6] is an acyclic, directed graph containing a set of nodes, a set of arcs and a set of numeric probability distributions. The nodes represent variables, and each node can assume a set of values that are mutually exclusive and exhaustive. Arcs and numeric probability distributions represent probabilistic relationships between nodes. A singly connected belief network is a belief network in which there are no two paths between any two nodes. A multiply connected belief network is a belief network in which there may be more than one path between any two nodes. Figure 1 is a example of a multiply connected belief net with distributions.

Belief networks provide an intuitive knowledge representation for probabilistic models. A belief network contains the information needed to answer all probabilistic queries or make probabilistic inference about the variables in the network, and it provides a computational architecture for the propagation of evidence. The time cost of probabilistic inference in an algorithm is determined by the methods used to perform the probabilistic computations. Since more than one observation could be inserted in a belief net and more than one query could be asked after each observation, the incremental characteristics with respect to observations and queries of an algorithm are important.

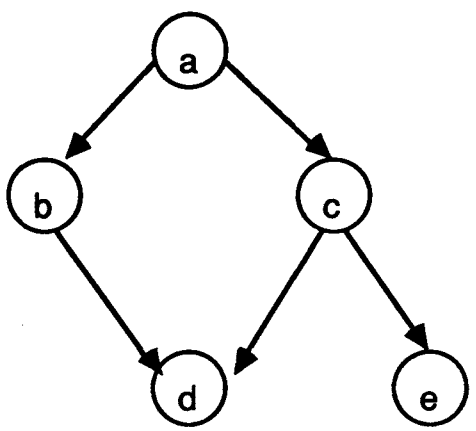
There are three exact numerical methods for dealing with probabilistic inference in a belief network. First, Pearl [6,7] developed an extended representation for belief networks which adds two distributions, lambda and pi, to each node. These numeric distributions, associated with each variable, can be updated by local communication between neighboring nodes. The marginal distribution of each node can then be calculated from local lambda and pi values. The propagation algorithm is only for singly connected belief networks. Related to the propagation algorithm, two algorithms, *Conditioning* and *Clustering* [6,10,12] have been developed for multiply connected

*This research funded in part by NSF IRI88-21660.

networks. The second exact numerical algorithm developed by Shachter [9,10] is *Reduction* which performs probabilistic inference by transforming the original belief network into a network in which only the nodes of interest are left by using node removal and arc reversal techniques. Finally, D'Ambrosio [3] developed a goal-directed symbolic reasoning algorithm (*SPI*) which performs probabilistic inference by logical reasoning and then numeric computation among the nodes concerned.

Since there are several different algorithms for probabilistic inference, it is desirable to compare them and analyze the advantages and the disadvantages of each algorithm, and it would be useful if we could predict the time cost when an inference is made in a given belief network. In this paper the characteristics of four algorithms, Conditioning, Clustering, Reduction, and SPI will be discussed, and the factors related to the time complexity of each algorithm will be analyzed. A set of test cases has been randomly generated for the time complexity experiment. In each test case, several variables have been randomly chosen as observations and several variables from the remaining variables have been chosen as queries after each observation. The predictive model of time cost for each algorithm will be presented after statistical analysis of the test results. Finally, these models will be verified by testing some other randomly generated belief nets.

The implementation of Conditioning, Clustering and Reduction, called 'IDEAL' system, is by Breese and Srinivas at Rockwell and the implementation of SPI is by D'Ambrosio at Oregon State University.



p(a): p(a=1) = .2

p(b|a): p(b=1|a=1)=.8 p(b=1|a=0)=.2

p(c|a): p(c=1|a=1)=.2 p(c=1|a=0)=.05

P(d|b,c): p(d=1|b=1,c=1)=.8
p(d=1|b=1,c=0)=.8
p(d=1|b=0,c=1)=.8
p(d=1|b=0,c=0)=.05

p(e|c): p(e=1|c=1)=.8 p(e=1|c=0)=.6

Figure 1. A multiply connected belief net with probability distributions.

2 AFFECTING-TIME FACTORS IN PROBABILISTIC INFERENCE ALGORITHMS

In this section, we will discuss three families of probabilistic inference algorithms and some factors which influence the computation time of each algorithm.

2.1 Polytree algorithm

Pearl developed an algorithm for updating probabilities in singly connected belief networks [6,7]. In Pearl's algorithm, two elements, lambda and pi, associated with each variable are added in the belief network representation and probability propagation. The process of probabilistic inference is carried out through a series of probabilistic operations, in which each node receives information from its neighboring nodes and combines these messages to update its probability. Let $\mathbf{e}^-_X$ stand for the evidence contained in the tree rooted at x and let $\mathbf{e}^+_X$ stand for the evidence contained in the rest of the network, the lambda and pi message, $\lambda(x)$ and $\pi(x)$, are denoted as

$$\lambda(x) = p(\mathbf{e}^-_X | x)$$

and

$$\pi(x) = p(x | \mathbf{e}^+_X),$$

and the belief distribution of x is

$$BEL(x) = p(x | \mathbf{e}^+_X, \mathbf{e}^-_X) = \alpha p(\mathbf{e}^-_X | x, \mathbf{e}^+_X)\, p(x | \mathbf{e}^+_X) = \alpha p(\mathbf{e}^-_X | x) p(x | \mathbf{e}^+_X) = \alpha\, \lambda(x)\, \pi(x),$$

here $\alpha = [p(\mathbf{e}^-_X | \mathbf{e}^+_X)]^{-1}$ is a normalizing constant rendering $\Sigma_x BEL(x) = 1$.

In Pearl's algorithm, propagation for polytree, there are three steps for local propagation and belief computation which can be executed in any order. Assuming that a typical node x has n parents $U_1, \ldots, U_n$ and m children $Y_1, \ldots Y_m$, the belief distribution of variable x can be computed provided that three types of distributions of node x are available:

1. The current π message contributed by each parent U_i:
$$\pi_x(u_i) = p(u_i | \mathbf{e}^+_{u_i x})$$
2. The current lambda message contributed by each child y_j:
$$\lambda_{y_j}(x) = p(\mathbf{e}^-_{x y_j} | x)$$
3. The conditional probability distribution $P(x | u_1, \ldots, u_n)$.

where $\mathbf{e}^+_{u_i x}$ stands for the observations contained in the ancestors of nodes U_i, and $\mathbf{e}^-_{x y_j}$ stands for the observations contained in the sub-tree rooted at node Y_j. Then three steps can be performed as:

1. Belief updating: The belief distribution of variable x, BEL(x), is

$$BEL(x) = \alpha \lambda(x)\, \pi(x)$$

where

$$\lambda(x) = \Pi_j \lambda_{y_j}(x)$$

$$\pi(x) = \Sigma_{u_1,\ldots u_n} P(x|u_1,\ldots, u_n) \Pi_i \pi_x(u_i)$$

and α is a normalizing constant rendering $\Sigma_x BEL(x) = 1$, and $P(x|u_1,\ldots, u_n)$ is the conditional probability of x given $u_1,\ldots, u_n$.

2. Bottom-up propagation: Using the messages received, node x computes new λ message to be sent to its parents U_i:

$$\lambda_x(u_i) = \beta \Sigma_x \lambda(x) \Sigma_{u_k, k\neq i} P(x|u_1,\ldots, u_n) \Pi_{k\neq i} \pi_x(u_k)$$

3. Top-down propagation: node x computes the new pi message to be sent to its child y_j:

$$\pi_{y_j}(x) = \alpha [\, \Pi_{k\neq j} \lambda_{y_k}(x) \,] \Sigma_{u_1,\ldots u_n} P(x|u_1,\ldots, u_n) \Pi_i \pi_x(u_i)$$

$$= \alpha\, BEL(x) / \lambda_{y_j}(x)$$

The algorithm can be executed in parallel, or sequentially executed in any order for each node. When a belief net is initialized, the pi message of each root node is equal to its prior probability p(x) and all childless nodes are assigned the lambda message as $\lambda(x) = (1, \ldots, 1)$. Then the algorithm starts lambda and pi propagations and belief computations. When observations are inserted, all observed nodes are set as the lambda message as (0,...,0,1,0,...,0) with 1 at the position that the observation is true, and then the propagation is activated. When an observation is inserted, only pi propagation is needed for the sub-tree rooted as the observed node, and lambda propagation for its antecedents and pi propagation for the other children of the antecedents; therefore, it is reasonable to process propagation in certain order in sequential case.

