Journal of Applied Intelligence, 9, 173–184 (1998)
© 1998 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Computational Properties of Two Exact Algorithms for Bayesian Networks

NEVIN LIANWEN ZHANG
Department of Computer Science, University of Science & Technology, Hong Kong, China

Abstract. This paper studies computational properties of two exact inference algorithms for Bayesian networks, namely the clique tree propagation algorithm (CTP) [1] and the variable elimination algorithm (VE). VE permits pruning of nodes irrelevant to a query while CTP facilitates sharing of computations among different queries. Experiments have been conducted to empirically compare VE and CTP. We found that, contrary to common beliefs, VE is often more efficient than CTP, especially in complex networks.

Keywords: Bayesian networks, inference algorithms, computational properties.

1. Introduction

Bayesian networks (BNs)[2] (Pearl 1988) are a knowledge representation framework widely used by AI researchers for reasoning under uncertainty. There is a rich collection of exact and approximate algorithms for inference in BNs. This paper studies computational properties of two exact algorithm, namely clique tree propagation (CTP) and variable elimination (VE).

CTP was developed over a few years by Pearl (1988), Lauritzen and Spiegelhalter (1988), Shafer and Shenoy (1990), and Jensen *et al* (1990). VE is conceptually much simpler than CTP. It has its roots in the work of Shachter (1986, 1988). The underlying ideas are implicit in many papers (e.g. Shenoy 1992 and Shafer 1996). The algorithm was first made explicit by Zhang and Poole (1994) and was extended to exploit independence of causal influence[3] (Heckerman 1993) by Zhang and Poole (1996). Dechter (1996) also calls it bucket elimination.

VE and CTP have different computational features; VE permits pruning of nodes irrelevant to a query (Section 3) while CTP facilitates sharing of computations among different queries (Section 4). The two features are conflicting to each other. If one prunes irrelevant nodes, then one needs to work with different subnetworks of a BN when answering different queries, which precludes computation sharing.

CTP is the most popular exact inference algorithm for BNs. It is widely believed to be the most efficient. Experiments have been conducted with the CPCS networks (Pradhan and Provan 1994) to empirically compare VE and CTP. We found that, contrary to common beliefs, VE is often more efficient than CTP. More specifically, we found that

1. VE takes much less time than or roughly the same time as CTP when computing the posterior probabilities of twenty or less query variables given a set of observations consisting of twenty or less observations.
2. As network complexity increases, VE compares more and more favorably to CTP.
3. There are networks where VE can answer queries in real time while CTP cannot make

any inference at all due to large memory consumption.

On the other hand, we also found that the average performance of CTP increases with the number of observations and the number of query variables given a set of observations, while the average performance of VE decreases with them. Consequently, CTP can be more efficient than VE in situations where there is a large number of observations and there is a large number of query variables given a set of observations. We argue that there are many applications where such situations do not arise. In influence diagram evaluation, for instance, observations correspond to relevant informational predecessors of a decision node and query nodes correspond to components of utility function that are influenced by a decision node (Zhang 1997). There are usually only a few of them[4]. Also if the end user is a human being instead of some other programs, the numbers of observations and query variables should not be large. It is difficult for a human being to manually enter a large number of observations and examine the posterior probabilities of a large number of variables.

We will briefly review CTP and VE, discuss their computational properties, and report on our experiments.

2. Bayesian networks and inference

This section briefly reviews the concept of Bayesian networks and discusses inference in Bayesian networks.

2.1. Bayesian networks

A *Bayesian network* (BN) (Pearl 1988) is an annotated directed acyclic graph, where each node represents a random variable and is attached with a conditional probability of the node given its parents. We will use the terms "nodes" and "variables" interchangeably hereafter.

In addition to the explicitly represented conditional probabilities, a BN also implicitly represents conditional independence assertions. Let x_1, x_2, ..., x_n be an enumeration of all the nodes in the BN such that each node appears after its parents, and let π_{x_i} be the set of parents of a node x_i. The following assertions are implicitly represented:

> For i=1, 2, ... n, x_i is conditionally independent of variables in $\{x_1, x_2, \ldots, x_{i-1}\}\backslash\pi_{x_i}$ given variables in π_{x_i}.

The conditional independence assertions and the conditional probabilities together entail a joint probability over all the variables. As a matter of fact, by the chain rule, we have

$$\begin{aligned} P(x_1, x_2, \ldots, x_n) &= \prod_{i=1}^{n} P(x_i|x_1, x_2, \ldots, x_{i-1}) \\ &= \prod_{i=1}^{n} P(x_i|\pi_{x_i}), \end{aligned} \tag{1}$$

where the second equation is true because of the conditional independence assertions. The conditional probabilities $P(x_i|\pi_{x_i})$ in (1) are given in the specification of the BN.

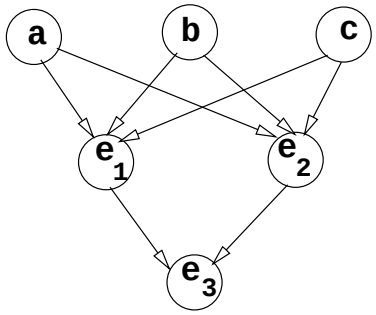
2.2. Inference in Bayesian networks

Inference refers to the process of computing the posterior probability $P(x|Y{=}Y_0)$ of a *query node* x after obtaining some *observations or evidence* $Y{=}Y_0$. Here Y is the list of observed variables and Y_0 is the corresponding list of observed values.

