A Clique Tree Algorithm Exploiting Context Specific Independence

by

Leslie Tung

B.Sc., University of British Columbia, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
~~to t~~he required standard

The University of British Columbia

August 2002

© Leslie Tung, 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date August 14, 2002

Abstract

Context specific independence can provide compact representation of the conditional probabilities in Bayesian networks when some variables are only relevant in specific contexts. We present *cve-tree*, an algorithm that exploits context specific independence in clique tree propagation. This algorithm is based on a query-based contextual variable elimination algorithm (*cve*) that eliminates in turn the variables not needed in an answer. We extend *cve* to producing the posterior probabilities of all variables efficiently and allow the incremental addition of evidence. We perform experiments that compare *cve-tree* and *Hugin* using parameterized random networks that exhibit various amounts of context specific independence, as well as a standard network, the *Insurance* network. Our empirical results show that *cve-tree* is efficient, both in time and in space, as compared to the *Hugin* architecture, on computing posterior probabilities for Bayesian networks that exhibit context specific independence.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor, David Poole, for his ideas, comments and guidance. I would also like to thank my parents for their encouragement and support. Without them, my studies and research would not have been possible.

LESLIE TUNG

The University of British Columbia
August 2002

Chapter 1

Introduction

1.1 Bayesian Network

Reasoning under uncertainty is an important issue in artificial intelligence because human and computer agents must make decisions on what actions to take without knowing the exact state of the world and without being able to precisely predict the outcomes of actions.

In situations in which causality plays a role, Bayesian networks (Pearl, 1988) allow us to model the situations probabilistically. Bayesian networks provide a graphical representation of the dependencies and independencies among the variables.

A Bayesian network $\mathcal{N}$ is a triplet (V, A, P) where

- V is a set of variables.
- A is a set of arcs, which together with V constitutes a directed acyclic graph $G = (V, A)$.
- $P = P(v|\pi_v) : v \in V$, where π_v is the set of parents of v, where the parents of

v is the set of variables such that v is independent of all other variables given the parents of v. That is, P is the set of conditional probabilities of all the variables given their respective parents. If a variable has no parents, then π_v is empty and $P(v|\pi_v)$ is simply the prior probability of v.

The arcs in a Bayesian network represent direct influence between the random variables. The independency encoded in a Bayesian network is that each variable is independent of its non-descendants given its parents.

Figure 1.1 shows a Bayesian network of 6 variables, along with the conditional probability tables for each variable. The domain of all variables is Boolean.

By conditional independence, the prior joint probability of a Bayesian network can be obtained by

$$P(V) = \prod_{v \in V} P(v|\pi_v)$$

and for any subset X of V, the marginal probability $P(X)$ is obtained by

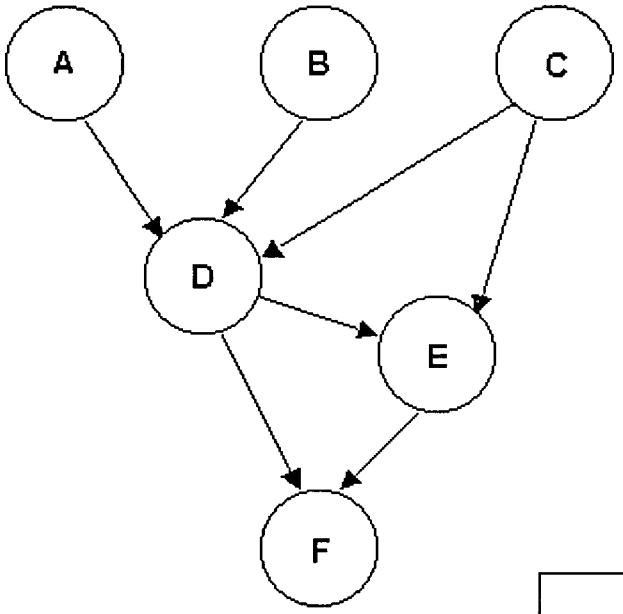
$$P(X) = \sum_{V-X} \prod_{v \in V} P(v|\pi_v)$$

If $Y \subseteq V$ is the set of variables observed to have specific values Y_0 and $X \subseteq V$ is the set of variables whose joint probability is of interest, then the posterior probability $P(X|Y = Y_0)$ can be obtained by

$$P(X|Y = Y_0) = \frac{P(X \wedge Y = Y_0)}{P(Y = Y_0)}$$

1.2 Probability Inference

The most common task we wish to solve using Bayesian networks is probabilistic inference. Probabilistic inference is the process of computing the posterior probability $P(X|Y = Y_0)$, where X is a set of one or more query variables and Y is a set of observed variables with corresponding observed values Y_0.



P(A)	
true	false
0.4	0.6

P(B)	
true	false
0.2	0.8

P(C)	
true	false
0.3	0.7

C	D	P(E | C, D) true	false
true	true	0.8	0.2
true	false	0.1	0.9
false	true	0.8	0.2
false	false	0.3	0.7

A	B	C	P(D | A, B, C) true	false
true	true	true	0.7	0.3
true	true	false	0.7	0.3
true	false	true	0.7	0.3
true	false	false	0.7	0.3
false	true	true	0.2	0.8
false	true	false	0.4	0.6
false	false	true	0.2	0.8
false	false	false	0.9	0.1

D	E	P(F | D, E) true	false
true	true	0.4	0.6
true	false	0.7	0.3
false	true	0.4	0.6
false	false	0.8	0.2

Figure 1.1: Example Network #1

The computation of a variable's posterior probability given evidence in a Bayesian network is in general NP-hard (Cooper, 1987). However, there exist many architectures and algorithms for computing all the exact posterior probabilities in a Bayesian network efficiently for many networks, including Lauritzen and Spiegelhalter (1988), Shafer and Shenoy (1990) and Jensen, Lauritzen and Olesen (1990). These algorithms are based on a secondary structure called a clique tree (or join tree or junction tree), which is built by triangulating a Bayesian network's moralized graph. Other algorithms, such as bucket elimination (Dechter, 1996) and variable elimination (Zhang and Poole, 1994) use a query-based approach to calculate the posterior probabilities of variables as they are demanded.

The definition of a Bayesian network only captures the independence relations among variables, that is, independencies that hold for any assignment of values to the variables. However, it does not constrain how a variable depends on its parents. There is often some structure in the conditional probability tables, such as causal independence (Heckerman and Breese, 1994; Zhang and Poole, 1996; Zhang and Yan, 1997) and context specific independence (Boutilier, Friedman, Goldszmidt and Koller, 1996; Geiger and Heckerman, 1996; Zhang, 1998; Zhang and Poole, 1999; Poole and Zhang, 2002), that can be exploited to improve the time and space required in probabilistic inference.

Causal independence refers to the situation where multiple causes contribute independently to a common effect. With causal independence, the probability function can be described using a binary operator that can be applied to values from each of the parent variables.

Context specific independence refers to variable dependencies that depend on particular values of parent variables. Intuitively, in the network in Figure 1.1, the

regularities of the conditional probability table of D ensure that D is independent of B and C given the context $A = true$, but is dependent on B and C in the context $A = false$.

We propose an algorithm called *cve-tree* that exploits context specific independence using the clique tree structure to compute all the posterior probabilities in a Bayesian network. Our experiments show that *cve-tree* is more efficient both in time and space than *Hugin* (Lauritzen and Spiegelhalter, 1988) when some context specific independence is present in the Bayesian network.

1.3 Organization

We first present definitions of context specific independence and describe some related works in the research of exploiting context specific independence in Chapter 2. Chapter 3 presents some algorithms for probabilistic inference that are related to our proposed algorithm, *cve-tree*. The *cve-tree* algorithm is discussed in detail in Chapter 4. Chapter 5 gives some empirical evaluation of *cve-tree* using standard and random networks that contain various amounts of context specific independence. Chapter 6 presents conclusions and future work.

Chapter 2

Context Specific Independence

2.1 Definitions

In this section, we give some definitions for context specific independence. We also define generalized rules which we will use to capture context specific independence and facilitate probabilistic inference exploiting context specific independence. These definitions are based on Boutilier et al. (1996) and Poole and Zhang (2002).

Definition 1 Given a set of variables C, a *context* on C is an assignment of one value to each variable in C. Two contexts are *incompatible* if there exists a variable that is assigned different values in them; otherwise, they are *compatible.* Context c_1 on C_1 is a *subcontext* of context c_2 on C_2 if $C_1 \subseteq C_2$ and c_1 and c_2 are compatible. An *empty context*, denoted $TRUE$, occurs when C is an empty set.

For example, the context $A = true \wedge B = false$ is compatible with the context $A = true \wedge C = true$, and is incompatible with the context $A = false \wedge C = true$.

Definition 2 Let X, Y, and C be sets of variables. X and Y are *contextually independent given context* $C = c$, where $c \in domain(C)$, if $P(X|Y = y_1 \wedge C = c) =$

$P(X|Y = y_2 \wedge C = c)$ for all $y_1, y_2 \in domain(Y)$ such that $P(Y = y_1 \wedge C = c_1) > 0$ and $P(Y = y_2 \wedge C = c_1) > 0$.

In the network in Figure 1.1, variable F is contextually independent of variable D given the context $E = true$, since $P(F = true|E = true) = 0.4$ and $P(F = false|E = true) = 0.6$ regardless of the value of D. On the other hand, if $E = false$, then the probability of F would depend on the value of D.

Definition 3 Suppose we have a total ordering of the variables $X_1, \ldots, X_n$ of a Bayesian network such that all parents of a variable precede the variable. Given variable X_i, we say that $C = c$, where $C \subseteq \{X_1, \ldots, X_{i-1}\}$ and $c \in domain(C)$, is a *parent context* for X_i if X_i is contextually independent of the predecessors of X_i (namely, $\{X_1, \ldots, X_{i-1}\}$) given $C = c$.

Definition 4 A *mutually exclusive and exhaustive set of parent contexts* for variable X_i is a set of parent contexts for X_i such that, for every context C on the parents of X_i, there is exactly one element of the mutually exclusive and exhaustive set of parent contexts that is compatible with C.

A contextual belief network has the same graphical structure as a belief network, but each node X_i in the network is associated with a mutually exclusive and exhaustive set of minimal parent contexts, Π_i, and, for each $\pi \in \Pi_i$, a probability distribution $P(X_i|\pi)$ on X_i. Instead of having a conditional probability table for a node, each node has a set of generalized rules whose contexts form a mutually exclusive and exhaustive set of minimal parent contexts.

Definition 5 A *generalized rule* is of the form $\langle c, t \rangle$, where c is a context, say, $X_1 = v_1 \wedge \ldots \wedge X_k = v_k$, and t is a table that represents a function on variables

$X_{k+1}, \ldots, X_m$. $t[v_{k+1}, \ldots, v_m]$ is a number that represents the contribution of the context $X_1 = v_1 \wedge \ldots \wedge X_k = v_k \wedge X_{k+1} = v_{k+1} \wedge \ldots \wedge X_m = v_m$. For this thesis, generalized rules are simply called rules.

The rule

$\langle A = false \wedge C = false,$

B	D	$Value$
$true$	$true$	0.4
$true$	$false$	0.6
$false$	$true$	0.9
$false$	$false$	0.1

$\rangle$

represents the part of the conditional probability table of $P(D|A, B, C)$ in Figure 1.1 with $A = false$ and $C = false$.

The probability table of $P(D|A, B, C)$ in Figure 1.1 can be represented by rules with contexts $A = true$, $A = false \wedge C = true$, $A = false \wedge C = false \wedge B = true$, and $A = false \wedge C = false \wedge B = false$. These contexts form a mutually exclusive and exhaustive set of parent contexts for variable D because for any truth value assignment on the parent variables A, B, and C, the assignment is compatible with exactly one of the four contexts.

The rule base of the network is the union of the rules in all the nodes of the network. Figure 2.1 lists the rule base of the Example Network #1 in Figure 1.1. With context specific independence, some of the larger tables are collapsed into a number of rules with smaller tables. For example the probability table $P(D|A, B, C)$ without context specific independence encoded has 16 entries. This table is split into 3 rules of table size 2, 2, and 4, for a total of 8 entries. Notice that the two parent contexts for D, $A = false \wedge C = false \wedge B = true$ and $A = false \wedge C = false \wedge B = false$, are combined into one rule in the rule base because no savings can be obtained by maintaining them as two separate rules. The amount of space savings can be

$\langle TRUE,$

A	$Value$
$true$	0.4
$false$	0.6

$\rangle,$ $\langle TRUE,$

B	$Value$
$true$	0.2
$false$	0.8

$\rangle,$

$\langle TRUE,$

C	$Value$
$true$	0.3
$false$	0.7

$\rangle,$ $\langle A = true,$

D	$Value$
$true$	0.7
$false$	0.3

$\rangle,$

$\langle A = false \wedge C = true,$

D	$Value$
$true$	0.2
$false$	0.8

$\rangle,$

$\langle A = false \wedge C = false,$

B	D	$Value$
$true$	$true$	0.4
$true$	$false$	0.6
$false$	$true$	0.9
$false$	$false$	0.1

$\rangle,$

$\langle D = true,$

E	$Value$
$true$	0.8
$false$	0.2

$\rangle,$ $\langle D = false,$

C	E	$Value$
$true$	$true$	0.1
$true$	$false$	0.9
$false$	$true$	0.3
$false$	$false$	0.7

$\rangle,$

$\langle E = true,$

F	$Value$
$true$	0.4
$false$	0.6

$\rangle,$ $\langle E = false,$

D	F	$Value$
$true$	$true$	0.7
$true$	$false$	0.3
$false$	$true$	0.8
$false$	$false$	0.2

$\rangle$

Figure 2.1: Rule base for Example Network #1

significant when a variable has many parents, but does not depend on all of them in some contexts.