From the three steps above, based on the number of values in each variable and on the number of parents of each node, we know that the time complexity of lambda, pi and belief distribution computations for each node are exponential, because the summation in each formula ranges over all value combinations of parent variables. Computation time of the algorithm is also affected by the location of observations since the positions of observations may decrease the number of parents of a node, and different observations will cause different propagations.

The main limitation of the algorithm is that the performance of belief updates is limited to a singly connected belief net or polytree.

2.1.1 Conditioning algorithm

Since belief nets for practical use are usually not singly connected, Pearl [6] presents two ways of converting a multiply connected belief network into a singly connected network, and then applying his algorithm. One of the methods is conditioning, which is based on the idea of changing the connectivity of a belief network and rendering it singly connected by instantiating a selected group of variables.

An intuitive way of converting a multiply connected directed graph to a singly connected graph is removing extra arcs in the graph, such that no two directed pathways exist between any two nodes. The process can be performed in a belief network by instantiating a variable in a pathway instead of removing an arc since there is a characteristic in a belief network that an instantiated variable in a pathway will block information passed between its parents and its children and among its children. We call the set of variables to be instantiated a *cutset*. Given a multiply connected belief net with variables x_1, x_2, ..., x_n, the method of conditioning for querying a variable x given observations E is to choose a group of variables x_{i_1}, x_{i_2},..., x_{i_k} from the network and instantiate them so that the information passing in the network is like that in a singly connected belief network, then use the polytree updating algorithm for probability distribution calculation. Obviously, the number of instantiations of the chosen variables is the product of the number of values of each variable, that is, the times that the propagation algorithm is used for probabilistic inference is exponential in the number of variables to be instantiated. The final result of inference is determined by averaging the whole intermediate results weighted by the posterior probabilities. The formula for calculating the query is:

$$P(x|E)= \Sigma P(x|E, x_{i_1},...,x_{i_k}) * P(x_{i_1},...,x_{i_k}|E)$$

in which, $P(x|E, x_{i_1},...,x_{i_k})$ and $P(x_{i_1},...,x_{i_k}|E)$ can both be calculated by the polytree algorithm.

Since the computation time is exponential in the number of variables in a cutset, minimal cutset in a belief net is desirable. Unfortunately, finding a minimal cutset in a belief net is NP-hard [12]. Suermondt and Cooper [12] proposed a heuristic algorithm for finding a cutset in polynomial time with respect to the number of nodes. The main steps of the algorithm are as follows:

A. Remove all nodes that have a single parent and no children and the nodes that have single child and no parent, remove the arcs that connected the pruned nodes.

B. If there are any nodes left, find a good cutset candidate. Three steps for choosing a good cutset candidate are:

1. Choose the nodes that have one or no parents;
2. Choose the node from step 1 that has the most neighbors;
3. If there is more than one node, choose the node that has the lowest number of possible values.

Add the node chosen above into the cutset, then remove it from the network. If there remain nodes in the network, return to step A.

Figure 2 gives an example of using the conditioning algorithm to query the probability of node d given the observations e =1 in the belief net in figure 1. Using the algorithm above, we can obtain the cutset which has one variable A. The belief net in figure 2(a) is a singly connected graph by instantiating the cutset variable A with value 0 and the figure 2(b) shows the same graph with the instantiation of A=1.

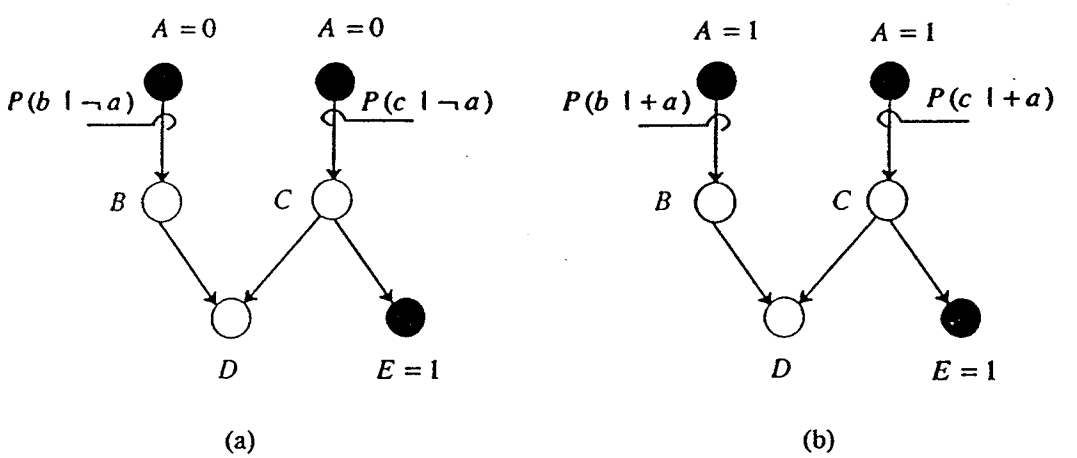


Figure 2. The example of computing the probability p(d|e=1) by using the conditioning algorithm.

The time complexity of the algorithm is $O(n^2)$. This algorithm does not guarantee to find a minimal cutset, but the experiment with 60 randomly generated belief networks showed that it found the minimal cutsets in approximately 70 percent of the networks. Therefore, the conditioning algorithm is exponential in cutset size which determines the number of times the polytree algorithm will be used and is exponential in the maximum antecedents of a node in the

remaining polytree when using the polytree algorithm for each instantiated polytree. That is $_2O(\text{ecutset\_size} + \text{number\_of\_parents\_not\_in\_cutset})$.

Different belief nets may have the same cutset_size but different computation time for some queries. There are some other parameters , like the number of nodes, number of arcs per node, the maximal in-degree in a belief net and so on, which may also affect the computation time. How these parameters are related to the computation time is unknown since there exist many different belief nets given the same number of nodes and number of arcs, but they may tell us roughly how complicated a belief net is. Also, the number of queries and the number of observations are significant factors affecting the computation time of some algorithms. Thus, we will consider these parameters as *affecting-time factors* that influence computation time in each algorithm.

In the conditioning algorithm, the number of nodes, arcs, maximal in-degree in a belief net, number of observations and cutset size should be considered as factors that influence computation time in the algorithm. Number of queries is not such a factor because any marginal probability can be calculated easily after local propagation of lambda and pi distributions. The number of observations will cause different time cost depending on whether an observed variable decreases the cutset size or not. If an observed variable does not decrease the cutset size, the time of querying any variables will be a little longer than that of querying the same variables before inserting the observation because more time will be used for propagating the observation; but if an observation decreases cutset size, the querying time will be almost cut in half (assuming that the observed variable has two values). Therefore, it is hard to predict the total querying time after a group of observations. It is possible to predict querying time after first observation if we know the cutset size of the belief net according to theoretical analysis of the algorithm.

2.1.2 Clustering algorithm

Another method of handling multiply connected belief nets which uses the polytree algorithm is clustering. Clustering involves forming compound variables (a compound variable consists of several variables in a belief network) in such a way that the resulting network of compound variables is singly connected. Generally, there are many different ways to cluster a belief network since any two nodes in a belief network can be merged to one clique node if there is only one directed path connecting the two nodes or if there is no directed path between them.

In the extreme case, a clustered tree can be formed by lumping together all non-leaf variables as one variable. However, the exponential cardinality of a compound variable and its structureless nature make it difficult to compute and explain the beliefs accrued by individual hypotheses within this variable. Pearl [6] proposed a strategy of clustering, and Lauritzen and Spiegelhalter [5] presented a modified version.