In theory, $P(x|Y{=}Y_0)$ can be obtained from the marginal probability $P(x, Y)$, which in turn can be computed from the joint probability $P(x_1, x_2, \ldots, x_n)$ by summing out variables outside $\{x\}\cup Y$ one by one. In practice, this is not viable because summing out a variable from a joint probability requires an exponential number of additions.

The key to more efficient inference lies in the concept of factorization. A *factorization* of a joint probability is a list of *factors* (non-negative functions of variables) from which one can construct the joint probability. Because of equation (1), a BN represents a factorization of a joint probability. For example, the Bayesian network in Figure **??** factorizes the joint probability $P(a, b, c, e_1, e_2, e_3)$ into the following list of factors:

$$P(a), P(b), P(c), P(e_1|a, b, c), P(e_2|a, b, c), P(e_3|e_1, e_2).$$



The product of the factors is the joint probability.

Suppose a joint probability $P(y_1, y_2, \ldots, y_m)$ is factorized into a list of factors $\mathcal{F}=\{f_1, f_2, \ldots, f_m\}$. While obtaining $P(y_2, \ldots, y_m)$ by summing out y_1 from $P(y_1, y_2, \ldots, y_m)$ requires an exponential number of additions, obtaining a factorization of $P(y_2, \ldots, y_m)$ can be done with much less computation. Both CTP and VE makes use of this fundamental fact. To be specific, suppose the y_1 appears in and only in factors $f_1, f_2, \ldots, f_k$. Then

$$\begin{aligned} P(y_2, \ldots, y_m) &= \sum_{y_1} P(y_1, y_2, \ldots, y_m) \\ &= \sum_{y_1} \prod_{i=1}^{m} f_i \\ &= [\sum_{y_1} \prod_{i=1}^{k} f_i][\prod_{i=k+1}^{m} f_i], \end{aligned} \quad (2)$$

Thus, a factorization of $P(y_2, \ldots, y_m)$ can be obtained by calling the following procedure with y_1 and $\mathcal{F}$.

Procedure `sum-out`$(z, \mathcal{F})$:

- Inputs: $\mathcal{F}$ — a list of factors, z — a variable.
- Output: Another list of factors.

1. Remove from $\mathcal{F}$ all the factors, say $f_1, \ldots, f_k$, that contain z.
2. Add the new factor $\sum_z \prod_{i=1}^{k} f_i$ to $\mathcal{F}$.
3. Return $\mathcal{F}$.

Only variables appearing in the factors that contain z take part in the computation. Those can be only a small portion of all the variables. This is why inference in a BN can be tractable.

3. The variable elimination algorithm

This section reviews the VE algorithm and discusses its computational properties.

3.1. The algorithm

Let $f(x, A)$ be a function of variable x and variables in set A. *Instantiating* x to a particular value α in $f(x, A)$ yields $f(x{=}\alpha, A)$, which is a function of variables in A.

VE computes the posterior probability $P(x|Y{=}Y_0)$ in a BN $\mathcal{N}$ by eliminating nodes outside $\{x\}\cup Y$ one by one.

Procedure `VE`$(x, Y{=}Y_0, \mathcal{N})$

1. Let $\mathcal{F}$ be the set of conditional probabilities in $\mathcal{N}$. Instantiate the observed nodes to their corresponding observed values in all factors in $\mathcal{F}$.
2. Find an ordering ρ of all nodes outside $\{x\}\cup Y$.
3. **While** ρ is not empty,
   (a) Remove the first variable z from ρ,
   (b) $\mathcal{F}=$ `sum-out`$(z, \mathcal{F})$. **Endwhile**
4. Multiply all the factors in $\mathcal{F}$. (The product is a function $f(x)$ of x.)
5. Return $f(x)/\sum_x f(x)$. (Renormalization)

Let Z be the set of unobserved nodes. VE is correct because after step 1, the product of all factors in $\mathcal{F}$ is the probability function $P(Z, Y{=}Y_0)$. By repeatedly using equation (2), one can show that the function $f(x)$ obtained at step 4 is $P(x, Y{=}Y_0)$. Hence $f(x)/\sum_x f(x)$ must be $P(x|Y{=}Y_0)$.

The ordering ρ at step 2 is usually called an *elimination ordering*. It determines the complexity of VE. The problem of finding an optimal elimination ordering is known to be NP-complete (Arnborg *et al* 1987). In practice, two heuristics called minimum deficiency search (Bertelè and Brioschi 1972) and maximum cardinality search (Tarjan and Yannakakis 1982) are

commonly used. See Kjærulff (1990) for empirical comparisons of various heuristics.

3.2. Pruning irrelevant nodes

VE processes one query at a time and hence allows pruning of nodes that are irrelevant to the query. This subsection summarizes work on irrelevant nodes (Shachter 1988, Geiger *et al* 1990, Lauritzen *et al* 1990, and Baker and Boult 1990) and provides new and simple proofs for the results. We will need to deal with more than one BN. For clarity, we will use $P_{\mathcal{N}}(.)$ to refer to probabilities in a BN $\mathcal{N}$.

In a BN $\mathcal{N}$, a node x is an *ancestor* to another node y if there is a directed path from x to y. The *ancestral set* $an(A)$ of a set A of nodes consists of nodes in A and ancestors of those nodes. The following proposition says that nodes outside $an(\{x\}\cup Y)$ are irrelevant to the query $P_{\mathcal{N}}(x|Y{=}Y_0)$.