2.2 Past Work on Context Specific Independence

There are a few different ways in the literature that make use of the additional structure of context specific independence to improve probabilistic inference.

Boutilier et al. (1996) presents two algorithms to exploit context specific independence in a Bayesian network. The first is network transformation and clustering. With this method, auxiliary variables are introduced into the original network such that many of the context specific independencies are qualitatively encoded within the network structure of the transformed network. The transformation of the Bayesian network makes use of the structure of CPT-trees, which are the conditional probability tables of the Bayesian network represented as decision trees. Computational savings can be achieved with transformations that reduce the overall number of values present in a family (that is, a node and its parents) using clustering algorithms such as *Hugin* (Lauritzen and Spiegelhalter, 1988).

The other algorithm suggested by Boutilier et al. (1996) is a form of cutset conditioning. The cutset algorithm for probability inference works by selecting a set of variables (called a cutset) that, once instantiated, makes the network singly connected. Each variable in the cutset is instantiated with a value as evidence and inference is performed on the resulting network. This is done using reasoning by cases, where each case is a possible assignment to the variables in the cutset. The results of inference for all cases are combined to give the final answer to the query. The running time of the cutset algorithm is determined by the number of cases, that is, the total number of values for the cutset variables. With context specific

independence, instantiating a particular variable in the cutset to a certain value in order to cut a loop may render other arcs vacuous. This may cut additional loops without instantiating additional variables in the cutset. As a result, the number of cases may be reduced and so the number of times that inference must be performed is reduced, speeding up the total inference time for queries.

Geiger and Heckerman (1996) presents another method to exploit context specific independence. With the notion of similarity networks, context specific independencies are made explicit in the graphical structure of a Bayesian network. In a similarity network, the edges between the variables can be thought of as a network whose edges can appear or disappear depending on the values of certain variables in the network. This allows for different Bayesian networks to perform inference for different contexts.

Poole and Zhang (2002) presents a rule-based contextual variable elimination algorithm. This algorithm will be discussed in detail in Section 3.4.

Chapter 3

Algorithms for Probabilistic Inference

3.1 Variable Elimination

The variable elimination (*ve*) algorithm (Zhang and Poole, 1994) is a query-oriented algorithm for probabilistic inference which does not make use of any secondary structure and works on factors representing conditional probabilities.

The *ve* algorithm works as follows: Suppose a Bayesian network consists of variables $X_1, \ldots, X_n$ and suppose the variables $E_1, \ldots, E_s$ are observed to have values $e_1, \ldots, e_s$, respectively, and the conditional probability of variable X is queried given the observations. We have

$$P(X|E_1 = e_1 \wedge \ldots \wedge E_s = e_s) = \frac{P(X \wedge E_1 = e_1 \wedge \ldots \wedge E_s = e_s)}{P(E_1 = e_1 \wedge \ldots \wedge E_s = e_s)}$$

where the denominator is a normalizing factor:

$$P(E_1 = e_1 \wedge \ldots \wedge E_s = e_s) = \sum_{v \in domain(X)} P(X = v \wedge E_1 = e_1 \wedge \ldots \wedge E_s = e_s)$$

and the numerator can be calculated by summing out the non-query, non-observed variables Y, where $Y = \{Y_1, \ldots, Y_k\} = \{X_1, \ldots, X_n\} - \{X\} - \{E_1, \ldots, E_s\}$. Thus:

$$P(X|E_1 = e_1 \wedge \ldots \wedge E_s = e_s) = \sum_{Y_k} \cdots \sum_{Y_1} P(X_1, \ldots, X_n)_{[E_1=e_1 \wedge \ldots \wedge E_s=e_s]}$$
$$= \sum_{Y_k} \cdots \sum_{Y_1} \prod_{i=1}^{n} P(X_i|\pi_{X_i})_{[E_1=e_1 \wedge \ldots \wedge E_s=e_s]}$$

where the subscripted probabilities mean that the associated variables are assigned the corresponding values.

The problem of probabilistic inference is thus reduced to the problem of summing out variables from a product of functions. This can be efficiently solved using the distribution law of algebra. To compute the sum of products $xy + xz$ efficiently, we can distribute out the common factors and get $x(y + z)$. This is the essence of the variable elimination algorithm.

A *factor* is a representation of a function from a tuple of variables into the real numbers. This can be represented as a d-dimensional table, where d is the number of variables in the factor. Initially, the factors represent the set of conditional probability tables in the Bayesian network.

The *product* of two factors f_1 and f_2, written $f_1 \otimes_t f_2$, is a factor on the union of the variables in f_1 and f_2. To construct the product of factors $f_1 \otimes_t f_2 \otimes_t \ldots \otimes_t f_k$, whose union of variables are, say, $X_1, \ldots, X_r$, we construct an r-dimensional table having an entry for each combination $v_1, \ldots, v_r$, where $v_i \in domain(X_i)$. The value for the entry corresponding to $v_1, \ldots, v_r$ is obtained by multiplying the values obtained from each f_i applied to the projection of $v_1, \ldots, v_r$ onto the variables of f_i.

The *summing out* of variable Y from a factor f with variables $X_1, \ldots, X_k, Y$, written $\sum_Y f$ is the factor with variables $X_1, \ldots, X_k$, defined by:

$$(\sum_{Y} f)(X_1, \ldots, X_k) = \sum_{v_i \in domain(Y)} f(X_1, \ldots, X_k, Y = v_i)$$

Summing out a variable reduces the dimensionality of a factor by one. The values of the resulting factor are obtained by adding the values of the factor for each value of the variable to be summed out.

The *ve* algorithm works by keeping a set of factors, originally obtained from the initial set of conditional probabilities in the Bayesian network and then eliminating the non-query variables in the network one by one. The order that the variables are eliminated is called the elimination order. To eliminate an unobserved variable Y, we replace the factors in the set that contains Y, say, $f_{m+1}, f_{m+2}, \ldots, f_r$, by their product with Y summed out, namely, $\sum_Y f_{m+1} \otimes_t f_{m+2} \otimes_t \ldots \otimes_t f_r$. After all non-query variables are eliminated, the set only contains factors on the query variable(s). The posterior probability of the query variable(s) is obtained by multiplying the factors in the set and normalizing the resulting factor. The *ve* algorithm is summarized in Figure 3.1.

3.2 Variable Elimination with Clique Trees

While Bayesian networks are used to encode the probabilities and conditional independencies, many architectures perform probabilistic inference on a secondary structure of the Bayesian network called a clique tree, instead of working on the entire set of conditional probabilities directly. When the posterior probability of more than one variable is wanted, the clique tree serves as a cache to save us from recalculating some of the probabilities already computed for other variables.

The graph of a Bayesian network is a directed acyclic graph (DAG). The

To compute $P(X|E_1 = e_1 \wedge \ldots \wedge E_s = e_s)$

let F be the factors obtained from the original conditional probabilities
replace each $f \in F$ that involves some E_i with $f_{E_1=e_1,\ldots,E_s=e_s}$
while there is a factor involving a non-query variable
select non-query variable Y to eliminate
$F \leftarrow \mathit{eliminate}(Y, F)$
return $\mathit{normalize}(F, X)$

Procedure $\mathit{eliminate}(Y, F)$

partition F into:
$\{f_1, \ldots, f_m\}$ that do not contain Y
$\{f_{m+1}, \ldots, f_r\}$ that do contain Y
compute $f = \sum_Y f_{m+1} \otimes_t \ldots \otimes_t f_r$
return $\{f_1, \ldots, f_m, f\}$

Procedure $\mathit{normalize}(\{f_1, \ldots, f_r\}, X)$

compute $f = f_1 \otimes_t \ldots \otimes_t f_r$
compute $c = \sum_X f$
return f/c

Figure 3.1: The Variable Elimination Algorithm

moralized graph of a DAG is created by marrying the parents of each node and then dropping the directions of the arcs in the DAG. Marrying the parents of a node X means that we identify the set of nodes that represent the parents of X and connect each pair of nodes in the set with an undirected arc. An undirected graph is *triangulated* if and only if every cycle of length four or higher in the graph contains an arc that connects two non-adjacent nodes in the cycle. Kjærulff (1990) describes a procedure to triangulate arbitrary undirected graphs. A *clique* in an undirected graph is a set of nodes that forms a complete and maximal subgraph of this undirected graph. That is, every pair of distinct nodes in the clique is connected by an arc in the undirected graph and the clique is not properly contained in a larger complete subgraph. A *clique tree* is a tree whose nodes are cliques in the triangulated, moralized Bayesian network such that for the path connecting two cliques K_i and K_j, the intersection $K = K_i \cap K_j$ is a subset of every clique on the path. The efficiency of clique tree algorithms depends largely on the size of the clique tree and the size of the largest clique. These sizes are determined by the triangulation of the moral graph. Finding good triangulations in the building of compact clique trees is related to finding good elimination orderings for the variable elimination algorithm, as triangulating a graph can be done by selecting an ordering that the vertices of the graph (ie. the variables in the network) should be eliminated (Kjærulff, 1990). Figure 3.2 shows the graph of a Bayesian network (Example Network #2) with 8 variables (the probability tables are omitted). Its moral graph and triangulated moral graph are shown in Figure 3.3, and the resulting clique tree is shown in Figure 3.4.

Even though the variable elimination algorithm as described in the previous section is a query-based algorithm, it can be extended to work on clique trees to