There are two parts to the clustering algorithm. First, create cliques (a clique is a maximal subset of nodes which is a complete sub-graph) and form a join tree (a tree with cliques as nodes). Second, update the tree when any observations are inserted. Creating cliques and forming a join tree involves the following steps:

1. Convert the belief network into a Markov network by interconnecting the parents of each node and then making the original directed edges in the belief network non-directed. A Markov network is an undirected graph in which unconnected nodes are conditionally independent.

2. Triangulate the network by using the maximum cardinality search, which transforms the belief net into a chordal graph (a chordal graph is a undirected graph in which every cycle of length of four or more has a chord). After triangulation, the chordal graph can be decomposed into a set of cliques that have the running intersection property. The running intersection property here means that when all cliques are sorted in an order of $(C_1, C_2, ..., C_n)$, then for all $j \geq 2$, there exits $i < j$ such that $C_i \supseteq C_j \cap (C_1 \cup ... \cup C_{j-1})$.

3. Sort the cliques in the increasing order of the maximum index number of nodes in a clique and assemble them in a join tree by connecting each clique node C_i to a clique node C_j $(j < i)$ sharing the maximum index number of nodes with C_i. Each clique is now modeled as a clique node in the join tree.

4. Set up the conditional probability distributions of clique nodes according to the links in the join tree. Each combination of states of the component belief net nodes of the clique node is one state of the clique node.

Figure 3 shows one join tree generated from the belief net in figure 1 by the algorithm above.

The time complexity of step 1 is $O(n^2)$. Step 2 needs $O(n + e)$ time for triangulation according to Tarjan and Yannakakis's algorithm [5], here n is number of nodes and e is number of arcs in a belief net. So it is $O(n^2)$ for that e is $O(n^2)$ at most. There are at most (n-1) cliques in a belief net since any node and its parents form a clique. Since the number of clique nodes in a join

tree is less than the total nodes in the belief net, the complexity of forming the join tree in is $O(n^2)$ in step 3. In step 4, setting up probability distributions is time consuming. The states of a join node are determined by the combination of all belief nodes in that join node, so the operations expected to compute the probability distributions for a join node are exponential in clique size of the join node or number of belief nodes in that join node. This indicates that the maximal clique size in a join tree is a key factor of considering time complexity in probabilistic inference in clustering algorithm and that it can be computed at time cost $O(n^2)$ in the number of nodes in a belief net.

When observations are inserted, the join tree will be updated by local computation of lambda, pi and probability distributions for each clique node. Three steps perform this function:

1. Create dummy clique nodes that send lambda messages to the clique nodes affected by the observations.
2. Update the join tree by using the polytree algorithm. The belief of a clique node is the joint probability of appropriate states of the component belief net nodes, given observations used for inference in the join tree.
3. Calculate the beliefs of each of the belief net node by finding the smallest clique node in which the belief net node is a member, and then marginalizing the clique node's belief over the states of the other component nodes of the clique node.

The time cost for step 1 and step 3 is very low and can be ignored compared to step 2. The complexity of step 2 results from using the polytree algorithm which is exponential with respect to the number of parents of a node. It seems that it shouldn't take very long because the number of parents in the join tree for any join node is at most one. But the key point is that, since states of a clique node are determined by the states of the belief nodes in that clique, the calculation of clique states affected by the observation is exponential in clique size based on the number of values of variables. Furthermore, one observation may affect more than one clique node and so the time cost will be linear in the number of clique nodes affected. From analysis above we know that it is possible to use the maximum clique size to estimate the total updating time for the algorithm.

Observations are handled as dummy clique nodes in the algorithm and propagation provides incrementality with respect to observations. These dummy clique nodes remain attached to the cliques unless the observations are changed by new observations relating to the nodes

inserted. The join tree can be repeatedly used for inference when new observations added. Thus the time cost is polynomial in number of observations.

Considering two parts together we know that clustering algorithm is exponential in clique size when using polytree algorithm for probabilistic inference. The factors that influence computation time should also include the number of nodes and number of arcs which will affect the number of cliques and the number of cliques which have the maximum clique size, and number of observations which will affect clique size of some clique nodes. The number of queries is not a affecting-time factor since the marginal probability of a belief node can easily be calculated from the join node it exists in. As in the conditioning algorithm we usually consider the total time to be polynomially related to the number of nodes and arcs in a belief net.

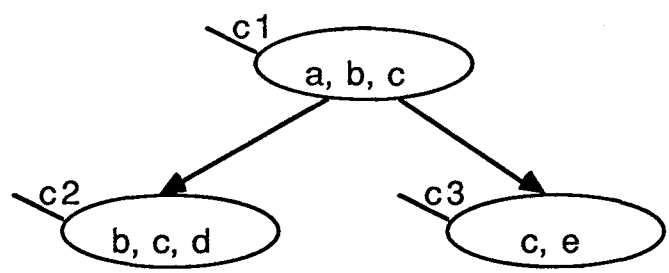


Figure 3. The join tree generated from the belief net in figure 1, and c1, c2 and c3 are clique nodes.

2.2 Reduction algorithm

Reduction is another method for probabilistic inference in a belief network. Schachter [9,10] outlined a goal-directed algorithm for belief net transformations. The main idea of the reduction algorithm is as follows: given a belief net with n nodes representing probabilistic relations of n variables x_1, x_2, ..., x_n, then the marginal probability of any variable x_i given observations x_{i_1}, ..., x_{i_k}, that is $p(x_i|x_{i_1}, ..., x_{i_k})$, corresponds to a particular graph. This graph can be derived from the original belief net by node removal and arc reversal; these transformations maintain the consistency between the original net and the transformed net for the query.

Rege and Agogino [8] developed a heuristic algorithm for the reduction process. The algorithm takes $O(n^2)$ storage and $O(n^3)$ time for symbolic or topological transformation. The symbolic transformation algorithm removes nodes and reverses arcs, until only conditioning and

conditional nodes are left. Letting C be the set of conditioning and conditional nodes and K be the other nodes to be removed, the five steps of the algorithm are:

1. Remove all nodes in K that have no successors. Repeat until all nodes without successors are removed. If K is empty go to 5.

2. Remove all nodes in K that have only one successor each. If more than one node is to be removed, remove them in the order of fewer predecessor first. Repeat until all nodes with one successor are removed. If K is empty go to 5.

3. Select the node in K with least number of successors, let the node be x.

4. Pick a successor of node x and check the number of paths from x to the successors; if there is more than one path, reject it; otherwise reverse the arc between x and that successor with concomitant updating of their predecessor and successor set according to Bayes theorem. Check if x has only one successor, if so remove x and if K is not empty go to 2 otherwise go to 5. If x has more than one successor go to 4.

5. For every successor of a conditional node, check for more than one path between conditional node and that successor. If there is more than one path, reject that successor; if not, keep track of the successor with the least number of predecessors (y). Reverse the arc between the conditional node and y with a concomitant updating of successor and predecessor sets. Repeat 5 until the conditional node has no successors.