Proposition 1. *Let $\mathcal{N}_1$ be the BN obtained from a BN $\mathcal{N}$ by pruning nodes outside $an(\{x\}\cup Y)$. Then*

$$P_{\mathcal{N}}(x|Y{=}Y_0) = P_{\mathcal{N}_1}(x|Y{=}Y_0).$$

Proof: It suffices to show that $P_{\mathcal{N}_1}(an(\{x\}\cup Y)) = P_{\mathcal{N}}(an(\{x\}\cup Y))$. Let $x_1, \ldots, x_k$ be an enumeration of all nodes in $an(\{x\}\cup Y)$ such that each node appears after all its parents. Such an enumeration is possible because all the parents of each node in $an(\{x\}\cup Y)$ are also in the set. Then

$$\begin{aligned} P_{\mathcal{N}}(x_1,\ldots,x_k) &= \prod_{i=1}^{k} P_{\mathcal{N}}(x_i|x_1,\ldots,x_{i-1}) \\ &= \prod_{i=1}^{k} P_{\mathcal{N}}(x_i|\pi_{x_i}) \\ &= \prod_{i=1}^{k} P_{\mathcal{N}_1}(x_i|\pi_{x_i}) \\ &= P_{\mathcal{N}_1}(x_1,\ldots,x_k), \qquad (3) \end{aligned}$$

where the second equation is due to the conditional independence assumptions of BNs (see Section 2) and the third equation is due to the fact that the conditional probability of each x_i given its parents is the same in both $\mathcal{N}$ and $\mathcal{N}_1$. The proposition is proved. □

The *moral graph* (Lauritzen and Spiegelhalter 1988) of a BN is obtained by marrying the parents of each node (i.e adding an edge between each pair of parents) and then dropping all directions. Two nodes x and y are *m-separated* by a set A if, in the moral graph, every path connecting them contains at least one node in A. Note that A m-separates each $x{\in}A$ from each $y{\notin}A$. Roughly speaking, the following proposition says that nodes that are m-separated from x by Y are irrelevant to the query $P_{\mathcal{N}_1}(x|Y{=}Y_0)$.

Proposition 2. *Let Z_1 be the set of nodes in a BN $\mathcal{N}_1$ that are m-separated from x by Y, with nodes in Y excluded. Let Z_2 be the set of remaining nodes outside Y. Let Y_1 be the set of nodes in Y that do not have parents in Z_2 and let $Y_2{=}Y\backslash Y_1$. Suppose $\mathcal{N}_2$ is the BN obtained from $\mathcal{N}_1$ by*

1. *Removing all the nodes in Z_1,*
2. *Removing arcs into and conditional probabilities of nodes in Y_1 (thereafter those nodes have no parents), and*
3. *Setting the probabilities of nodes in Y_1 to be the uniform distributions.*

Then

$$P_{\mathcal{N}_1}(x|Y{=}Y_0) = P_{\mathcal{N}_2}(x|Y{=}Y_0).$$

Proof: By definition of the set Z_2, we have $x{\in}Z_2$. Hence it suffices to show that $P_{\mathcal{N}_2}(Z_2,Y_2|Y_1) = P_{\mathcal{N}_1}(Z_2,Y_2|Y_1)$. Let $x_1, \ldots, x_k$ be an anumeration of nodes in $Z_2\cup Y_2$ such that x_j is not an ancestor of x_i if $j > i$. By the definitions of Z_1, Z_2, and Y_2, none of the x_i' have parents in Z_1. Hence, all parents of each x_i must be in the set $\{x_1,\ldots,x_{i-1}\}\cup Y_1$. Consequently,

$$\begin{aligned} P_{\mathcal{N}_1}(x_1,\ldots,x_k|Y_1) &= \prod_{i=1}^{k} P_{\mathcal{N}_1}(x_i|x_1,\ldots,x_{i-1},Y_1) \\ &= \prod_{i=1}^{k} P_{\mathcal{N}_1}(x_i|\pi_{x_i}), \end{aligned}$$

where the second equation is due to the conditional independence assumptions of BN (see Sec-

tion 2). Similarly,

$$P_{\mathcal{N}_2}(x_1, \ldots, x_k|Y_1) = \prod_{i=1}^{k} P_{\mathcal{N}_2}(x_i|\pi_{x_i}).$$

Since the conditional probability of each x_i given its parents is the same in both $\mathcal{N}_1$ and $\mathcal{N}_2$, the proposition follows. □

From now on, we assume that VE always prunes the irrelevant nodes identified by the foregoing propositions before proceeding to answer a query.

4. Clique tree propagation

This section briefly reviews CTP and discusses its computational properties. CTP appears in several basically equivalent variations (Lauritzen and Spiegelhalter 1988, Shafer and Shenoy 1990, Jensen *et al* 1990). The description of CTP given below combines features of the variation by Shafer and Shenoy (1990) and the variation by Jensen *et al* (1990). The reader is referred to Shafer (1996) for a discussion on the subtle differences among the variations.

4.1. Elementary CTP

In a BN, a *clique* is simply a subset of nodes. A *clique tree* is a tree whose nodes are cliques such that if a node of the BN appears in two different cliques, then it also appears in all the cliques on the path between the two cliques.