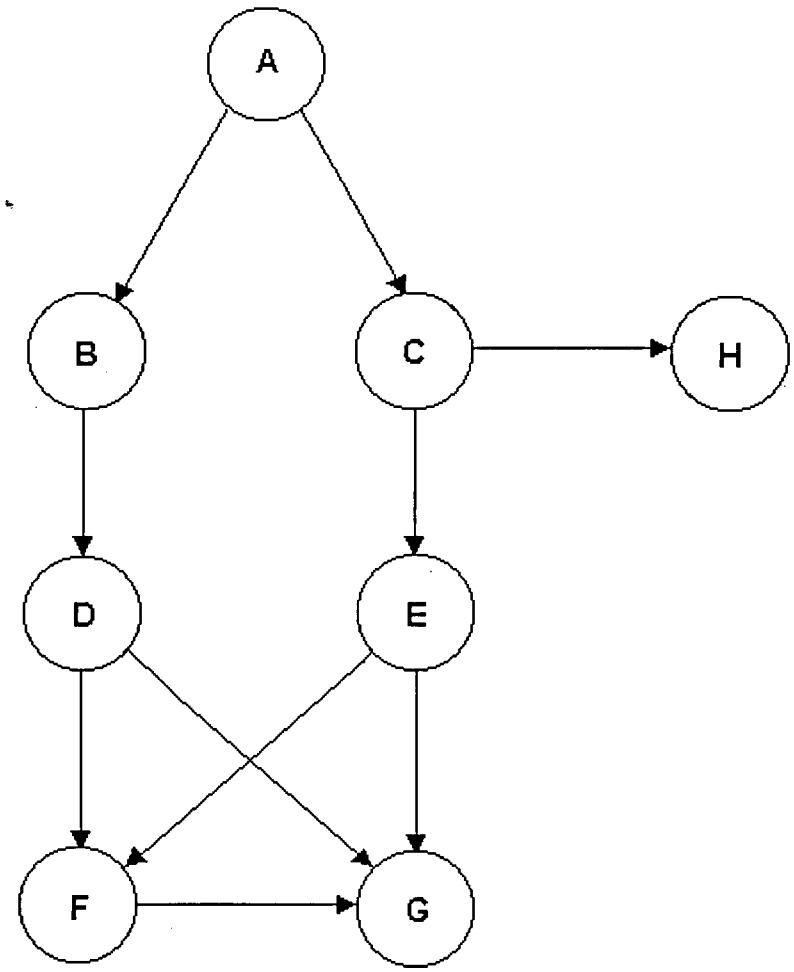


Figure 3.2: Example Network #2

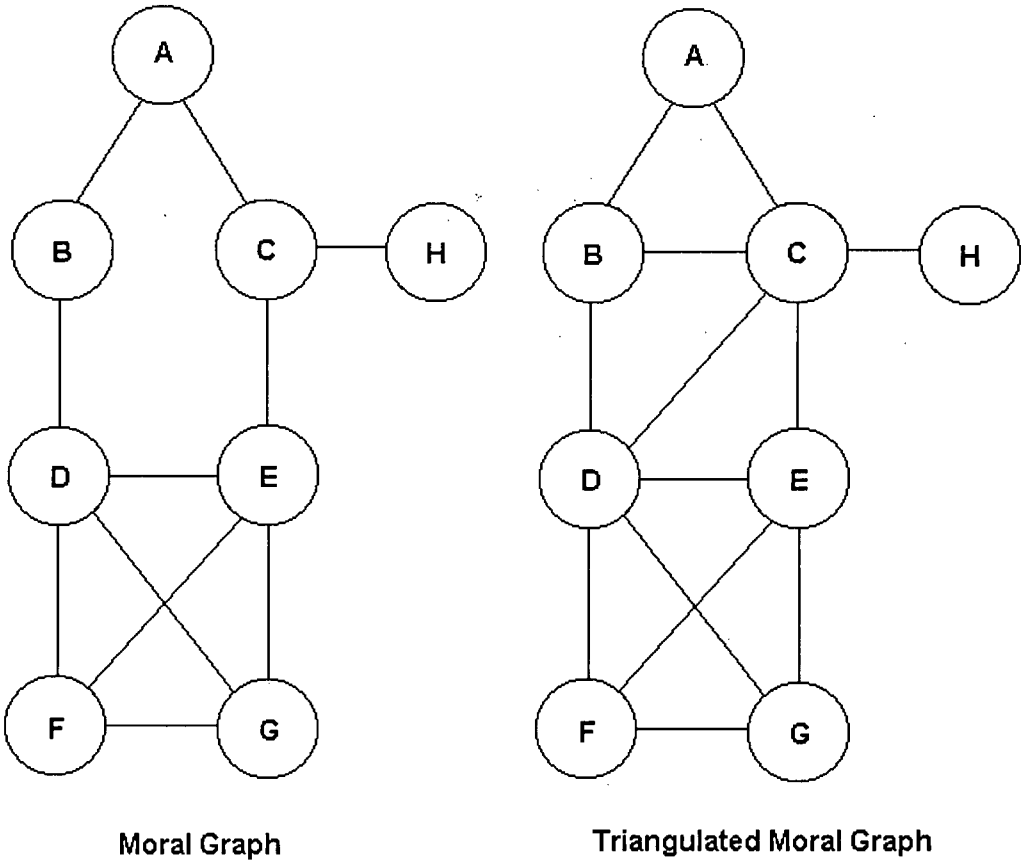


Figure 3.3: The moral graph and one of the triangulated moral graphs for Example Network #2

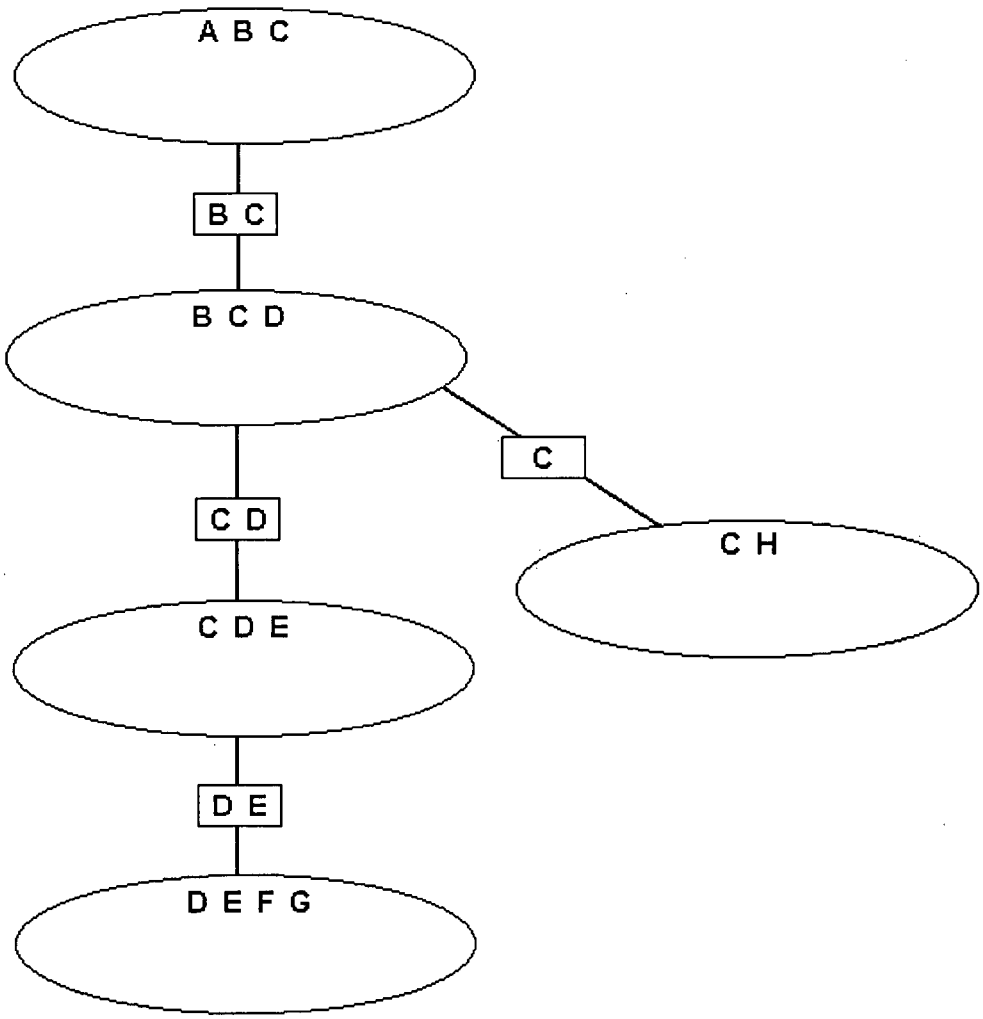


Figure 3.4: A clique tree for Example Network #2

obtain the posterior probabilities of all the variable in a Bayesian network simultaneously. We introduce the *ve-tree* algorithm as an extension of variable elimination on clique trees. The idea of *ve-tree* is initiated by other clique tree algorithms such as *Hugin* (Lauritzen and Spiegelhalter, 1988) that calculates the posterior probabilities of all variables. Even though the *ve-tree* algorithm is less efficient than *Hugin*, the contribution of *ve-tree* is that it provides the concept and structure that are the foundation of the *cve-tree* algorithm we propose to exploit context specific independence in Chapter 4.

With *ve-tree*, after the clique tree for a Bayesian network is constructed, each conditional probability table in the Bayesian network is assigned to a clique that consists of all the variables for that probability table. The cliques simply keep the conditional probabilities in a set. Observations are incorporated into the factors in the same way as in the *ve* algorithm. Each factor that contains an observed variable can zero out the entries in the factor that do not correspond to the variables' observed values, or it can be shrunk by deleting the dimensions that do not correspond to the variables' observed values. Probabilistic inference proceeds as message passing among the cliques. We keep a sepset between each pair of connected cliques in the clique tree to hold the two messages being passed between the cliques, on for each direction. The clique tree for the Bayesian network in Figure 3.2 with the initial factors assigned is shown in Figure 3.5.

The clique tree can now be made consistent by performing global propagation that consists of a series of message passes between adjacent cliques. In a consistent clique tree, each clique's factor represent the marginal probability of the variables in the clique given the observations. In a single message pass from clique K_1, with the set of variables X_1, to a neighbouring clique K_2, with the set of variables X_2,

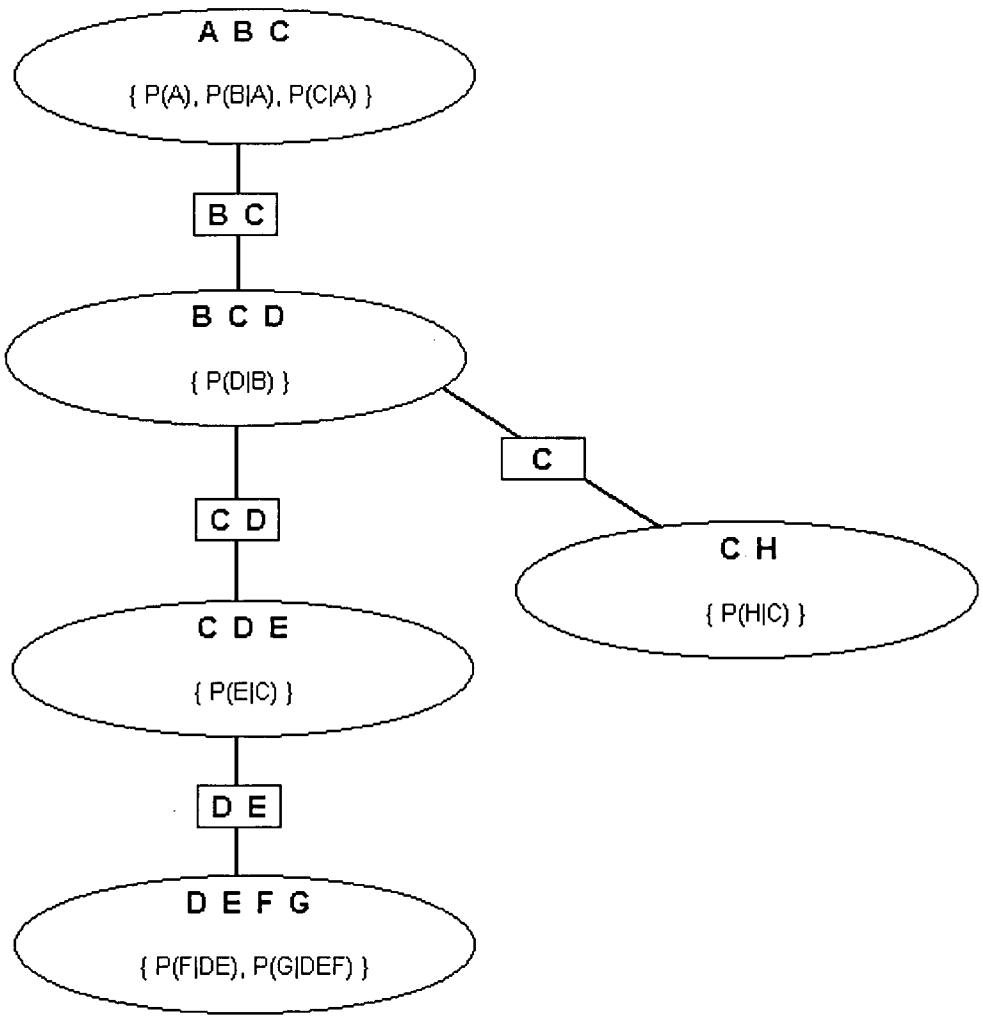


Figure 3.5: The initial clique tree of Example Network #2 using the *ve* algorithm

we take the set of factors in clique K_1, subtracted by the set of factors stored in the sepset between K_1 and K_2 in the direction to K_1 (this set of factors corresponds to a previous message passed from K_2 to K_1; it would be an empty set if K_2 has not passed any message to K_1). Let this set of factors be F. The message passed from K_1 to K_2, $M_{K_1K_2}$, is obtained by:

for each $Y \in X_1 - (X_1 \cap X_2)$
  $F \leftarrow eliminate(Y, F)$
$M_{K_1K_2} \leftarrow F$

where the *eliminate* procedure is as defined in Figure 3.1. The message $M_{K_1K_2}$ is stored in the sepset between K_1 and K_2 in the K_2 direction, and is unioned to the set of factors in the clique K_2.

For instance, if the clique $DEFG$ is to pass a message to the clique CDE in Figure 3.5, and CDE has not passed a message to $DEFG$ (so the sepset DE has no message stored), we need to eliminate variables F and G from the set of factors in clique $DEFG$. By calling the *eliminate* procedure on variables F and G on the factors $\{P(F|DE), P(G|DEF)\}$, the message:

$$\{f_1(D,E) = \sum_G \sum_F P(F|DE)P(G|DEF)\}$$

is obtained. This set of factors is stored in the sepset DE in the direction of clique CDE and unioned with the factors in clique CDE.

Given a clique tree with n cliques, global propagation starts with choosing an arbitrary clique, R, as the root clique, and then performing $2(n-1)$ message passes, divided into two phases, to make the clique tree consistent. During the *collect-evidence* phase, each clique passes a message to its neighbour in R's direction, beginning with the cliques farthest away from R. During the *distribute-evidence* phase, each clique passes messages to its neighbours away from R's direction. Each

of the two phases causes $n-1$ message to be passed. The procedures are shown in Figure 3.6.