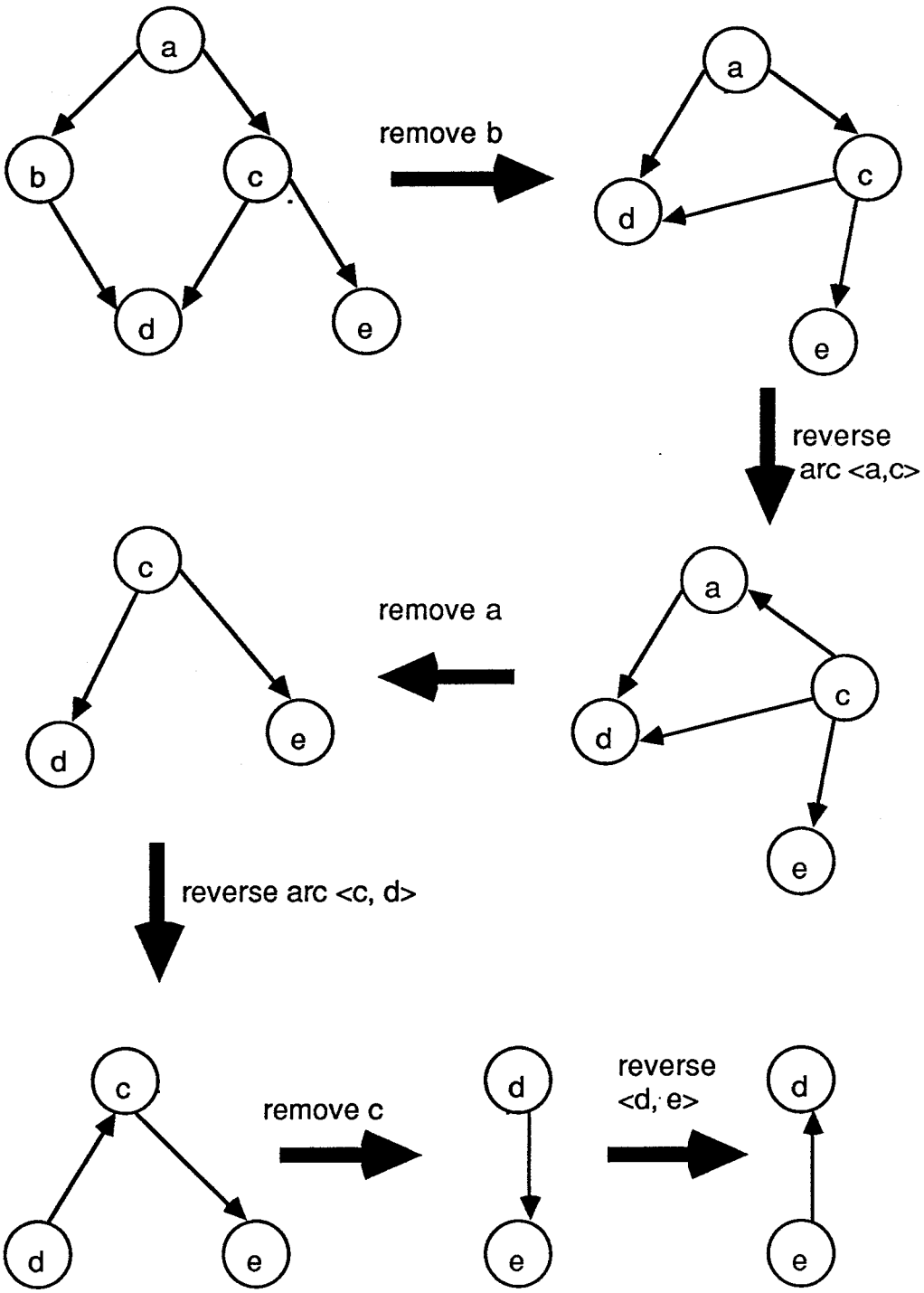
Figure 4 shows the process of transforming the belief net in figure 1 to the belief net which corresponds to querying the variable d given the observation of e, i.e. p(d|e=1).

Most of the cost results from calculating conditional probability in arc reversal. Assume that node x_i conditionally depends on nodes w_i, node x_j conditionally depends on node x_i and nodes w_j, and both xi and x_j conditionally depend on nodes w_{ij}, then after arc reversal between node x_i and x_j, the conditional probability of $P(x_j|w_i, w_j, w_{ij})$ and $P(x_i|w_i,w_{ij})$ will be:

$$P(x_j|w_i,w_j,w_{ij})= \sum_{xi} P(x_j|x_i,w_j,w_{ij})*P(x_i|w_i,w_{ij})$$

$$P(x_i|x_j,w_i,w_j,w_{ij})=P(x_j|x_i,w_j,w_{ij})*P(x_i|w_i,w_{ij})/p(x_j|w_i,w_j,w_{ij}).$$

From the above formulas we know that the cardinalities of x_j and x_i are usually increased after arc reversal, and calculation of the conditional probability distribution is exponential in the number of parents of node x_j and x_i. We use the word *dimensionality* of node x_i, to denote the number of arcs from set of nodes { x_j, w_i, w_j, w_{ij}} to node x_i, then the probability computation for node x_i will be exponential to its dimensionality.



The maximum dimensinality is 4.

Figure 4. The process of transforming the belief net in figure 1 into the belief net which corresponds to querying the conditional probability p(d|e=1).

Since a query $P(x_i|x_{i1},x_{i2},...x_{ik})$ will generate a corresponding belief net from the original net, it is hard to use any symbolic intermediate result for new observation or querying another variable; no caching is implemented in the algorithm and any query will start transformation from the original belief net. Obviously, if we know the maximum dimensionality in a belief network transformation, the time cost in the algorithm is intimately related polynomial-time to the number of queries and exponential-time to the maximum dimensionality. The number of nodes and the number of arcs should also be the affecting-time factors in the algorithm since they are closely related to the algorithm. The observations is not a affecting-time factor since it is carried out in the step 5 very quickly.

A parameter which records the maximum dimensionality of computing numeric probability in transforming original belief net to a desired net is chosen as affecting-time factor of the algorithm in our test and it can be computed at $O(n^3)$ in the number of nodes in the belief net.

2.3 SPI algorithm

The symbolic probabilistic inference algorithm (SPI) developed by D'Ambrosio [3] is a query-driven algorithm which uses Bayes theorem directly for probabilistic inference in multiply connected networks. In this algorithm, probabilistic inference occurs in two steps. In the first step, symbolic reasoning determines which nodes in a belief net should be queried or computed according to current marginal or conditional queries; then the probability computations are carried out for those nodes.

Symbolic representation of a node in SPI is the node marginal in terms of its conditional distribution and the immediate antecedents needed for computation. When a node is queried, its marginal probability can be calculated from its symbolic expression if all marginal probabilities of its immediate antecedent's are known. If some of them are unknown, then sub-queries are generated for these nodes. This is a recursive process which continues until all values needed are known. All queries and sub-queries are processed according to the structure of the belief net; it is possible to cache some intermediate results for some nodes. If computation proceeds in the same path as before, cached results can be used.

In order to get a better factoring result, a partition strategy , which also supports intermediate result caching, is used in SPI, and an ordering heuristic algorithm is used before the probability computation. A partitioned belief net is a tree structure. Associated with each partition is a *conditioning* expression which stores the union of the expressions for the observed value of each observed variable in that partition. There is one conditioning expression for each partition because any observation in a partition will only affect its child partition node.

There are three steps in the SPI algorithm:

1. Create a partition tree from the belief net.
2. After each observation, modify the conditioning expressions of the belief net in the partition affected by the observation.
3. When querying a variable x, use the formula

$$P(x) = \text{Union}(\exp(x), \exp(\textit{conditioning}_{root}))/\ \text{Union}(\exp(\textit{conditioning}_{root}))$$

to compute its probability. For conditioning queries or joint probability, for example p(x&y|z), use the following formula

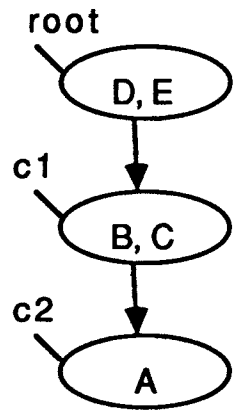
$$P(x\&y|z) = \text{Union}(\exp(x), \exp(y), \exp(z), \exp(\textit{conditioning}_{root}))/\text{Union}(\exp(z), \exp(\textit{conditioning}_{root})).$$

Here, $\textit{conditioning}_{root}$ is the conditioning expression for the root partition. The symbolic reasoning is carried out first; it generates whole expressions relevant to the current query. If the values of some of the expressions in the above formula are unknown, some other expressions related to computing the unknown expressions are generated and computed by using a similar formula as above. This process is carried out recursively until all expressions needed are obtained. Usually only those conditioning expressions which affect the queried node would be considered as the conditioning expression in the above formula according to Bayes rule, but it will take time to determine which observation would affect a queried node. Therefore, it is better to consider all conditioning expressions in a partition tree, starting conditioning from the root partition then recursively to all its child partitions until an observation is met , which blocks propagation of the observation from other nodes in that branch, or until the leaves of the partition tree are met. The cost of always starting in the root is lower than the cost of finding the lowest valid starting partition. If there is not a conditioning variable in the query, the probability computation for the denominator can be replaced by a normalizing factor. Figure 5 shows the partition tree created from the belief net in the figure 1, the internal expressions of the variables in the belief net and conditioning expression for each partition node, and the process for computing the probability p(d|e=1).