CTP compiles a BN into a clique tree at a preprocessing step. The clique tree must cover the BN in the sense that for each node in the BN, there must exist at least one clique that contains both the node itself and all its parents. Such a tree can be constructed from an ordering of all nodes in the BN (e.g. Shafer and Shenoy 1990, Zhang 1993). The tree is *initialized* by attaching the conditional probability of each node to *one* clique that contains both the node itself and all its parents. If a clique is attached with more than one factor, the factors are multiplied. If a clique is attached with no factors, the constant factor 1 is attached to it for uniformity. After initialization, each clique is associated with one and only one factor and the product of all factors is the joint probability of the BN.

Let $\mathcal{T}$ be the initialized clique tree. Before computing the posterior probability $P(x|Y{=}Y_0)$ of a node x, CTP absorbs observations into the tree as follows.

Procedure `absorbEvidence`$(Y{=}Y_0, \mathcal{T})$:

- For each observation $y{=}y_0$, create an *evidence factor* $f_{y=y_0}(y)$ that is 1 when $y{=}y_0$ and 0 otherwise. Find one clique that contains y and attach $f_{y=y_0}$ to the clique.

After evidence absorption, some cliques might have more than one factor; the factor created at initialization plus possibly some evidence factors. Let Z be the set of all the unobserved nodes. After evidence absorption, the product of all factors in the clique tree is the probability function $P(Z, Y{=}Y_0)$.

CTP computes posterior probabilities by passing messages around in the clique tree. The messages are factors (i.e. non-negative functions of nodes). Let C and C' be two neighbors cliques. A message can be sent from C to C' as follows after C has received messages from all its other neighbors.

Procedure `sendMessage`(C, C'):

- (Let f_1, ..., f_k be the factors attached to C or sent to C from all its neighors other than C'.) Send the following factor to C':

$$\sum_{C' \setminus C} \prod_{i=1}^{k} f_i,$$

where is the summation is taken over all possible values of the nodes in $C' \setminus C$.

Message passing is organized as follows (Shafer 1996) to obtain the posterior probability $P(x|Y{=}Y_0)$.

Procedure `elemCTP`$(x, Y{=}Y_0, \mathcal{T})$:

1. Find a clique that contains x. (It will be referred to as the *pivot clique* and denoted by C_x.)

2. (Propagation) Each clique C waits until it has received messages from all its neighbors except for the one, denoted by C', that is nearer to the pivot clique than C and then calls `sendMessage`(C, C').
3. (Let $g_1, \ldots, g_l$ be the factors attached to C_x or sent to C_x from all its neighors.) Return $\sum_{C_x \setminus \{x\}} \prod_{i=1}^{l} g_i / \sum_{C_x} \prod_{i=1}^{l} g_i$.

The propagation step starts at *leaf cliques*, i.e. cliques that have only one neighbor, and proceeds toward the pivot clique. In terms of numerical computations, it is similar to the while-loop in `VE`.

To verify the correctness of `elemCTP`, consider the cliques that have not sent out messages at any time during execution. Let Z' be the set of all the unobserved nodes in those cliques. By using equation (2) and the property of clique trees described in the first paragraph of this section[5], one can inductively show that the product of the factors attached to those cliques and the factors sent to them from other cliques is the probability function $P(Z', Y{=}Y_0)$. In particular, the factor $\prod_{i=1}^{l} g_i$ in step 3 is $P(C'_x, Y{=}Y_0)$, where C'_x is the set of unobserved nodes in C_x. Hence the factor returned by `elemCTP` is indeed $P(x|Y{=}Y_0)$.

4.2. Computation sharing in the case of fixed observations

In terms of efficiency, the biggest advantage of CTP is that it allows computation sharing among different queries. This subsection discusses computation sharing among queries with the same observations and the next subsection will extend the discussions to queries with different observations.

Consider computing the posterior probability of two different query nodes x_1 and x_2 given the same observations $Y{=}Y_0$. Let C_{x_1} and C_{x_2} be the pivot cliques `elemCTP` chooses for x_1 and x_2 respectively. Suppose C and C' are two neighboring cliques that are not both on the path between C_{x_1} and C_{x_2}. Then C' is nearer to C_{x_1} than C if and only if it is nearer to C_{x_2} than C. In other words, a message needs to be sent from C to C' when computing $P(x_1|Y{=}Y_0)$ if and only if a message needs to be sent from C to C' when computing $P(x_2|Y{=}Y_0)$. The contents of the two messages are the same[6] and hence can be shared.

To materialize the opportunities of computation sharing, create, at each clique, a *port* for each of its neighbors to store the message from that neighbor (Shafer and Shenoy 1990). For any two neighboring cliques C and C', use $port(C, C')$ to denote the port of C for C'. It is initially empty. Modify `elemCTP` as follows.

Procedure `CTP`$(x, Y{=}Y_0, \mathcal{T})$:

1. Same as in `elemCTP`.
2. For each neighbor C of C_x, `collectMessage`(C_x, C).

Subroutine `collectMessage`(C_x, C):

- If $port(C_x, C)$ is empty,
  (a) For each neighbor C' of C other than C_x, `collectMessage`(C, C').
  (b) `sendMessage`(C, C_x).
  (c) Store the message at port(C_x, C).

3. Same as in `elemCTP`.

Note that if $port(C_x, C)$ is not empty, i.e. if C has sent a message to C_x earlier when processing other query nodes, `collectMessage` does nothing. The earlier message is reused in the current computation. The `collectMessage` subroutine is similar to the `collectEvidence` procedure of Jensen *et al* (1990).

4.3. Computation sharing in the case of changing observations

There are practical situations where one needs to add or delete observations after posterior probabilities have been obtained. CTP allows some messages computed before the change in observations be reused after the change in observations.