The result of global propagation is that each clique passes its information to all other cliques in the clique tree. Note that a clique can only pass a message to a neighbour after the clique has received messages from all of its other neighbours. This ensures consistency of the clique tree when global propagation is completed. Figure 3.7 shows the clique tree of Figure 3.5 with the messages passed during global propagation (where clique BCD is chosen as the root clique) and the resulting set of factors in each clique after global propagation is completed. The order of message passed is shown by the circled numbers of the arrows in the direction of the messages.

After global propagation is completed, the clique tree becomes consistent and we can compute the probability of each variable of interest, say X, using the *calculate-probability* procedure.

The advantage of this extended algorithm of variable elimination is two-fold. First, the majority of the work for computing the posterior probabilities of all variables in the Bayesian network is done only once by making the clique tree consistent with global propagation. When the probability of a variable is desired, it can be obtained from the set of factors in the clique that contains that variable. This set of factors is much smaller in size than the entire set of conditional probabilities that *ve* must use to calculate for each query variable, and so the required probabilities can be more efficiently computed. This efficiency is especially apparent when the probabilities of many variables are queried at the same time, as *ve* needs to perform the variable elimination process from the beginning for each query variable. Second, this algorithm is more efficient than *ve* in the case of incremental observations, that is, where observations do not all come in together in the beginning. With *ve*, we

Procedure ***global-propagation***
    select an arbitrary clique R as root clique
    unmark all cliques
    call *collect-evidence*$(R, null)$
    unmark all cliques
    call *distribute-evidence*(R)

Procedure ***collect-evidence***$(K, Caller)$
    mark K
    for each $K' \in$ neigbours of K
        if K' unmarked
            call *collect-evidence*(K', K)
    if $Caller \neq null$ // not processing root clique
        call *pass-message*$(K, Caller)$

Procedure ***distribute-evidence***(K)
    mark K
    for each $K' \in$ neigbours of K
        if K' unmarked
            call *pass-message*(K, K')
            call *distribute-evidence*(K')

Procedure ***pass-message***(K_1, K_2) *//ve-tree*
    let $S =$ sepset between K_1 and K_2
    let $M_{21} =$ message stored in S in the direction of K_1
    let $F =$ set of factors in K_1
    $F \leftarrow F - M_{21}$
    for each $Y \in K_1 - (K_1 \cap K_2)$
        $F \leftarrow$ *eliminate*(Y, F) // Note: *eliminate* is shown in Figure 3.1
    set F in S in the direction of K_2
    let $G =$ set of factors in K_2
    $G \leftarrow G \cup F$

procedure ***calculate-probability(X)*** *//ve-tree*
    let $F =$ set of factors in K
    identify clique K such that $X \in K$
    for each $Y \in K - \{X\}$
        $F \leftarrow$ *eliminate*(Y, F)
    return *normalize*(F, X) // Note: *normalize* is shown in Figure 3.1

Figure 3.6: The *ve-tree* Algorithm

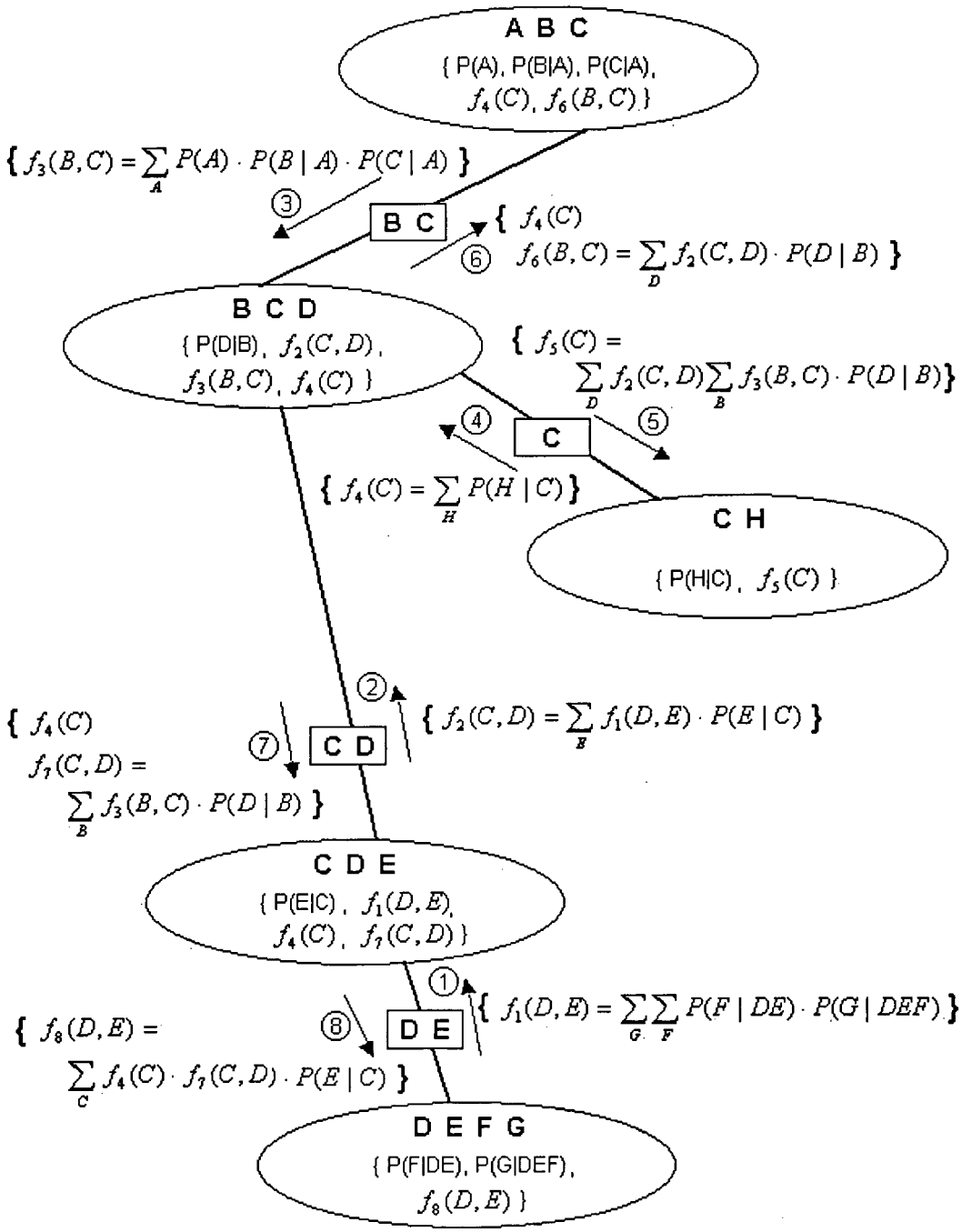


Figure 3.7: The consistent clique tree of Example Network #2 using the *ve-tree* algorithm

must redo all the calculations from the initial set of conditional probabilities for each query variable as each observation comes in. But with *ve-tree*, observations can be incorporated into a consistent clique tree by a *distribute-evidence* call from the clique whose factors are changed by the observation. If more than one observation comes in at one time, affecting more than one clique, then the clique tree can be made consistent again with a *collect-evidence* call followed by a *distribute-evidence* call. Therefore, in the case of incremental observations, instead of having to recalculate the posterior probabilities of all variables individually as in *ve*, *ve-tree* can maintain the clique tree's consistency with at most two phases of message passing, which is a significant savings if the posterior probability of many variables are required.

3.3 The Hugin Architecture

The *Hugin* architecture (Lauritzen and Spiegelhalter, 1988) is one of the most established and efficient algorithms for calculating exact posterior probabilities. The procedure for *Hugin* is very similar to *ve-tree* as described in the previous section. Whereas the cliques and sepsets in *ve-tree* keep sets of factors representing conditional probabilities, the cliques and sepsets in *Hugin* keep marginal probabilities by immediately multiplying the sets of factors when they are initialized and passed as messages. Moreover, the sepsets in *Hugin* only keeps one factor instead of two separate messages for each direction. Clique tree construction is done in the same way as *ve-tree*. When the conditional probability tables are assigned to the cliques, they are immediately multiplied to form a single factor. Sepsets and the cliques with no conditional probability table assigned are initialized with tables consisting of the sepsets and cliques with all entries being one. Observations are incorporated by zeroing out the entries in the factors that do not correspond to the variables'

observed values. Figure 3.8 shows the initial clique tree of the *Hugin* architecture for the network in Figure 3.2. After initialization, global propagation is performed the same way as in *ve-tree* (Figure 3.6), except for the *pass-message* procedure, which is revised as follows:

Procedure ***pass-message***(K_1, K_2) *//Hugin*
  let S = sepset between K_1 and K_2
  let ϕ_1 = factor in K_1
  let ϕ_2 = factor in K_2
  let ϕ_{old} = factor in S
  $\phi_{new} \leftarrow \sum_{K_1-(K_1 \cap K_2)} \phi_1$
  $\phi_2 \leftarrow \phi_2 \cdot (\phi_{new}/\phi_{old})$ // update the factor in clique K_2
  $\phi_{old} \leftarrow \phi_{new}$

The division step (ϕ_{new}/ϕ_{old}) in the *pass-message* procedure is analogous to the set subtraction step in *ve-tree*. It is done to prevent information that a neighbour previously passed to a clique to be passed back to the same neighbour. ϕ_{new} and ϕ_{old} are factors of the same set of variables and the division is done entry by entry in the factors.

As an example to demonstrate the *Hugin* algorithm, Figure 3.9 shows the clique tree of Example Network #1 in Figure 1.1, along with the initial factors for the cliques (without any observations). Suppose the clique $ABCD$ is chosen as the root clique. Propagation starts with the *collect-evidence* phase by passing a message from the leaf clique DEF to the clique CDE. The message is the factor of DEF with F summed out, resulting in a factor in the variables D and E. This factor in the message is divided by the factor stored in sepset DE (which is all ones at this time). The resulting factor is then mulitiplied with the factor in clique CDE and the clique CDE replaces its factor with this product. Finally, the factor in the message is saved in sepset DE. Clique CDE then passes a message to clique $ABCD$

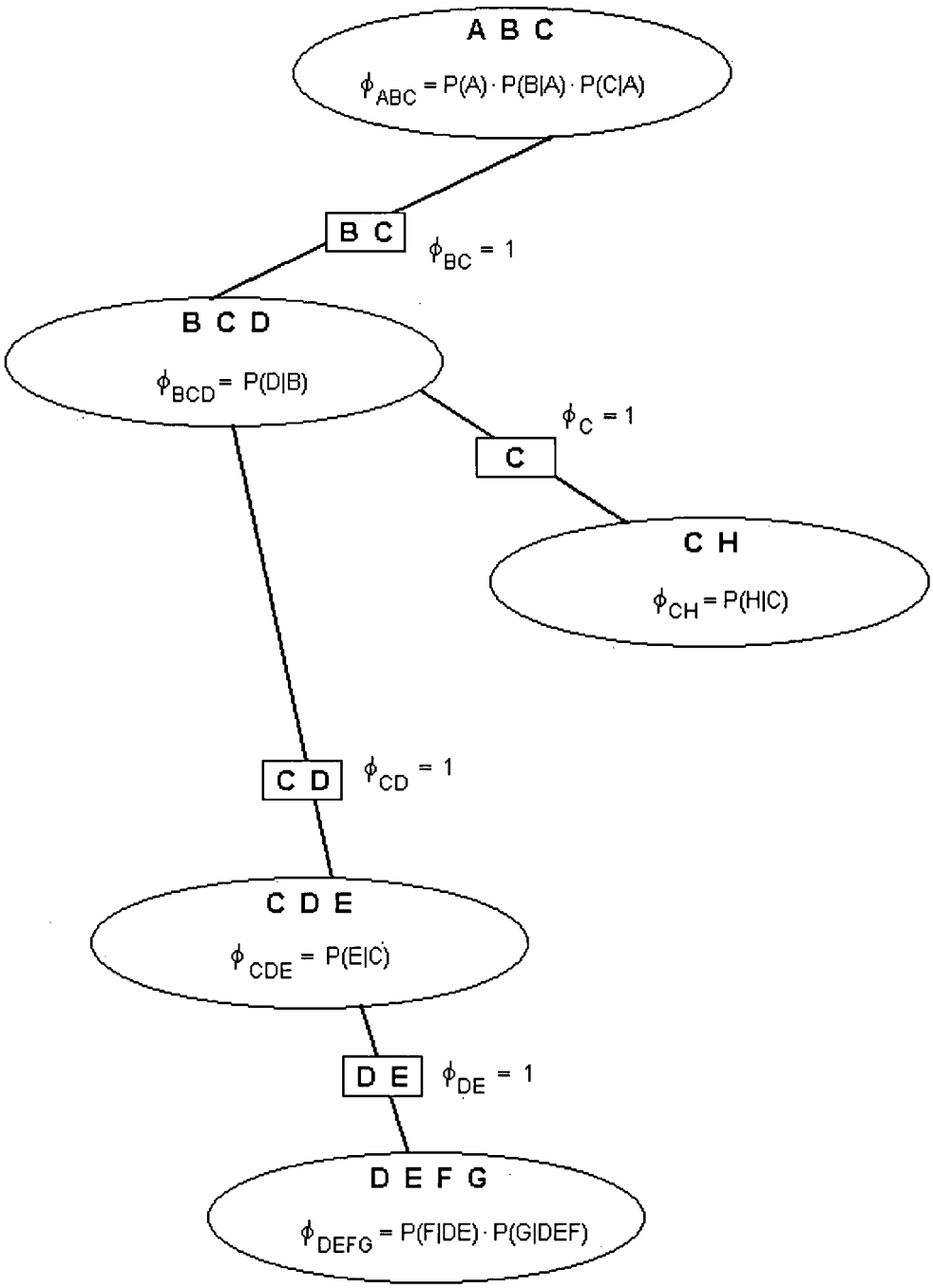


Figure 3.8: The initial clique tree of Example Network #2 using *Hugin*

in the same fashion. It first sums out the variable E from its factor and passes the resulting factor as message. This message is divided by the factor stored in sepset CD. The resulting factor is multiplied with the factor in clique $ABCD$, and the clique $ABCD$ replaces its factor with this product. Finally, the factor of the sepset CD is replaced with the factor in the message. This concludes the first sweep. In the *distribute-evidence* phase, messages are passed from the root clique to the leaf cliques. The same procedure of message passing is done, first passing a message from clique $ABCD$ to clique CDE, and then passing a message from clique CDE to clique DEF. After the message propagations are completed, the clique tree is consistent. The resulting factors of the entire clique tree are shown in Figure 3.10.