Step 1 is a heuristic partitioning algorithm; its complexity is $O(n^2)$. In step 2, after each observation, the algorithm will change the representations of the children of the observed node. The algorithm resets the expression of the observed node as 1.0 and its value space is a singleton consisting of only the observed value. It also resets the *conditioning* expression to the union of its current expression and the expression for the observed value of the observed node; where union is an operation defined for computing the expression for certain joint distributions. Hence, the representation in SPI can be modified incrementally in observations. Since there is a caching strategy in the algorithm, there should be a cache modification after each observation. The cache invalidation strategy used here is the following: when an observation is inserted, all caches on the path from that partition to the root are invalidated. The time cost in step 2 is very little compared to step 1 or step 3. Most time is spent in step 3. The time of computing any two expressions in the formula in step 3 is exponential in number of variables in the two expressions or its dimensionality. Since computation time is exponential in the number of variables in two expressions, it is important to factor them as optimally as possible. The strategy used here is to do a static factoring of the probability space prior to the processing of any queries and then to use fast expression ordering heuristics to minimize the cost of computations within each part of static factoring. The time complexity of symbolic reasoning is $O(n^3)$ in the number of nodes since for each partition node union, decomposition and ordering operations are needed which are of time complexity $O(e^2)$, $O(e)$ and $O(e^2)$ respectively. Here e is the number of factors in factoring. Therefore, the total time used for querying a node mainly depends on the number of factors and the maximal dimensionality in the query.

From the analysis above we know that affecting-time factors in SPI algorithm are dimensionality of a factor which can be computed at $O(n^3)$ in the number of nodes in a belief net, number of queries, and maybe the number of nodes and the number of arcs. Time complexity is exponential in the maximum dimensionality in a query, polynomial in number of queries. The *number of observations is not a time affecting-factor since it takes very little time for modifying* affected partitions and related probability expressions. Since the factoring algorithm is heuristic and the number of factors is related to the paths retrieved in the belief net, the time model consisting of above parameters may not fit real test time very well.

Partition tree:



Observation: e=1:

The internal expressions:

exp(A)= A

exp(B)=$B_{|A}A$

exp(C)=$C_{|A}A$

exp(D)=$D_{|BC}BC$

exp(E)=$E_{t|C}C$

Conditioning root = $E_{t|C}C$

Conditioning$_{c1}$ = 1.0

Conditioning$_{c2}$ = 1.0

The process of computing the probability of p(d|e=1)

posterior(D)= Union(exp(D), *Conditioning root*) = Union($D_{|BC}BC$, $E_{t|C}C$) = $D_{|BC}$ B C $E_{t|C}$

Union(exp(B), exp(C), *Conditioning*$_{c1}$) = Union ($B_{|A}A$, $C_{|A}A$, 1.0)= $B_{|A}$ $C_{|A}$ A

Union (exp(A), *Conditioning*$_{c2}$) = Union(A, 1.0) = A

posterior(D) = $D_{|BC}$ $B_{|A}$ $C_{|A}$ A $E_{t|C}$ =P(D|B,C)P(B|A)P(C|A)P(A)P(E=t|C)

= $[E_{t|C}(\Sigma_B D_{|BC}[\Sigma_A B_{|A}(C_{|A}\ A)]_{c1})]_{root}$

The maximum dimensionality is 3.

Figure 5. The partition tree created from the belief net in the figure 1, the internal expressions of the variables in the belief net and conditioning expression for each partition node, and the process for computing the probability p(d|e=1).

2.4 Summary

The characteristics of the four probabilistic inference algorithms are summered in table 1.

	CONDITIONING	CLUSTERING	REDUCTION	SPI
BASIC APPROACH	Polytree Propagation Algorithm	Polytree Propagation Algorithm	Node Removal Arc Reversal	Goal-direction Algorithm
FACTORS THAT INFLUENCE COMPUTATION TIME	Number of nodes Number of arcs Number of observations Cutset size Number of parents	Number of nodes Numberof arcs Number of observations Clique size	Number of nodes Number of arcs Number of observations Number of queries Dimensionality	Number of nodes Number of arcs Number of observations Number of queries Dimensionality
TIME COMPLEXITY	Exponential in cutset size and max antecedent	Exponential in max clique size	Exponential in max dimensionality	Exponential in max dimensionality
INTERMEDIATE RESULT CACHING	No	No	No	Yes
INCREMENTAL TO OBSERVATIONS	No	Yes	No	Yes

Table 1. The characteristics of the four probabilistic inference algorithms.

3 EXPERIMENTAL STUDY

Once we determined qualitatively the factors influencing computation time for each algorithm, we turned our attention to developing a quantitative model of these influences. The methodology used was to generate some test cases with the time affecting-factors chosen randomly; run each algorithm with these test cases, analyze the test results statistically and find a good regression model for each algorithm; and finally verify the predictive models with a second set of randomly generated test cases.

3.1 Input data and test results

We use J. Suermondt't random net generator to generate all test cases. This generator starts with a fully connected belief net of size n, and removes arcs selected at random until the number of the remaining arcs is a selected value. There were twenty belief nets generated for experiments initially. They were randomly generated in the range of ten to thirty nodes and 1.0 to 5.0 average incoming arcs per node. Since some of the belief nets were too big for one or another algorithm to run in a reasonable time, another six belief nets with nodes ranging ten to thirty and 1.0 to 1.6 average arcs per node were added to the experiments. Table 2 presents some characteristics of the 26 belief networks; **max-ant** is the maximum number of antecedents of any node in a belief net; **cutset-size** is the maximum cutset size in the conditioning algorithm; **clq-size** is the maximum clique size in the clustering algorithm; **rdct-dm** is the maximum dimensionality in the reduction algorithm; and **spi-dm** is the maximum dimensionality in the spi algorithm. The *, in all tables, denotes an unknown value since the algorithm could not finish running that test cases in ten hours.

net	nodes	arcs	arc/node	max_ant	cutset-size	clq-size	rdct-dm	spi-dm
1	23	28	1.2	5	3	6	9	6
2	13	62	4.8	8	9	11	10	9
3	13	61	4.7	10	9	11	11	10
4	18	85	4.7	11	*	*	14	13
5	16	54	3.4	7	10	11	12	7
6	17	34	2	5	7	9	6	6
7	23	60	2.6	9	*	13	15	12
8	10	15	1.5	3	3	4	6	4
9	27	35	1.3	4	3	5	9	5
10	12	26	2.2	5	5	7	6	6
11	23	87	3.8	9	*	13	18	12
12	11	36	3.3	6	4	7	8	7
13	14	15	1.1	3	1	4	4	4
14	16	40	2.5	5	7	8	9	6
15	19	76	4	10	*	*	*	13
16	29	131	4.5	11	*	*	*	*
17	29	90	3.1	8	*	*	*	16
18	16	35	2.2	5	4	7	8	6
19	15	53	3.5	10	9	11	14	10
20	26	101	3.9	10	*	*	*	13
21	28	34	1.2	4	2	5	5	4
22	25	30	1.2	4	3	5	7	6
23	22	29	1.3	4	5	5	9	5
24	27	41	1.5	5	6	8	7	6
25	25	25	1	4	1	5	8	5
26	16	16	1	5	1	6	6	5

Table 2. The characteristics of 26 randomly generated belief nets for experiments.