Addition of observations implies introduction of new evidence factors to cliques and deletion of observations implies removal of evidence factors currently associated with cliques. They both affect contents of some (but not all) messages.

Suppose we have a new observation $z{=}z_0$. To absorb it, we need to add the evidence factor $f_{z=z_0}(z)$ to a clique C_z that contains z. Let C

and C' be two neighboring cliques. Consider the messages between them before and after absorbing the observation. Suppose C is nearer to C_z than C'. From `collectMessage`, we see that the message from C to C' is affected by the new evidence factor, while the message from C' to C is not. In other words, the message from C' to C can be reused after absorption of the new observation while the message from C to C' cannot.

The following procedure absorbs the new observation $z{=}z_0$ into the clique tree $\mathcal{T}$, deletes messages that cannot be reused, and keeps messages that can.

Procedure `addEvidence`($z{=}z_0, \mathcal{T}$):

1. Find a clique C_z that contains z and add the evidence factor $f_{z=z_0}(z)$ to C_z.
2. For each neighbor C of C_z, `cleanMessage`(C_z, C).

Subroutine `cleanMessage`(C_z, C):

- If $port(C, C_z)$ is not empty,
  (a) Clean the message stored at $port(C, C_z)$. (Afterwards the port is empty).
  (b) For each neighbor C' of C other than C_z, `cleanMessage`(C, C').

After applying `addEvidence`, `CTP` can be called to compute new posterior probabilities. Messages that are not deleted by `addEvidence` are reused. As an example, suppose $P(x|Y{=}Y_0)$ has been computed previously by `CTP`. Suppose we compute the new posterior probability $P(x|Y{=}Y_0, z{=}z_0)$ by first applying `addEvidence`($z{=}z_0, \mathcal{T}$) and then calling `CTP`. Then only the messages along the path from C_z to C_x are recomputed. All other necessary messages have been computed earlier and are recycled in the current computation.

Deletion of observations can be handled in a similar way. An observation $z{=}z_0$ can be deleted by using the following procedure.

Procedure `deleteEvidence`($z{=}z_0, \mathcal{T}$):

1. Find the clique C_z that is associated with the evidence factor $f_{z=z_0}(z)$. Remove the factor from C_z.
2. For each neighbor C of C_z, `cleanMessage`(C_z, C).

Afterwards, `CTP` can be used to compute new posterior probabilities. Again messages that are not deleted by `deleteEvidence` are reused.

5. Empirical comparisons

VE and CTP have different computational properties; VE permits pruning of irrelevant nodes while CTP facilitates computation sharing. Experiments have been conducted with the CPCS networks (Pradhan and Provan 1994) to empirically compare VE and CTP. This section reports on the experiments. Before diving into the details, here is a summary of our findings:

1. VE takes much less time than or roughly the same time as CTP when computing the posterior probabilities twenty or less query nodes given a set of observations with twenty or less observations.
2. As network complexity increases, VE compares more and more favorably to CTP.
3. There are networks where VE can answer queries in real time while CTP cannot make any inference at all due to its large appetite for memory.
4. The average performance of CTP increases with the number of observations and the number of query nodes given a set of observations, while the average performance of VE decreases with them.

5.1. The CPCS networks

The CPCS networks are multilevel, multivalued BNs for medicine. They are created by Pradhan and Provan (1994) based on the Computer-based Patient Case Simulation system (CPCS-PM) developed by Parker and Miller (1987). The networks vary in the number of nodes (NN), and the average number of parents of a node (ANP),

and the average number of possible values of a node (ANV). Four networks were used in our experiments[7]. Their complexity attributes are summarized in the following table.

Networks	NN	ANP	ANV
Network 1	145	1.14	2.27
Network 2	245	1.45	2.0
Network 3	245	1.45	2.25
Network 4	364	2.01	2.0

5.2. Experiment setups

Two experiments were conducted. The first experiment compares VE and CTP in situations where one needs to compute the posterior probabilities of 5, 10, or 20 nodes given a fixed set of observation consisting of 5, 10, or 20 observations. There are nine combinations (situations). For each situation (n, m), where n and m are the numbers of observations and query nodes respectively, a large number (300 for Networks 1 and 2, 100 for Network 3) of sets of observations consisting of n observations were randomly generated. For each set of observations, m unobserved nodes were randomly selected to be query nodes. VE and CTP were called to compute the posterior probability of each of the m nodes given the set of observations and the total times VE and CTP took were recorded. The performances of VE and CTP are compared in terms of the averages of the total times across the sets of observations.

5.3. Results for Network 1

The second experiment compares VE and CTP in the case of changing observations. The sets of observations and the query nodes used were the same as in the first experiment, except each set of observation was modified by replacing one of its member, which was randomly selected, with a randomly generated new observation. VE and CTP were called to computed the new posterior probabilities of the query nodes. In the case of CTP, `addEvidence` and `deleteEvidence` were first applied. While VE is expected to take the same as in the first experiment on average, CTP is expected to take less time since some messages computed before the change of observations are reused.