With a consistent clique tree, each clique's factor is the marginal probability $P(X|Y = Y_0)$, where X is the set of variables in the cliques and Y is the set of observed variables with corresponding observed values Y_0. The posterior probability of a variable can be readily obtained from the factor of a clique that contains the variable by summing out the other variables in that factor.

Jensen et al. (1990) and Huang and Darwiche (1996) provide more implementation details of the algorithm.

The advantage of *Hugin* over *ve-tree* is that the sets of factors in the cliques of *ve-tree* need to be multiplied for each message pass, whereas they are multiplied only once in *Hugin* (when the message is received). Moreover, in a consistent clique tree, each clique already has the posterior marginal probability on the variables of the clique, and so the posterior probability of a variable can be very efficiently computed by simply summing out the other variables. This makes Hugin more efficient than *ve-tree* in computation time.

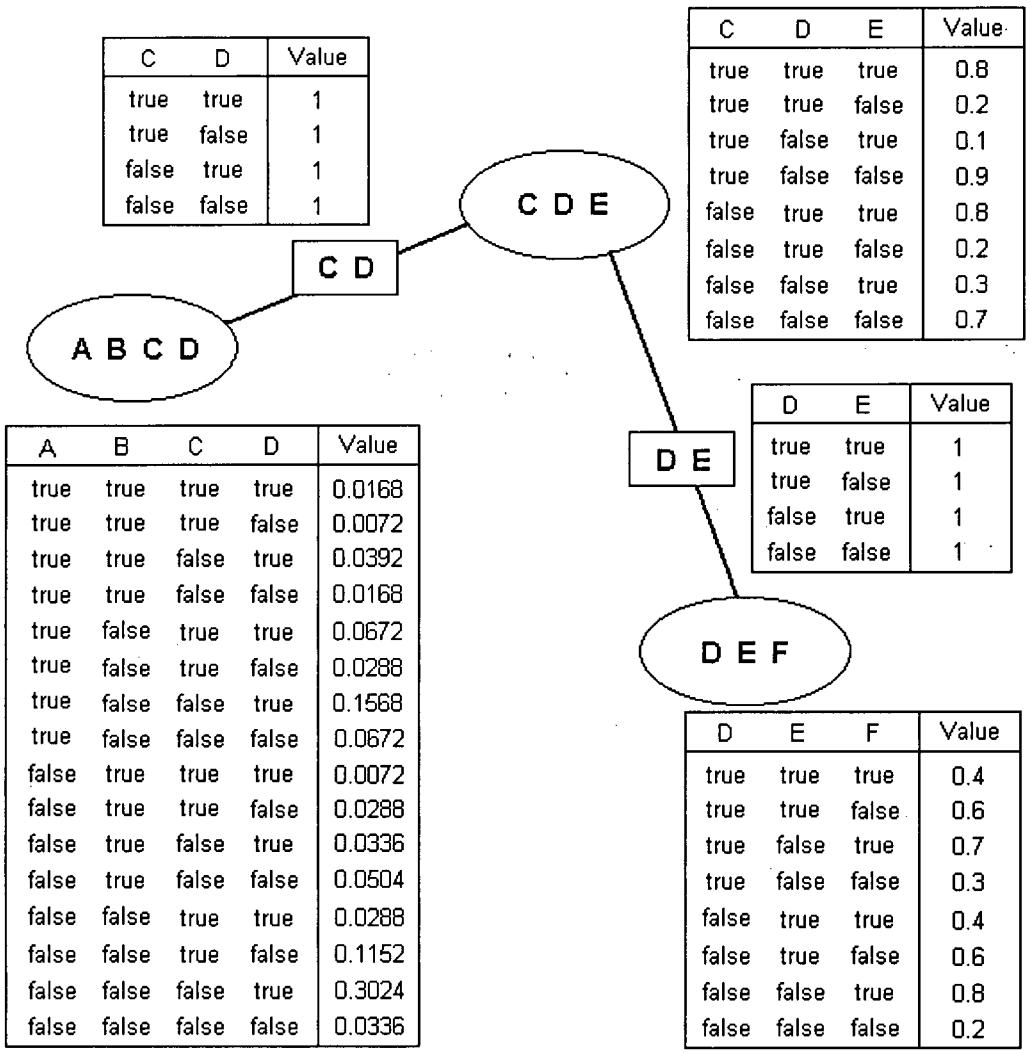
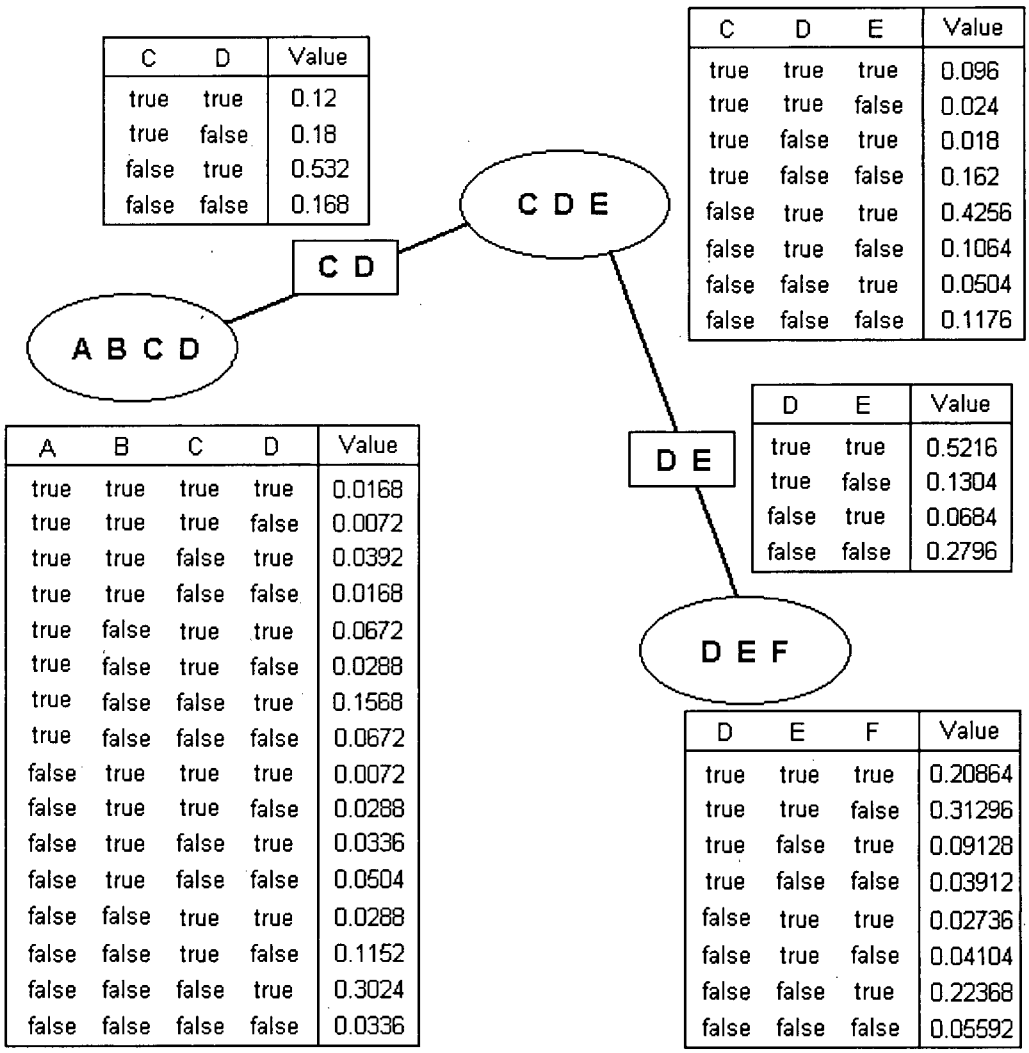


Figure 3.9: A clique tree for Example Network #1 with *Hugin*'s initial factors



C	D	Value
true	true	0.12
true	false	0.18
false	true	0.532
false	false	0.168

C	D	E	Value
true	true	true	0.096
true	true	false	0.024
true	false	true	0.018
true	false	false	0.162
false	true	true	0.4256
false	true	false	0.1064
false	false	true	0.0504
false	false	false	0.1176

D	E	Value
true	true	0.5216
true	false	0.1304
false	true	0.0684
false	false	0.2796

A	B	C	D	Value
true	true	true	true	0.0168
true	true	true	false	0.0072
true	true	false	true	0.0392
true	true	false	false	0.0168
true	false	true	true	0.0672
true	false	true	false	0.0288
true	false	false	true	0.1568
true	false	false	false	0.0672
false	true	true	true	0.0072
false	true	true	false	0.0288
false	true	false	true	0.0336
false	true	false	false	0.0504
false	false	true	true	0.0288
false	false	true	false	0.1152
false	false	false	true	0.3024
false	false	false	false	0.0336

D	E	F	Value
true	true	true	0.20864
true	true	false	0.31296
true	false	true	0.09128
true	false	false	0.03912
false	true	true	0.02736
false	true	false	0.04104
false	false	true	0.22368
false	false	false	0.05592

Figure 3.10: A consistent clique tree for Example Network #1 (*Hugin*)

3.4 Contextual Variable Elimination

The contextual variable elimination (*cve*) algorithm (Poole and Zhang, 2002) is an extension to variable elimination (Zhang and Poole, 1994), exploiting the context specific independence that may exist in the conditional probabilities of a Bayesian network. Instead of representing conditional probabilities in terms of factors, contextual variable elimination represents conditional probabilites in terms of *generalized rules* which capture context specific independence in variables (Section 2.1). A factor in *ve* is analogous to a set of rules whose contexts form a mutually exclusive and exhaustive set of parent contexts. The savings in time and space for probabilistic inference can be substantial when there is some context specific independence in the conditional probabilities. On the other hand, if there is no context specific independence, *cve* degenerates into *ve*, but with the overhead of maintaining the contexts.

The *cve* algorithm consists of three primitive operations: summing out a variable, multiplying rules, and splitting rules. At each stage of the algorithm, these operations maintain the following program invariant:

> The probability of a context on the non-eliminated variables can be obtained by multiplying the probabilities associated with rules that are applicable in that context.

The following descriptions and definitions on the operations are based on Poole and Zhang (2002).

There are two cases for *summing out* a variable, as the variable to be summed out can appear in the context or in the table. If the variable to be eliminated, Y, with domain $\{y_1, \ldots, y_s\}$, is in the table of a rule $\langle c, t \rangle$, then the rule can simply be

replaced by $\langle c, \sum_Y t \rangle$. If Y appears in the contexts of a set of rules R, we define $R_i = \{\langle c_i, t_i \rangle : \langle c_i \wedge Y = y_i, t_i \rangle \in R\}$, for $1 \leq i \leq s$, where each R_i represents the set of rules in which $Y = y_i$ is a part of the context and with $Y = y_i$ removed from the context. We define the operation $\oplus_g$ (addition for generalized rules) to sum out Y in R as:

$$R_i \oplus_g R_j = \{\langle c_i \cup c_j, \ set(t_i, c_j) \oplus_t set(t_j, c_i) \rangle : \\ \langle c_i, t_i \rangle \in R_i, \langle c_j, t_j \rangle \in R_j, \text{ and } compatible(c_i, c_j)\}$$

where $\oplus_t$ is table addition and $set(t, X = x)$ is the table t if t does not involve any variable in X, and is the table t with entries complying to the values $X = x$, if t involves some variable in X.