Table 3 presents the test results of all belief networks. In each test case, we randomly determined the number of observations to be inserted in that test case, then we randomly choose each observation from all variables in the belief net and finally we choose at random a set of variables as queries from remaining variables after each observation. The total observations and total queries are shown in columns **obs** and **query** and the columns from 4 to 7 present the total computation time in seconds. We generated all test cases as above because we think that all factors in a belief net generated at random may give us more reasonable test results and be closer to realistic models. The experimental results show that SPI gives the best performance of the algorithms. The best performance of SPI is due to the characteristics of the algorithm, goal-directed reasoning, caching intermediate results, and its incremental nature with respect to observations. But caching and incremental observations are not key points for speeding the algorithm. The test [3] shows that caching significantly improved the performance but not dramatically (about 10 to 40 percent, see [3]). The key point is the goal-directed reasoning which only considers the nodes relevant to the query and the numeric computations which are carried

out among those nodes. In contrast to the SPI, in the polytree propagation algorithm, when an observation is inserted, the message passes through all nodes; querying one variable will take almost the same time as querying all variables. Another advantage of SPI is that the heuristic strategy of factoring the probability space in SPI decreases the factor dimensionality in computation.

net	obs	query	cond/t	cluster/t	reduct/t	spi/t
1	23	142	265.82	924.70	114.81	4.3
2	7	27	*	*	304.85	3.14
3	10	33	*	*	1067.88	6.9
4	17	97	*	*	6517.16	6.95
5	9	49	*	*	540.55	2.72
6	1	1	229.28	5037.58	0.62	0.21
7	5	40	*	*	1215.84	9.42
8	10	31	55.55	25.84	10.63	0.62
9	27	127	666.45	709.02	280.83	5.74
10	1	8	80.84	438.00	5.22	0.27
11	4	22	*	*	35608.22	20.56
12	9	17	1136.6	3754.33	40.29	0.86
13	12	30	19.32	24.88	7.23	0.55
14	9	34	3983.83	10312.11	55.26	1.24
15	16	74	*	*	*	43.4
16	26	199	*	*	*	*
17	12	142	*	*	*	367.95
18	13	53	544.99	4099.10	44.56	1.31
19	15	48	*	*	784.25	3.85
20	11	98	*	*	*	81.79
21	4	46	37.48	72.92	28.53	0.86
22	9	130	117.49	195.57	95.15	4.29
23	10	88	247.15	208.34	121.23	2.4
24	1	25	380.19	1187.11	32.67	1.48
25	13	48	83.12	95.65	154.21	4.03
26	13	57	125.03	112.06	25.28	0.8

Table 3. The test results of 26 belief nets for the different algorithms. * denotes an unknown value. All algorithms are implemented in Common-Lisp and run in the SUN/4.

3.2 Predictive models

We know that the cutset can be found by using Cooper's heuristic algorithm at time cost only $O(n^2)$ in the number of nodes in a belief net; the maximal clique size can be computed at time cost $O(n^2)$ in the number of nodes in a belief net; the maximum dimensionality in the reduction algorithm can be computed at the time cost $O(n^3)$ in the number of the node in a belief net; and the maximum dimensionality in the SPI can be computed at the time cost $O(n^3)$. The time

cost for computing the affecting-time factors for each algorithm is trivial compared with the real probability computation. We used stepwise regression by hand to test different models for each algorithm, and we choose one which fits best to each algorithm.

3.2.1 Conditioning

Figure 6 is the statistical analysis for 17 test results of 26 generated test-cases; the rest of them can not be run in a suitable time. The test results show that the running time after first observation is exponential in cutset size and maximal in-degree of a belief net, and is polynomial in number of nodes and average arcs per node. It also shows that the test results and theoretical analysis above fit pretty well. One caveat the implementation of the conditioning algorithm in IDEAL system does not perform any intermediate result caching. When a new observation is inserted, the algorithm starts to consider all observations inserted so far for a query.

Response: ln(time/1st obs.)

Summary of Fit

Rsquare	.9884395
Observations (or Sum Wgts)	17

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-5.068423	1.07096	-4.73	0.0005
cutset	.44784380	.065629	6.82	0.0000
ln(nodes)	1.1048083	.393202	2.81	0.0158
ln(arcs)	2.4039352	.571994	4.20	0.0012
anticedents	.60619205	.135307	4.48	0.0008

Whole-Model Test

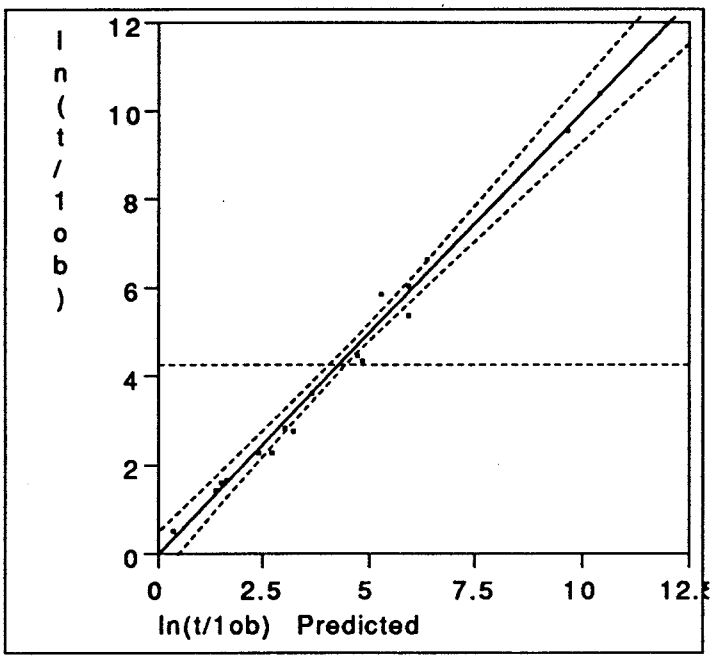


Figure 6. Statistical result of testing Conditioning algorithm.

The predictive model is:

ln(time) = -5.068 + 1.15 ln(nodes) + 2.404 ln(arcs) + 0.448 cutset + 0.606 antecedents.

3.2.2 Clustering

Figure 7 shows the regression result for the clustering algorithm using 15 test cases of 26 randomly generated belief networks.

Response: ln(time)

Summary of Fit

Rsquare	.9829846
Observations (or Sum Wgts)	15

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-7.514065	.927943	-8.10	0.0000
ln(nodes)	1.740452	.310834	5.60	0.0002
ln(arcs)	2.8928964	.394450	7.33	0.0000
ln(obs)	.71854693	.092004	7.81	0.0000
cliques	.97732662	.091748	10.65	0.0000

Whole-Model Test

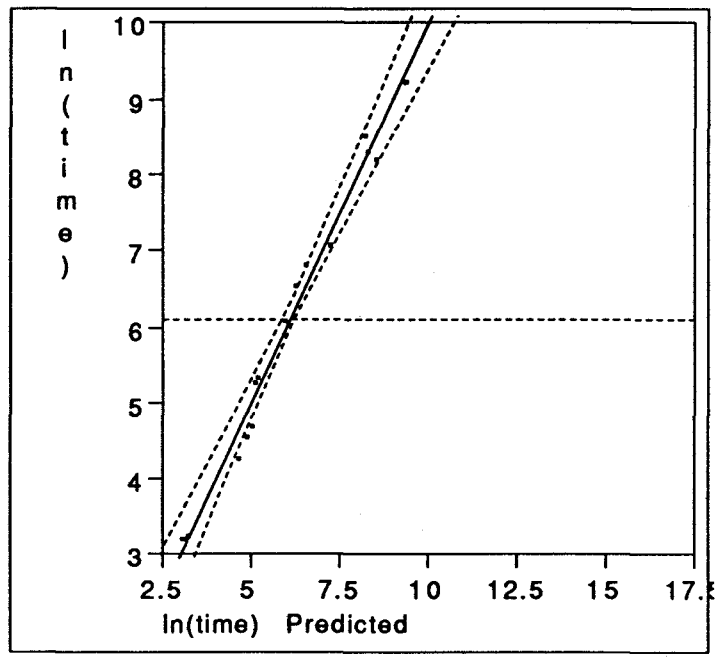


Figure 7. Statistical result of testing Clustering algorithm.