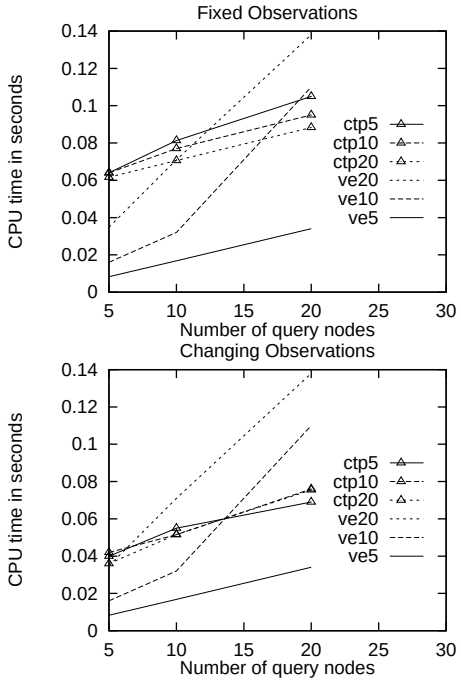


Fig. 1. Statistics for Network 1. The curve "ve5" , for instance, depicts that the average times VE took in the three situations with 5 observations and 5, 10, or 20 query variables.

For fairness, subroutines were shared whenever possible in the implementations of VE and CTP. The elimination orderings required by VE and needed in the construction of a clique tree were generated using the same heuristic, namely minimum deficiency search (Bertelè and Brioschi 1972). All experiments were carried out on a SUN SPARC1000 machine.

Statistics for Network 1 are shown in Figure 1. In the case of fixed observations, VE took less time than CTP in all the situations expect for situations (10, 20) and (20, 20). In particular, VE was significantly more efficient than CTP in the situations where there are 5 query nodes or 5 observations. In situation (5, 5), VE was about 8 times faster than CTP. In situation (10, 20), VE was slightly less efficient than CTP and in situation (20, 20), CTP was 1.6 times faster than VE.

In the case of changing observations, VE was more efficient than CTP in situations (5, 5), (5,

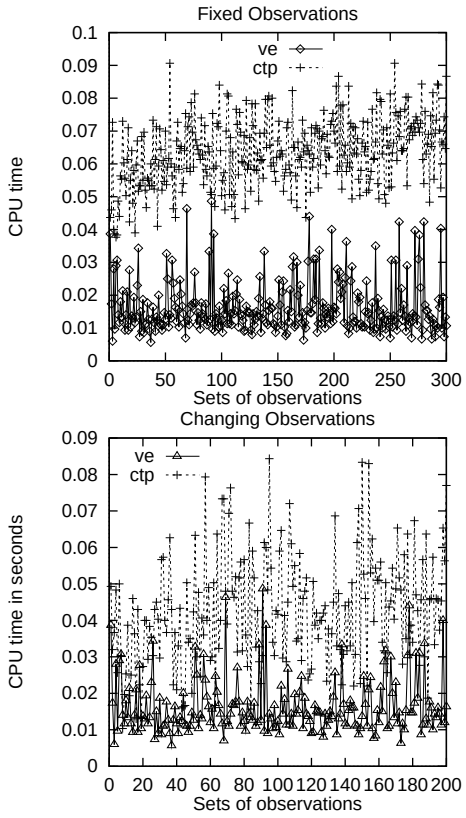


Fig. 2. Variances of computation times across sets of observations for situation (10, 5).

10), (5, 20), (10, 5), and (10, 10). In situation (5, 5), VE was about 5 times faster than CTP. In situation (20, 5), VE and CTP took roughly the same time. CTP was about 1.4 times faster than VE in situations (10, 20) and (20, 10), and about 2 times faster in situation (20, 20).

Data shown in Figure 1 are computation times averaged across a large number of sets of observations. To give the reader a feeling about the variances, Figure 2 depicts the computation times for situation (10, 5) as a function of sets of observation. We see that the variances for CTP are larger in the case of changing observations than in the case of fixed observation. This is due to the fact that there are two more sources of randomness in the former case, namely deletion of a randomly selected existing observation and addition of a randomly generated new observation. They both affect the amount of computations that can

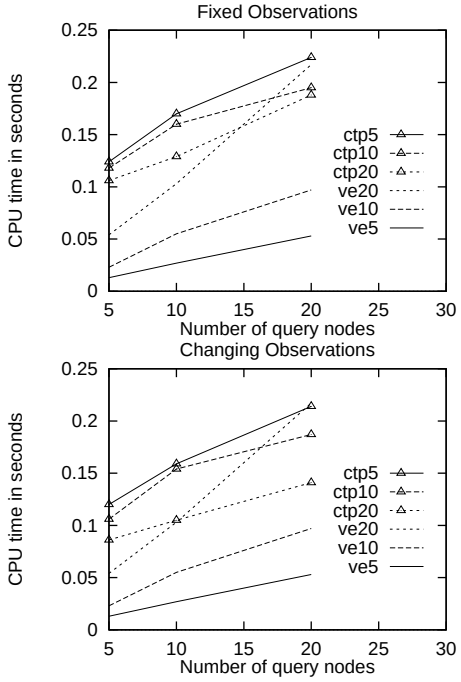


Fig. 3. Statistics for Network 2.

be shared before and after the change of observations.

5.4. *Results for Networks 2 and 3*

Statistics for Network 2 are shown in Figure 3. In both the case of fixed observations and the case of changing observations, VE took less time than CTP in all the situations expect for situation (20, 20). In particular, VE was significantly more efficient than CTP in the situations where there are 5 query nodes or 5 observations. In situation (5, 5), VE was about 10 times faster than CTP. In situation (20, 20), VE was slightly less efficient than CTP.