As an example, if we want to sum out the variable A from the rule collection

$R = \{ \ \langle A = false \wedge C = false,$

B	D	$Value$
$true$	$true$	0.24
$true$	$false$	0.36
$false$	$true$	0.54
$false$	$false$	0.06

$\rangle$,

$\langle A = false \wedge C = true,$

D	$Value$
$true$	0.12
$false$	0.48

$\rangle$,

$\langle A = true,$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle$ $\}$,

we have to partition the rules into rule collections R_1 and R_2, corresponding to rules compatible with $A = true$ and $A = false$, respectively, and with the variable A removed from the contexts of the rules.

$$R_1 = \{\langle TRUE,$$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle\}$

$$R_2 = \{\ \langle C = false,$$

B	D	$Value$
$true$	$true$	0.24
$true$	$false$	0.36
$false$	$true$	0.54
$false$	$false$	0.06

$\rangle,$

$$\langle C = true,$$

D	$Value$
$true$	0.12
$false$	0.48

$\rangle\ \}$

The resulting rules of adding the rule sets are:

$$R_1 \oplus_g R_2 = \{\ \langle C = false,$$

B	D	$Value$
$true$	$true$	$0.28 + 0.24 = 0.52$
$true$	$false$	$0.12 + 0.36 = 0.48$
$false$	$true$	$0.28 + 0.54 = 0.82$
$false$	$false$	$0.12 + 0.06 = 0.18$

$\rangle,$

$$\langle C = true,$$

D	$Value$
$true$	$0.28 + 0.12 = 0.4$
$false$	$0.12 + 0.48 = 0.6$

$\rangle\ \}$

These are the rules that result from summing out the variable A.

Multiplying rules in *cve* is much more complicated than multiplying factors in *ve* as rules involve both tables and contexts. If we want to multiply two rules whose contexts are identical, then the product is simply the rule with the same context and with the table that is the product of the two rules' tables. If we want to multiply two rules with different contexts, we cannot simply multiply their tables because they may belong to different contexts. To multiply two rules with different

contexts, we must change one of the rules or both rules so that their contexts become identical. This is done by *rule splitting.* When we want to split a rule $r = \langle c, t \rangle$ on a variable X, with domain $\{x_1, \ldots, x_k\}$, where X is not in the context c, there are two possible cases. First, if X does not appear in the table t, then we can simply replace r with the k rules $r_i = \langle c \wedge X = x_i, t \rangle$, for $1 \leq i \leq k$. If X appears in the table t, then r is replaced with the k rules $r_i = \langle c \wedge X = x_i, set(t, X = x_i) \rangle$, where $set(t, X = x_i)$ is the part of the table t that X takes on the value x_i, and with X removed from the table.

Definition 6 Given rule $r = \langle c, t \rangle$ and context c' such that c and c' are compatible, to *split* r *on* c' means that r is split on each of the variables that are assigned in c' that are not assigned in c.

When a rule r is split on a context c, we end up with a single rule with a context that is compatible with c. All other rules we end up with that are incompatible with c are called the *residual rules of splitting* r *on* c. Let $r = \langle c', t' \rangle$ and let c be a context that is compatible with c', then $residual(r, c)$ is defined as:

if $c \subseteq c'$
  then $residual(r, c) = \{\}$
  else Select a variable X that is assigned in c but not in c'
    Let x_0 be the value assigned to X in c
    $residual(r, c) =$
      $\{\langle c' \wedge X = x_i, set(t', X = x_i) \rangle : x_i \in domain(X)$ and $x_i \neq x_0\}$
      $\cup\ residual(\langle c' \wedge X = x_0, set(t', X = x_0) \rangle, c)$

Note that the result of splitting depends on the order that the variables are selected. However, the result satisfies the program invariant no matter which order the variables are selected.

For example, the residuals of splitting the rule

$$\langle X = true, \begin{array}{|ccc|c|} \hline Y & Z & W & Value \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \end{array} \rangle$$

in the context $Y = false \wedge Z = false$ (assuming all variables are boolean) are the following set of rules if we split first on Y and then on Z:

$$\{ \ \langle X = true \wedge Y = true, \begin{array}{|cc|c|} \hline Z & W & Value \\ \hline \vdots & \vdots & \vdots \\ \hline \end{array} \rangle,$$

$$\langle X = true \wedge Y = false \wedge Z = true, \begin{array}{|c|c|} \hline W & Value \\ \hline \vdots & \vdots \\ \hline \end{array} \rangle \ \}$$

where the table of the first rule contains values from the original table that correspond to $Y = true$, and the table of the second rule contains values from the original table that correspond to $Y = false$ and $Z = true$.

The rule can also be split by variable Z before variable Y, and would result in a different set of residuals:

$$\{ \ \langle X = true \wedge Z = true, \begin{array}{|cc|c|} \hline Y & W & Value \\ \hline \vdots & \vdots & \vdots \\ \hline \end{array} \rangle,$$

$$\langle X = true \wedge Z = false \wedge Y = true, \begin{array}{|c|c|} \hline W & Value \\ \hline \vdots & \vdots \\ \hline \end{array} \rangle \ \}$$

where the table of the first rule contains values from the original table that correspond to $Z = true$, and the table of the second rule contains values from the original table that correspond to $Z = false$ and $Y = true$.

Once two rules are split on each other's contexts, their contexts become identical and their tables can be multiplied by table multiplication. However, if residuals appear in the splitting, the residual rules must also be multiplied as well.

This would involve more rule splitting and become very complicated. Fortunately, there are situations that we do not need to split.

Definition 7 A set of rules R is *complete* in a set C of contexts if in any complete context with a member of C, there is exactly one member of R whose context is consistent with that complete context, and the context of every rule in R is consistent with at least one element of C. We say R is complete in a context c if R is complete in $\{c\}$.

For example,

- The rule set $R_1 = \{\langle a \wedge x, t_1 \rangle\}$ is complete in context $a \wedge x$.
- The rule set $R_2 = \{\langle a \wedge b \wedge x, t_1 \rangle, \langle a \wedge \bar{b} \wedge c \wedge x, t_2 \rangle, \langle a \wedge \bar{b} \wedge \bar{c} \wedge x, t_3 \rangle\}$ is complete in context $a \wedge x$.
- The rule set $R_3 = \{\langle a \wedge b \wedge x, t_1 \rangle, \langle a \wedge \bar{b} \wedge c, t_2 \rangle, \langle a \wedge \bar{b} \wedge \bar{c}, t_3 \rangle\}$ is not complete in context $a \wedge x$ as x is not in every rule's context in R_3, even though exactly one rule in R_3 will be used in any context consistent with $a \wedge x$.
- The rule set $R_4 = \{\langle a, t_1 \rangle, \langle \bar{a} \wedge b, t_2 \rangle\}$ is complete in the set of contexts $\{b, a \wedge b\}$.
- The set of rules obtained by considering the rows of a probability table is complete in the empty context.

If we have a set of rules that is complete in a context, we do not have to split other rules that contain this context when multiplying, because all the residual rules will also have a mate. We define the procedure *absorb* to multiply a single rule, $r = \langle c, t \rangle$, with a covering set of rules R (where $r \notin R$) that is complete in the empty context as follows:

$$
\begin{aligned}
&absorb(R, \langle c, t\rangle) = \\
&\quad \{\langle c_i, t_i\rangle \in R : incompatible(c, c_i)\} \cup \\
&\qquad \bigcup_{\substack{\langle c_i, t_i\rangle \in R \text{ and} \\ compatible(c, c_i)}} \left(\begin{array}{l} (residual(\langle c_i, t_i\rangle, c) \\ \cup\{\langle c \cup c_i, set(t, c_i) \otimes_t set(t_i, c)\rangle\} \end{array} \right)
\end{aligned}
$$

where $\otimes_t$ is table multiplication.

For example, to absorb the rule

$r = \langle A = false \wedge C = false,$

B	D	$Value$
$true$	$true$	0.4
$true$	$false$	0.6
$false$	$true$	0.9
$false$	$false$	0.1

$\rangle$

into the rule collection

$R = \{\ \langle A = false \wedge C = false, 0.6\rangle$

$\langle A = false \wedge C = true,$

D	$Value$
$true$	0.12
$false$	0.48

$\rangle$

$\langle A = true,$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle\ \}$

only the first rule in R is compatible with the context of r (which creates no residual).

So the resulting rule collection is:

$absorb(R, r) = \{\ \langle A = false \wedge C = false,$

B	D	$Value$
$true$	$true$	$0.4 * 0.6 = 0.24$
$true$	$false$	$0.6 * 0.6 = 0.36$
$false$	$true$	$0.9 * 0.6 = 0.54$
$false$	$false$	$0.1 * 0.6 = 0.06$

$\rangle,$

$\langle A = false \wedge C = true,$

D	$Value$
$true$	0.12
$false$	0.48

$\rangle,$

$\langle A = true,$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle$ $\}$

In order for absorption to work for multiplying rules, we need to be able to easily find a set of rules that is complete in a context. Initially, the rules that represent the conditional probability table $P(X|\pi_X)$ are complete in the empty context, for all variables X. But these initial rules may not exist anymore after they are split, multiplied, or added with other rules. It turns out, however, that at any stage of the contextual variable elimination, we can extract a set of rules that all contain a variable X and the set is complete in the empty context. This set of rules is called the *rules for* X and is defined as follows:

Definition 8 If X is a variable, the *rules for* X are a subset of the rule base consisting of the following rules:

- The rules that define the conditional probability $P(X|\pi_X)$ initially.
- The rules created when splitting a rule for X.
- The rules created when multiplying a rule for X with another rule.
- The rules created when adding a rule for X with another rule.

The set of rules for X forms the covering set of rules complete in the empty context that absorption bases on initially. When we eliminate X, we need to multiply

the rules that involve X. We start with the set of rules for X, and absorb all other rules that involve X into the set one by one. The result of the multiplication is the set of rules obtained after absorption is done for all the rules to be multiplied.

Evidence absorption is a simplification of the rule base by removing the observed variables from the rules' contexts and tables. If the variables $E_1, \ldots, E_s$ are observed with the values $e_1, \ldots, e_s$, respectively, then the rule base is modified as follows:

- remove any rules that contain $E_i = e_i'$ in the context, where $e_i \neq e_i'$
- remove any term $E_i = e_i$ in the context of a rule
- replace each table t with $t_{E_1=e_1 \wedge \ldots \wedge E_s=e_s}$
- remove any rule that does not involve any variable (i.e. rules with an empty context and a single number as the table)

Rules that do not involve any variable are considered redundant because the only purpose they serve is as a normalization constant. There is another kind of rules that is redundant: rules with no *for* variables and whose tables consist of all ones. These rules occur when eliminating a variable that has no children in the particular context. They are considered redundant rules because they are equivalent to recursively removing a non-observed, non-queried variable with no children in a Bayesian network, which would not change the conditional probability of the query variable. Redundant rules can be removed from the rule base because they do not affect the probability calculation for the query variable.

The posterior probability of a query variable X can be obtained after absorbing the evidence variables and eliminating the remaining variables. Only rules of the

form $\langle \{X = x_i\}, p_i \rangle$ and $\langle TRUE, t_k[X] \rangle$ remain (p_i is a table of a single number). We have:

$$P(X = x_i \wedge E = e) = \kappa \prod_{\langle \{X=x_i\}, p_i \rangle} p_i \prod_{\langle TRUE, t_k[X] \rangle} t_k[x_i]$$

where κ is a normalization constant, and we have:

$$P(X = x_i | E = e) = \frac{P(X = x_i \wedge E = e)}{\sum_{x_j} P(X = x_j \wedge E = e)}$$

Notice that the normalization constants κ in the numerator and the denominator cancel each other.

A summary of the contextual variable elimination algorithm is given in Figure 3.11.