The predictive model is:

ln (time) = -7.514 + 1.741ln(nodes) + 2.893 ln(arcs) + 0.719ln(obs) + 0.977 cliques.

3.2.3 Reduction

Figure 8 shows the regression result for the reduction algorithm with 22 test cases.

Response: ln(time)

Summary of Fit

Rsquare	.9215450
Observations (or Sum Wgts)	22

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-3.225386	.619687	-5.20	0.0001
dimensionality	.56022507	.043936	12.75	0.0000
ln(query)	.76246946	.147666	5.16	0.0001

Whole-Model Test

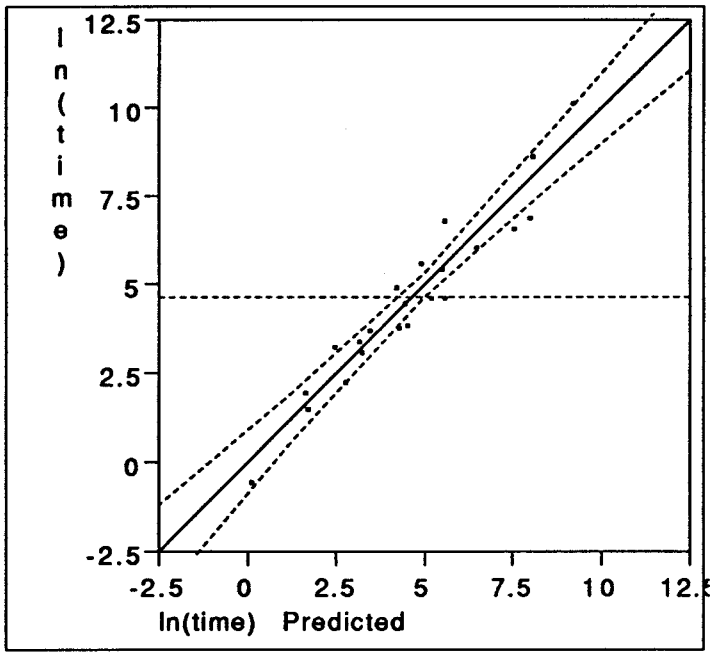


Figure 8. Statistical result of testing Reduction algorithm.

The predictive model is:

ln(time) = -3.225 + 0.763 ln(query) + 0.5602 dimensionality.

3.2.4 SPI

Figure 9 shows the regression result using the same randomly generated test cases. The number of arcs is deleted from the model for that it is not significant in the statistical analysis.

Response: ln(time)

Summary of Fit

Rsquare	.9148749
Observations (or Sum Wgts)	25

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-7.324417	1.0294	-7.12	0.0000
dimensionality	.35215165	.033013	10.67	0.0000
ln(query)	.55869691	.119700	4.67	0.0001
ln(nodes)	1.2635668	.395430	3.20	0.0043

Whole-Model Test

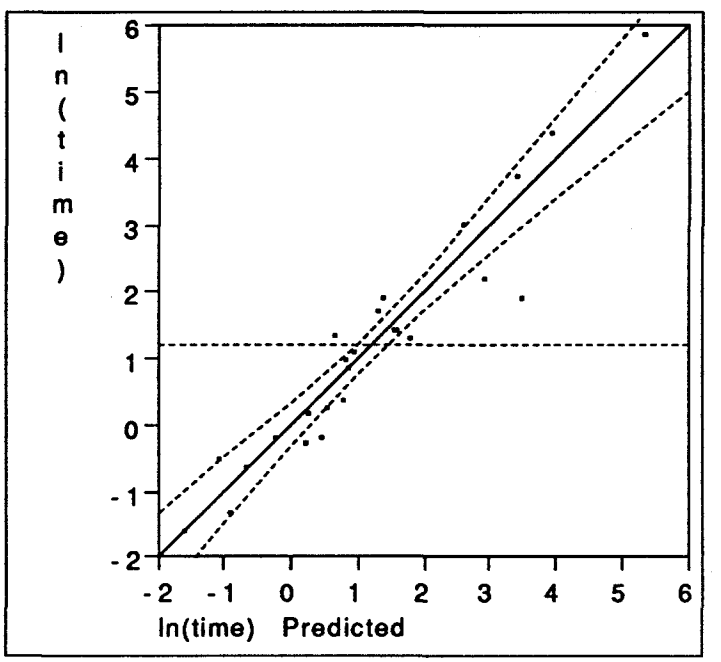


Figure 9. Statistical result of testing SPI algorithm.

The predictive model is:

ln(time) = -7.325 + 1.264 ln(nodes) + 0.559 ln(query) + 0.352 dimensionality.

3.2 5 Discussion

In the above time predictive models, Conditioning, Clustering and SPI show that the affecting-time factors in these models are consistent with the theoretical analyses in the section 2. However, we did not see the expected sensitivities to nodes and arcs as in the reduction algorithm. We surmise the reasons for non-sensitivity of some factors are two: one is that the reduction heuristic masks node and arc sensitivity by going into the higher dimensions. That is, the local nature of the reduction algorithm makes the reduction process more likely to be non-optimal as number of nodes increases, and the other is that the node removal strategy in the algorithm makes the algorithm less sensitive to the number of nodes and the number of the arcs.

3.3 Validation test

Another 13 belief nets were generated for testing the predictive models. They were randomly generated in the ranges of ten to thirty nodes and 1.0 to 5.0 average arcs per node. Since some of the belief nets are too big for one or another algorithm to run in a reasonable time, another seven belief nets with nodes ranging from ten to thirty and 1.0 to 1.6 average arcs per node were added. The number of observations and the number of queries after each observation were randomly determined as well. The following figures (Figures 10 to 13) show the comparison between the time cost calculated from the predictive models and the real time cost of running those algorithms. The ln(estimate) axis represents the predicted time in natural logarithm and the ln(time) axis represents the real computation time in natural logarithm. From the figures, we know that the predictive models give very close estimates for the real computation time. The difference between the estimated time and the real time cost is at most five-fold in the all predictive models. Since the running time on the same inputs varies as much as twice its value sometimes, the predictive models are acceptable.

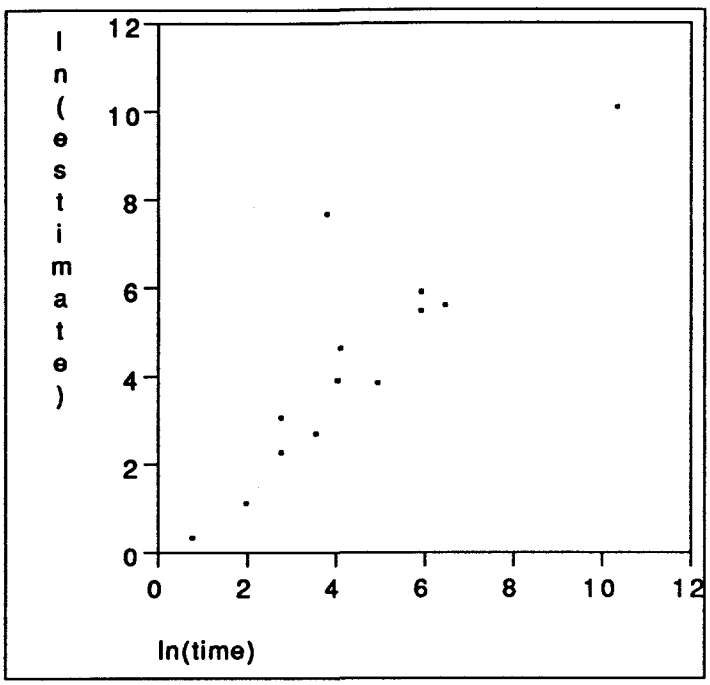


Figure 10. Comparison between the predicted time , ln(estimate), and real computation time, ln(time) for the conditioning algorithm. There is one point in the figure which dose not fit the real computation time well; this is because the observation reduces the number of cutsets by 1 in real computation, so it took less time than estimated.