Statistics for Network 3 are shown in Figure 4. In both the case of fixed observations and the case of changing observations, VE took less time than CTP in all situations. The differences are significant in situations where there are 5 or 10 query nodes. In situation (5, 5), VE was about 50 times faster than CTP in the case of fixed observations and about 40 times faster in the case of changing observations.

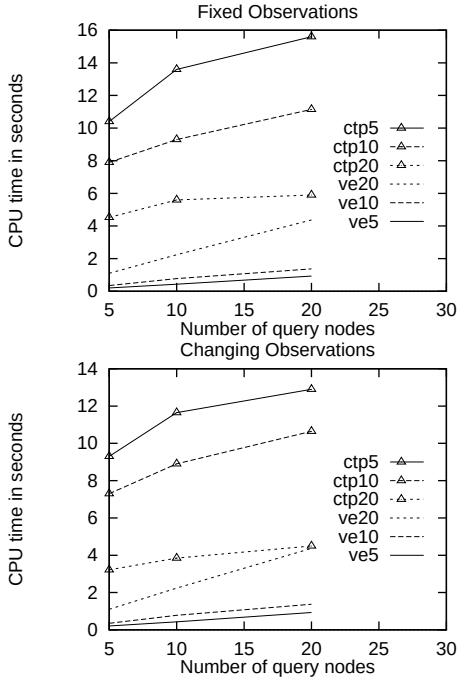


Fig. 4. Statistics for Network 3.

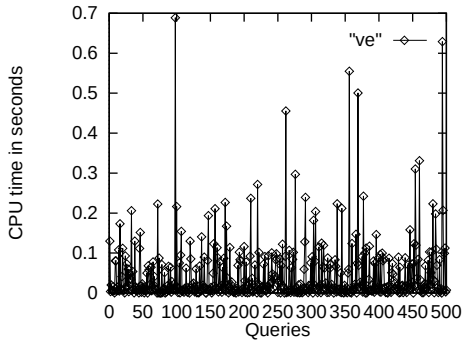


Fig. 5. Performance of VE in Network 4.

Network 2 is more complex than Network 1 because it has more nodes and the nodes have more parents. Network 3 is more complex than Network 2 because nodes can take more possible values. The differences between the performances of VE and CTP are larger in Network 2 than in Network 1 and are even larger in Network 3. Those suggest that VE compares more and more favorably to CTP as network complexity increases.

5.5. Results for Network 4

Network 4 has so far been ignored in the comparisons. The reason is that we were unable to collect any data for CTP in the network. The computer ran out of memory (200 megabytes) when initializing the clique tree for the network.

CTP requires much memory than VE for two reasons. First, it does not permits pruning of irrelevant nodes. Second, it stores intermediate computational results to facilitate computation sharing[8]. The big appetite for memory of CTP limits its usefulness in complex networks such as Network 4.

To demonstrate the effectiveness of VE in Network 4, we randomly generated 500 queries with 20 or less observations. The times VE took to answer the queries are shown in Figure 5. We see that VE was able to answer each of the queries in less than 1 second CPU time. The average time it took is 0.04 seconds CPU time.

5.6. Other interesting facts

A couple of other interesting facts emerge from the results presented above. First, it is always the case that the curve "ve20" lies above "ve10" and "ve10" lies above "ve5". On the other hand, the curve "ctp20" always lies below "ctp10" and "ctp10" always lies below "ctp5", except in the left plot of Figure 1. (This exception is due to randomness.) Those observations suggest that the amount of time VE took increases with the number of observations while the amount of time CTP took decreases with it. This is expected for the following reasons. As we have learned in Section 3, to compute $P(x|Y{=}Y_0)$, VE needs to work with a subnetwork consisting of nodes in the ancestral set $an(\{x\}\cup Y)$ that are not m-separated from x by Y. The size of the ancestral set increases with the number of observations and hence so does the amount of time VE takes. In CTP, on the other hand, more observations means more instantiated variables in the message factors, which in turn implies CTP takes less time.

Second, while both the amounts of time VE and CTP took increase with the number of query nodes, the former increases faster than the latter. This is due to computation sharing. In VE, no

computations are shared among different queries. The expected time for answering the next randomly generated query remains the same regardless of the number of queries processed previously. In CTP, on the other hand, some computations carried out when answering earlier queries are shared when answering later query nodes. Hence, the expected time for answering the next randomly generated query decreases with the number of previous queries.

To summarize, the average performance of CTP increases with the number of observations and the number of query nodes given a set of observations, while the average performance of VE decreases with them.

6. Conclusions

We study computational properties of two exact inference algorithms for BNs, namely VE and CTP. VE allows pruning of nodes irrelevant to a query while CTP facilitates sharing of computations among different queries. These two computational features are conflicting to each other; pruning of irrelevant nodes implies that one needs to work with different subnetworks of a BN when answering different queries, which precludes computation sharing.

Experiments have carried out to empirically compare VE and CTP. We have a number of findings. First, VE is more efficient than CTP when the number of observations and the number of query nodes given a set of observations are not large. Second, VE compares more and more favorably to CTP as network complexity increases. Third, there are networks where VE can answer queries in real time while CTP cannot make any inference at all due to its large appetite for memory. Finally, the average performance of CTP increases with the number of observations and the number of query nodes given a set of observations, while the average performance of VE decrease with them. Those findings can help practitioners to make the right choice between VE and CTP for their applications.

Acknowledgements

Research was supported by Hong Kong Research Council under grant HKUST658/95E and by Sino Software Research Institute under grant SSRC95/96.EG01 . Tak Yin Chan and Li Yan helped with some of the implementations.