To compute $P(X|E_1 = e_1 \wedge \ldots \wedge E_s = e_s)$
Given a rule base of generalized rules R
  observe$(E_1 = e_1 \wedge \ldots \wedge E_s = e_s)$
  while there is a rule in the rule base involving a non-query variable
    select non-query variable Y
    $R \leftarrow$ *eliminate*(Y, R)
  compute posterior probability of X

Procedure *eliminate*(Y, R)
  partition the rule base R into:
    $R^- = \{r \in R : r$ doesn't involve $Y\}$
    $R^+ = \{r \in R : r$ is *for* $Y)\}$
    $R^* = \{r \in R : r$ involves Y and r is not *for* $Y)\}$
  for each $r \in R^*$
    do $R^+ \leftarrow$ *absorb*(R^+, r)
  partition R^+ into:
    $R^t = \{r \in R^+ : Y$ in table of $r\}$
    $R_i = \{\langle c, t\rangle : \langle c \wedge Y = y_i, t\rangle \in R^+\}$ for each $1 \leq i \leq n$
  return rule base $R^- \cup (R_1 \oplus_g \ldots \oplus_g R_n) \cup \{\langle c', \sum_Y t'\rangle : \langle c', t'\rangle \in R^t\}$

Figure 3.11: The Contextual Variable Elimination Algorithm

Chapter 4

Probability Inference Using *cve-tree*

4.1 Contextual Clique Tree

The *cve* algorithm as described in Section 3.4 is a query-based algorithm that calculates posterior probabilities from the initial rule base each time a query variable is requested. This is inefficient if the network is large, the rule base is complicated, and the number of query variables is large, as many operations must be repeated for each query. We can alleviate this problem by extending the *cve* algorithm to be used in a clique tree structure, similar to the extension of the *ve* algorithm applied to clique trees as described in Section 3.2. Since each clique contains only a subset of the variables in the network, the number of rules applicable to a particular clique is much smaller than the entire rule base. As a result, after spending some overhead time to obtain a consistent clique tree, the time required to calculate the posterior probabilities for this new algorithm, *cve-tree*, can be much smaller than the time required by *cve*. Note that the *cve* algorithm is a specific case of the *cve-tree* algo-

rithm, in which the clique tree consists of a single clique, namely, the clique with all variables in the network.

The *cve-tree* algorithm works as follows:

Rule Base Initialization

- The initial rule base is constructed and the observations are incorporated to the rule base the same way as the query-based contextual variable elimination algorithm (Section 3.4).

Clique Tree Initialization

- The clique tree is constructed the same way as the *ve-tree* and *Hugin* architectures (Section 3.2), but instead of associating each clique with a factor, each clique is associated with a set of rules.

- The rules in the initial rule base that are *for* variable X are assigned to a clique that is *for* X. A clique is *for* a variable X if it contains X and all the parents of X. There may be multiple cliques *for* X, and any one of these cliques can be chosen to assign the initial rules without affecting the result.

- Sepsets are also created between neighbouring cliques to hold the messages (rules) passed between neighbours. Each sepset contains two rule collections to hold messages passed in both directions.

Message Propagation

- After initializing the clique tree, each clique contains a rule collection on the clique's variables. Clique tree propagation proceeds the same way as *ve-tree* (Section 3.2), with the messages being rule collections instead of factors.

- To pass a message from clique K_i to its neighbour K_j, the message is the resulting rule collection from calling the *eliminate*(Y, R) procedure (Figure 3.11) on each variable $Y \in K_i - K_j$, where R is the rule collection of K_i, minus those rules that K_j has passed to K_i previously (if any). This can be done simply by taking the set difference of the rule collection of K_i and the rule collection stored in the sepset betweeen K_i and K_j that is the message passed from K_j to K_i previously.

- When K_j receives a message, its rule collection, R_j, is simply updated with the union of R_j and the rules in the received message. A copy of the message is also stored in the sepset between K_i and K_j. As opposed to *Hugin*, no rule multiplications are performed when a clique receives a message.

Posterior Probability Computation

- After each clique has sent and received a message from all its neighbours, the clique tree is consistent. The posterior probability of a variable X can be obtained from the rule collection of the clique that is for X by calling the *eliminate* procedure (Figure 3.11) on the other variables that the clique contains.

The major difference between *Hugin* and *cve-tree* is that *Hugin* updates the marginal probabilities of the clique by immediately multiplying all messages received, whereas *cve-tree* only stores the rules received without multiplying them immediately. Multiplication of the rules is delayed until the posterior probability computation stage. The reason for the delay is that once the rules are multiplied, any structure of context specific independence captured in the rules may be lost. If rules are multiplied as they are received by the cliques in the message propagation

stage, then the cliques revert to storing marginals, defeating the advantages of *cve-tree* of exploiting the context specific independence in the network.

4.2 An Example

Let's run the algorithm with Example Network #1 (Figure 1.1). Suppose we want to calculate the probability of variable F, with no observations. We start by constructing the clique tree, which has the same structure as that of *Hugin* architecture, with the addition of which variables the cliques are for. The rules in the rule base (Figure 2.1) are assigned to the cliques according to the variable(s) that the rules are for. The clique tree constructed along with the initial rules is shown in Figure 4.1.

Suppose the clique $ABCD$ is chosen as the root. Propagation begins with passing messages from the leaf clique (DEF) inwards. Eliminating F from the two rules of the clique DEF, we find that the resulting rules are both redundant, as both rules are for F (so after eliminating F, the rules are for no variable) and both tables consist of all ones. As a result, a message of an empty rule collection is sent to the clique CDE. Eliminating the variable E from the two rules of the clique CDE also results in redundant rules, so a message of an empty rule collection is sent to clique $ABCD$.

After the root clique has received messages from all its neighbours, it can start propagating messages outwards to the leaf cliques. We need to eliminate the variables A and B from the rules of clique $ABCD$. We begin by eliminating A and partitioning the 6 rules (all of which are initially in clique $ABCD$, ie. not passed from clique CDE) into R^- (rules that do not involve A), R^+ (rules that are *for* A), and R^* (rules that involve A but are not *for* A):

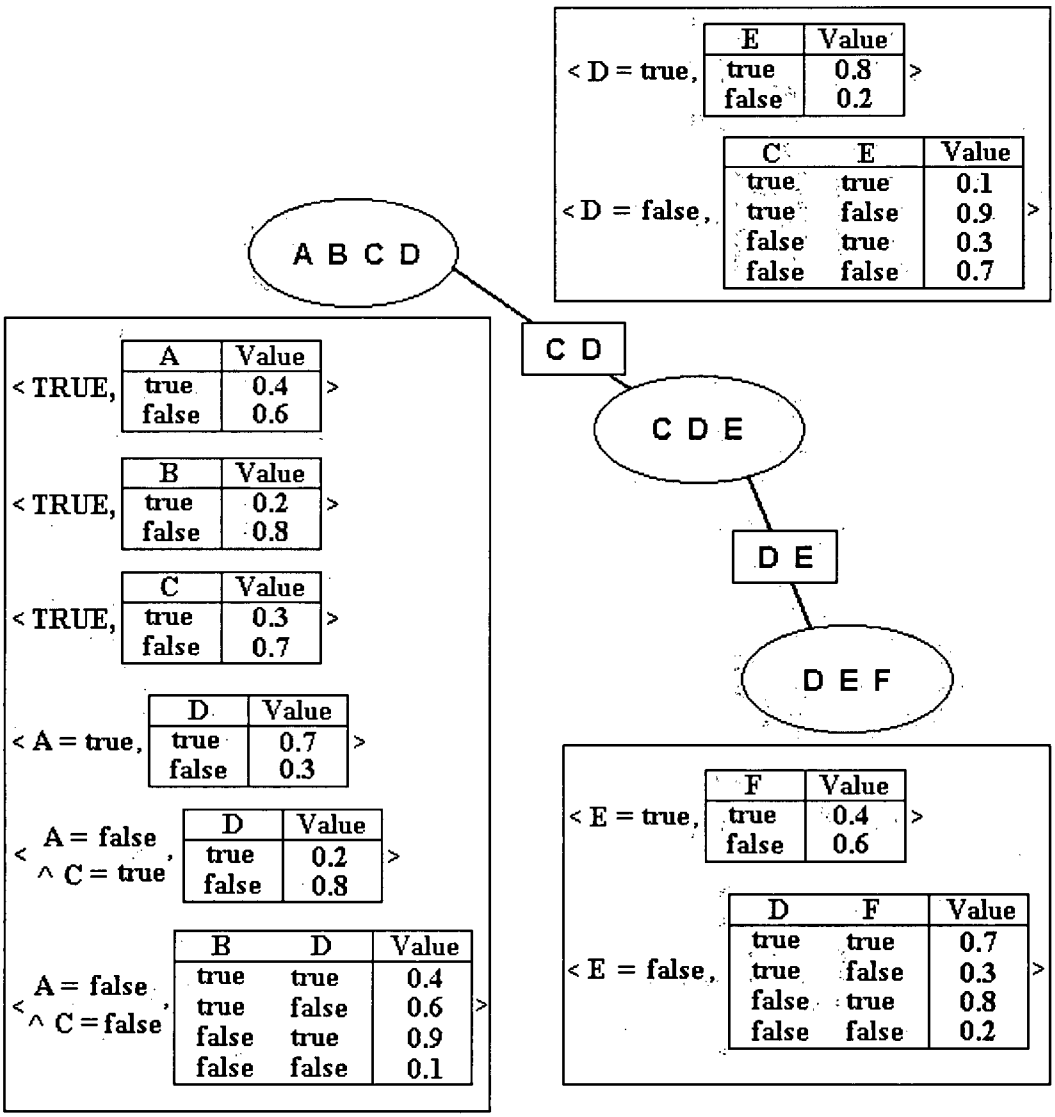


Figure 4.1: A clique tree of Example Network #1 with initial rules (*cve-tree*)

$$R^- = \{\ \langle TRUE,$$

B	$Value$
$true$	0.2
$false$	0.8

$\rangle, \langle TRUE,$

C	$Value$
$true$	0.3
$false$	0.7

$\rangle\ \}$

$$R^+ = \{\langle TRUE,$$

A	$Value$
$true$	0.4
$false$	0.6

$\rangle\ \}$

$$R^* = \{\ \langle A = true,$$

D	$Value$
$true$	0.7
$false$	0.3

$\rangle,$

$$\langle A = false \wedge C = true,$$

D	$Value$
$true$	0.2
$false$	0.8

$\rangle,$

$$\langle A = false \wedge C = false,$$

B	D	$Value$
$true$	$true$	0.4
$true$	$false$	0.6
$false$	$true$	0.9
$false$	$false$	0.1

$\rangle\ \}$

We absorb rules in R^* one by one into R^+:

$$R_r^+ = absorb(R^+, \langle A = true,$$

D	$Value$
$true$	0.7
$false$	0.3

$\rangle)$

$$= \{\langle A = false, 0.6\rangle, \langle A = true,$$

D	$Value$
$true$	$0.7 * 0.4 = 0.28$
$false$	$0.3 * 0.4 = 0.12$

$\rangle\ \}$

where the first rule is the residual rule created. So R^+ becomes:

$$R_{\prime}^{+} = \{\langle A = false, 0.6\rangle, \langle A = true,$$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle\}$

Continuing the absorption,

$$R_{\prime\prime}^{+} = absorb(R_{\prime}^{+}, \langle A = false \wedge C = true,$$

D	$Value$
$true$	0.2
$false$	0.8

$\rangle)$

$$= \{\ \langle A = false \wedge C = false, 0.6\rangle,$$

$$\langle A = false \wedge C = true,$$

D	$Value$
$true$	$0.2 * 0.6 = 0.12$
$false$	$0.8 * 0.6 = 0.48$

$\rangle,$

$$\langle A = true,$$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle\ \}$

where the first rule is a residual rule, and the third rule remains the same as it is incompatible with the absorbing rule. Absorbing the last rule, we have:

$$R_{\prime\prime\prime}^{+} = absorb(R_{\prime\prime}^{+}, \langle A = false \wedge C = false,$$

B	D	$Value$
$true$	$true$	0.4
$true$	$false$	0.6
$false$	$true$	0.9
$false$	$false$	0.1

$\rangle)$

$$= \{\ \langle A = false \wedge C = false,$$

B	D	$Value$
$true$	$true$	$0.4 * 0.6 = 0.24$
$true$	$false$	$0.6 * 0.6 = 0.36$
$false$	$true$	$0.9 * 0.6 = 0.54$
$false$	$false$	$0.1 * 0.6 = 0.06$

$\rangle,$

$$\langle A = false \wedge C = true,$$

D	$Value$
$true$	0.12
$false$	0.48

$\rangle,$

$\langle A = true,$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle$ }

After absorbing all the rules in R^*, the 3 resulting rules in R^+ all have the eliminating variable A in the contexts. So we split them into R_1 and R_2, corresponding to $A = true$ and $A = false$, respectively, and with A removed from the contexts:

$R_1 = \{\langle TRUE,$

D	$Value$
$true$	0.28
$false$	0.12

$\rangle\}$

$R_2 = \{ \ \langle C = false,$

B	D	$Value$
$true$	$true$	0.24
$true$	$false$	0.36
$false$	$true$	0.54
$false$	$false$	0.06

$\rangle,$

$\langle C = true,$

D	$Value$
$true$	0.12
$false$	0.48

$\rangle\}$

and we add the two set of rules:

$R_1 \oplus_g R_2 = \{ \langle C = false,$

B	D	$Value$
$true$	$true$	$0.28 + 0.24 = 0.52$
$true$	$false$	$0.12 + 0.36 = 0.48$
$false$	$true$	$0.28 + 0.54 = 0.82$
$false$	$false$	$0.12 + 0.06 = 0.18$