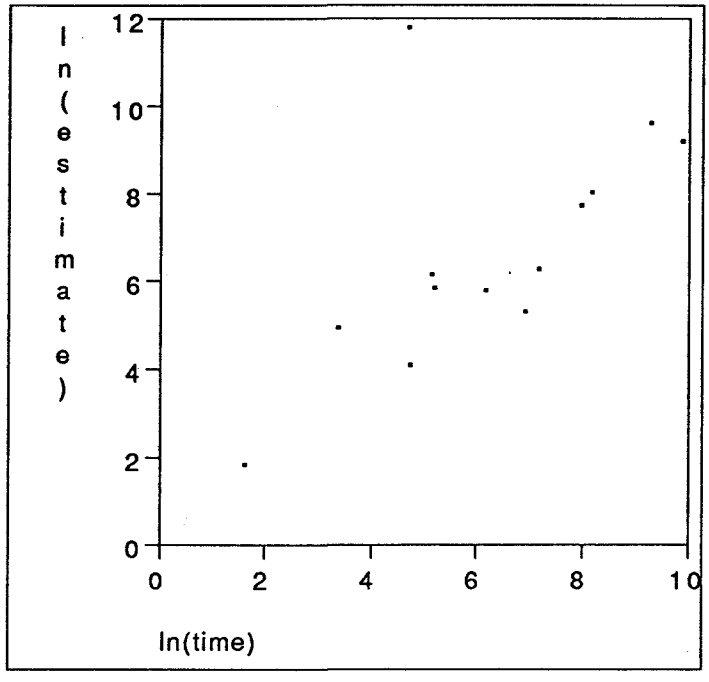


Figure 11. Comparison between the predicted time , ln(estimate), and real computation time, ln(time) for the clustering algorithm. There is one point in the figure which dose not fit the real computation time. In this test case, since there is only one clique node in the belief net, there is no propagation needed after each observation. The real time cost is much smaller than estimated.

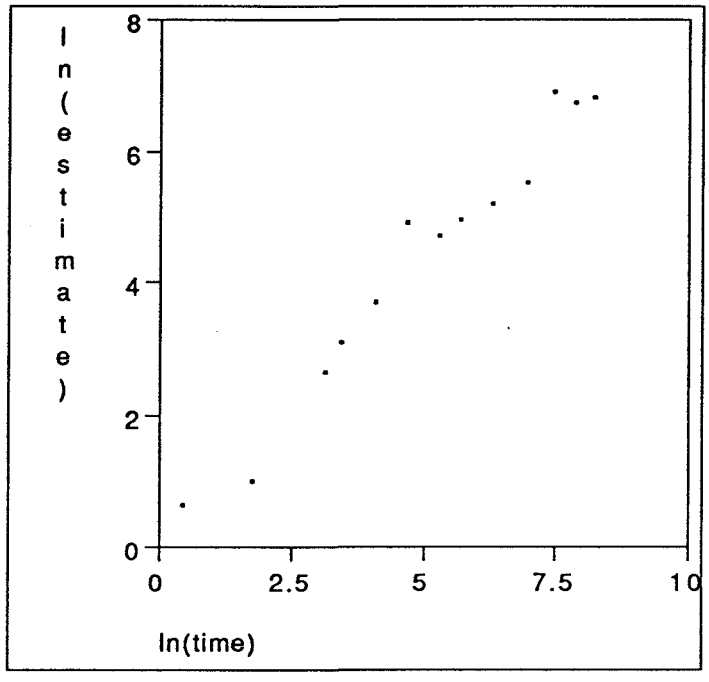


Figure 12. Comparison between the predicted time , ln(estimate), and real computation time, ln(time) for the reduction algorithm.

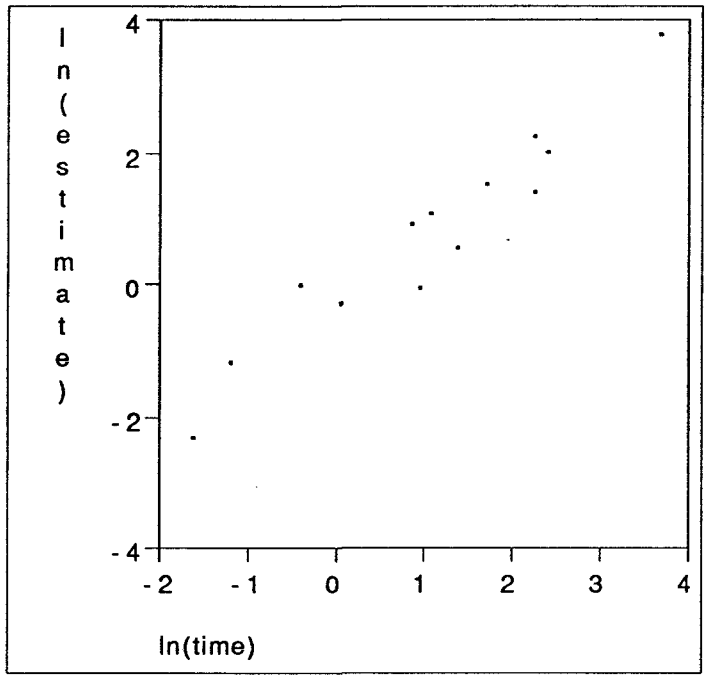


Figure 13. Comparison between the predicted time , ln(estimate), and real computation time, ln(time) for the SPI algorithm.

4 CONCLUSIONS

From the statistical analysis, the predictive models of computation time for four widely used exact algorithms in probabilistic inference in a belief net, -- Conditioning, Clustering, Reduction and SPI, are presented. They give a close estimation of the real time cost. One of the four algorithms, SPI shows better performance than the other algorithms. This is because SPI has the characteristics of query-directed reasoning, caching of intermediate results and incrementality with respect to observations. Its advantages compared with the other algorithms make it more suitable for practical probabilistic inference systems.

5 FURTHER RESEARCH

Probability computation is the most time consuming step in probabilistic inference algorithms. Further research should focus on how to speed up the computation. Factoring the probability space in SPI is one of the good ideas for dealing with this problem. An effective factoring strategy will reduce time cost.

7 REFERENCES

[1] Barlow, R. E. and Pereia, C. A. B. The Bayesian Operation and Probabilistic Inference Diagrams. Nov., 1987, draft book chapter.

[2] Cooper, G. F. Probabilistic inference using belief networks is NP-hard. Knowledge System Laboratory, Stanford University, Memo KSL-87-27, May 1987.

[3] D'Ambrosio, Bruce. Symbolic Probablistic Inference in Belief Nets. Dec. 15, 1989, OSU technical report.

[4] D'Ambrosio, Bruce. Incremental Construction and Evaluation of Defeasible Probabilistic Models. Mar. 13, 1989, to appear in International journal of Approx. reasoning.

[5] Lauritzen, S. L. and Spiegelhalter D. J. Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems. Royal Statistical Society. Jan., 1988.

[6] Pearl J. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Mateo, 1988.

[7] Pearl, J. Fusion, Propagation, and Structuring in Belief Networks. Artificial Intelligence, 29(3): 241-288, 1986.

[8] Rege, A. and Agogino A. M. Topological Framework for Representing and Solving Probabilistic Inference Problems in Expert Systems. IEEE, Nov, 1988.

[9] Shachter, R. D. Evaluating Influence Diagrams. Operation Research, vol. 34 No.34, Nov., 1986.

[10] Shachter, R. D. Evidence Absorption and Propagation Through Evidence reversals. Proceedings of the Fifth Workshop on Uncertainty in AI, 1989.

[11] Srinivas, S. and Breese, J. IDEAL: Influence Diagram Evaluation and Analysis in Lisp. May 8, 1989, Rockwell Palo Alto Lab technical report.

[12] Suermondt, H. J. and Cooper G. F. Probabilistic Inference in Multiply Connected Belief Networks Using Loop Cutsets. Mar., 1989, Stanford technical report.