Nevin L. Zhang is an assistant professor at Hong Kong University of Science and Technology. Dr Zhang's research interests include Bayesian belief network, decision networks, and Markov decision processes, and their applications to planning, learning, data mining, and information retrieval. He has received PhD degree in Applied Mathematics from Beijing Normal University and PhD degree in Computer Science from University of British Columbia.

Notes

1. Also known as join tree propagation and junction tree propagation.
2. Also known as belief networks, probabilistic inference diagrams, and causal probabilistic networks.
3. Also known causal independence.
4. In influence diagrams, one wants to compute an optimal decision rule for each decision node. A decision rule for a decision node is a mapping from all its relevant informational predecessors to itself. The size of the decision rule is exponential in the number of relevant informational predecessors. Consequently, there can only be a few relevant informational predecessors because otherwise representing the decision rule itself would be difficulty, to say nothing of computing it.
5. This property guarantees that when computing the message $\sum_{C' \setminus C} \prod_{i=1}^{k} f_i$, the nodes in $C' \setminus C$ appear only in the f_i's. They do not appear in any other factors in the clique tree.
6. The claim can be proved by induction. First, the claim is certainly true when C is a leaf clique; the contents of both messages are $\sum_{C \setminus C'} \prod_{i=1}^{k} f_i$, where the f_i's are the factors associated with C. As the induction hypothesis, assume all the messages sent to C from all its neighbors other than C' have the same contents both when processing x_1 and x_2. Then the contents of the two messages from C to C' are both $\sum_{C \setminus C'} \prod_{i=1}^{k} f_i$, where the f_i's are the factors associated with C or sent to C from all its neighbors other than C'.

7. In the networks, conditional probabilities for some of the nodes are represented as several pieces because of independence of causal independence. Independence of causal independence was ignored in our experiments; The conditional probabilities were first constructed from the pieces.
8. In the case of fixed observations, this demand for memory space can be reduced; After sending messages to all its neighbors, a clique can combine all its incoming messages and its associated factors into one factor and then delete those messages and factors. It is justified to do so because, after a clique has sent out messages to all its neighbors, we need to deal with the clique only when we want to compute the posterior probability of a node in the clique, which can be obtained from the combined factor. Jensen *et al* (1990) completely eliminate this memory demand for the case of changing observations. However, their method can only handle addition of new observations. All messages have to be recomputed if an existing observation is deleted. Even with the aforementioned optimizations, CTP still faces memory problems when dealing complex networks. In our experiments, the memory consumption exceeded 200 megabytes when initializing the clique tree for Network 4, before propagation could take place.

References

1. S. Arnborg, D. G. Corneil, A. Proskurowski (1987), Complexity of finding embedding in a k-tree, *SIAM J. Alg. Disc. Meth.*, 8(2), 277-284.
2. M. Baker and T. E. Boult (1990), Pruning Bayesian networks for efficient computation, in *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July, Cambridge, Mass. , pp. 257 - 264.
3. R. Dechter (1996), Bucket elimination: A unifying framework for probabilistic inference, *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pp. 211-219.
4. D. Geiger, T. Verma, and J. Pearl (1990), *d*-separation: From theorems to algorithms, in *Uncertainty in Artificial Intelligence* **5**, pp. 139-148.
5. D. Heckerman (1993), Causal independence for knowledge acquisition and inference, in *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 122-127.
6. F. V. Jensen, K. G. Olesen, and K. Anderson (1990), An algebra of Bayesian belief universes for knowledge-based systems, *Networks*, 20, pp. 637 - 659.
7. U. Kjærulff (1994), Reduction of computational complexity in Bayesian networks through removal of weak dependences. in Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, pp. 374-382.
8. S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H. G. Leimer (1990), Independence Properties of Directed Markov Fields, *Networks*, 20, pp. 491-506.
9. S. L. Lauritzen and D. J. Spiegelhalter (1988), Local computations with probabilities on graphical structures and their applications to expert systems, *Journal of Royal Statistical Society B*, 50: 2, pp. 157 - 224.
10. J. Pearl (1988), *Probabilistic Reasoning in Intelligence Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, Los Altos, CA.
11. M. Pradhan, G. Provan, B. Middleton, and M. Henrion (1994), Knowledge engineering for large belief networks, in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 484-490.
12. G. Shafer and P. Shenoy (1990), Probability propagation, *Annals of Mathematics and Artificial Intelligence*, **2**, pp. 327-352.
13. G. Shafer (1996), *Probabilistic Expert Systems*, SIAM.
14. R. Shachter (1986), Evaluating Influence Diagrams, *Operations Research*, 34, pp. 871-882
15. R. Shachter (1988), Probabilistic Inference and Influence Diagrams, *Operations Research*, 36, pp. 589-605.
16. P. P. Shenoy, (1992), Valuation-Based Systems for Bayesian Decision Analysis, *Operations research*, 40, No. 3, pp. 463-484.
17. R. E. Tarjan and M. Yannakakis (1984), Simple linear time algorithm to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM J. Comput.*, 13, pp. 566-579.
18. L. Zhang (1993), Studies on hypergraphs (I): Hyperforests, *Discrete Applied Mathematics*, 42, pp. 95-112.
19. N. L. Zhang and D. Poole (1994), A simple approach to Bayesian network computations, in *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pp. 171-178.
20. N.L. Zhang (1997), Probabilistic inference in inference diagrams, *Computational Intelligence*, to appear.