$\rangle,$

$\langle C = true,$

D	$Value$
$true$	$0.28 + 0.12 = 0.4$
$false$	$0.12 + 0.48 = 0.6$

$\rangle$ }

Thus, the rule base becomes:

$$R = R^{-} \cup (R_1 \oplus_g R_2)$$

$$= \{ \ \langle TRUE,$$

B	$Value$
$true$	0.2
$false$	0.8

$\rangle$,

$\langle TRUE,$

C	$Value$
$true$	0.3
$false$	0.7

$\rangle$,

$\langle C = false,$

B	D	$Value$
$true$	$true$	0.52
$true$	$false$	0.48
$false$	$true$	0.82
$false$	$false$	0.18

$\rangle$,

$\langle C = true,$

D	$Value$
$true$	0.4
$false$	0.6

$\rangle$ $\}$

The procedure for eliminating variable B from R is similarly performed:

$$R^{-} = \{\langle TRUE,$$

C	$Value$
$true$	0.3
$false$	0.7

$\rangle, \langle C = true,$

D	$Value$
$true$	0.4
$false$	0.6

$\rangle\}$

$$R^{+} = \{\langle TRUE,$$

B	$Value$
$true$	0.2
$false$	0.8

$\rangle\}$

$$R^* = \{ \ \langle C = false,$$

B	D	$Value$
$true$	$true$	0.52
$true$	$false$	0.48
$false$	$true$	0.82
$false$	$false$	0.18

$\rangle\}$

We absorb the rules in R^* into R^+:

$$R_r^+ = absorb(R^+, \langle C = false,$$

B	D	$Value$
$true$	$true$	0.52
$true$	$false$	0.48
$false$	$true$	0.82
$false$	$false$	0.18

$\rangle)$

$$= \{ \ \langle C = true,$$

B	$Value$
$true$	0.2
$false$	0.8

$\rangle,$

$$\langle C = false,$$

B	D	$Value$
$true$	$true$	$0.2 * 0.52 = 0.104$
$true$	$false$	$0.2 * 0.48 = 0.096$
$false$	$true$	$0.8 * 0.82 = 0.656$
$false$	$false$	$0.8 * 0.18 = 0.144$

$\rangle \ \}$

Since the variable B appears in the tables of both rules, we can simply sum it out in the table. Note that after B is summed out, the first rule in R_r^+ can be removed because it is redundant. The resulting rules to be passed as message are:

$$R = \{ \ \langle TRUE,$$

C	$Value$
$true$	0.3
$false$	0.7

$\rangle,$

$$\langle C = true,$$

D	$Value$
$true$	0.4
$false$	0.6

$\rangle,$

$\langle C = false,$

D	$Value$
$true$	$0.104 + 0.656 = 0.76$
$false$	$0.096 + 0.144 = 0.24$

$\rangle$,

When clique CDE receives this message, it simply unions these rules with its rule base (for a total of 5 rules). It then eliminates the variable C from its rule base to pass a message to clique DEF as follows:

$R^- = \{\langle D = true,$

E	$Value$
$true$	0.8
$false$	0.2

$\rangle\}$

$R^+ = \{\langle TRUE,$

C	$Value$
$true$	0.3
$false$	0.7

$\rangle\}$

$R^* = \{$ $\langle C = true,$

D	$Value$
$true$	0.4
$false$	0.6

$\rangle$,

$\langle C = false,$

D	$Value$
$true$	0.76
$false$	0.24

$\rangle$,

$\langle D = false,$

C	E	$Value$
$true$	$true$	0.1
$true$	$false$	0.9
$false$	$true$	0.3
$false$	$false$	0.7

$\rangle$ $\}$

Absorbing the rules in R^* into R^+, we have:

$$R_{\prime}^{+} = absorb(R^{+}, \langle C = true,$$

D	$Value$
$true$	0.4
$false$	0.6

$\rangle)$

$$= \{ \ \langle C = false, 0.7 \rangle$$

$$\langle C = true,$$

D	$Value$
$true$	$0.3 * 0.4 = 0.12$
$false$	$0.3 * 0.6 = 0.18$

$\rangle \ \}$

$$R_{\prime\prime}^{+} = absorb(R_{\prime}^{+}, \langle C = false,$$

D	$Value$
$true$	0.76
$false$	0.24

$\rangle)$

$$= \{ \ \langle C = false,$$

D	$Value$
$true$	$0.7 * 0.76 = 0.532$
$false$	$0.7 * 0.24 = 0.168$

$\rangle,$

$$\langle C = true,$$

D	$Value$
$true$	0.12
$false$	0.18

$\rangle \ \}$

$$R_{\prime\prime\prime}^{+} = absorb(R_{\prime\prime}^{+}, \langle D = false,$$

C	E	$Value$
$true$	$true$	0.1
$true$	$false$	0.9
$false$	$true$	0.3
$false$	$false$	0.7

$\rangle)$

$$= \{ \ \langle C = false \wedge D = true, 0.532 \rangle,$$

$$\langle C = false \wedge D = false,$$

E	$Value$
$true$	$0.168 * 0.3 = 0.0504$
$false$	$0.168 * 0.7 = 0.1176$

$\rangle,$

$$\langle C = true \wedge D = true, 0.12 \rangle,$$

$$\langle C = true \wedge D = false,$$

E	$Value$
$true$	$0.18 * 0.1 = 0.018$
$false$	$0.18 * 0.9 = 0.162$

$\rangle$ }

The variable C to be eliminated appears in the contexts of all 4 rules, so we partition the rules into R_1 and R_2 corresponding to $C = true$ and $C = false$, respectively, with the variable C removed from the contexts, and add the two sets of rules.

$$R_1 = \{ \ \langle D = true, 0.12 \rangle, \langle D = false,$$

E	$Value$
$true$	0.018
$false$	0.162

$\rangle\}$

$$R_2 = \{ \ \langle D = true, 0.532 \rangle, \langle D = false,$$

E	$Value$
$true$	0.0504
$false$	0.1176

$\rangle\}$

$$R_1 \oplus_g R_2 = \{ \ \langle D = true, 0.12 + 0.532 = 0.652 \rangle,$$

$$\langle D = false,$$

E	$Value$
$true$	$0.018 + 0.0504 = 0.0684$
$false$	$0.162 + 0.1176 = 0.2796$

$\rangle$ }

So the message passed from clique CDE to clique DEF is:

$$R = R^- \cup (R_1 \oplus_g R_2) = \{ \ \langle D = true,$$

E	$Value$
$true$	0.8
$false$	0.2

$\rangle$,

$$\langle D = true, 0.652 \rangle,$$

$\langle D = false,$

E	$Value$
$true$	0.0684
$false$	0.2796

$\rangle$ }

These rules are unioned with the rule base of clique DEF.

At this point, clique tree propagation is completed and the clique tree is consistent. The consistent clique tree with the cliques' rule collections are shown in Figure 4.2 (the messages stored in the sepsets are omitted here). We can now calculate the probability of variable F from the rules in clique DEF (which is *for* variable F), by eliminating the other variables, D and E.

To eliminate D from the rule base of clique DEF, the rules are first split into:

$R^- = \{\langle E = true,$

F	$Value$
$true$	0.4
$false$	0.6

$\rangle\}$

$R^+ = \{\langle D = true, 0.652\rangle\}$

$R^* = \{$ $\langle D = false,$

E	$Value$
$true$	0.0684
$false$	0.2796

$\rangle,$

$\langle D = true,$

E	$Value$
$true$	0.8
$false$	0.2

$\rangle,$

$\langle E = false,$

D	F	$Value$
$true$	$true$	0.7
$true$	$false$	0.3
$false$	$true$	0.8
$false$	$false$	0.2

$\rangle$ }

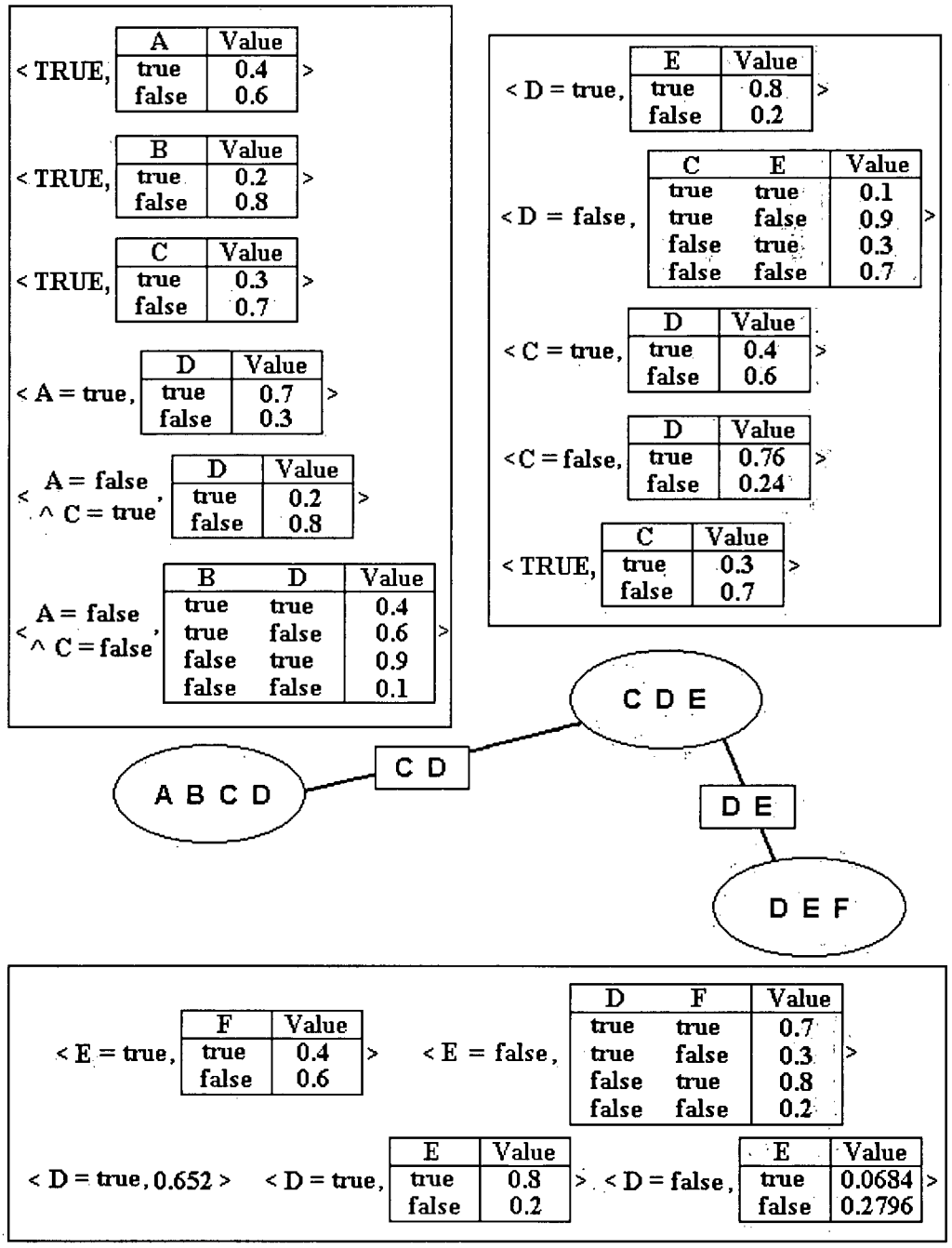


Figure 4.2: The consistent clique tree for Example Network #1 (*cve-tree*)

The rules in R^* are then absorbed into R^+ one by one.

$$R'^{+} = absorb(R^+, \langle D = false, \begin{array}{|c|c|}\hline E & Value \\ \hline true & 0.0684 \\ false & 0.2796 \\ \hline\end{array} \rangle)$$

$$= \{ \ \langle D = true, 0.652 \rangle , \langle D = false, \begin{array}{|c|c|}\hline E & Value \\ \hline true & 0.0684 \\ false & 0.2796 \\ \hline\end{array} \rangle \ \}$$

$$R''^{+} = absorb(R'^{+}, \langle D = true, \begin{array}{|c|c|}\hline E & Value \\ \hline true & 0.8 \\ false & 0.2 \\ \hline\end{array} \rangle)$$

$$= \{ \ \langle D = true, \begin{array}{|c|c|}\hline E & Value \\ \hline true & 0.652 * 0.8 = 0.5216 \\ false & 0.652 * 0.2 = 0.1304 \\ \hline\end{array} \rangle ,$$

$$\langle D = false, \begin{array}{|c|c|}\hline E & Value \\ \hline true & 0.0684 \\ false & 0.2796 \\ \hline\end{array} \rangle \ \}$$

$$R'''^{+} = absorb(R''^{+}, \langle E = false, \begin{array}{|c|c|c|}\hline D & F & Value \\ \hline true & true & 0.7 \\ true & false & 0.3 \\ false & true & 0.8 \\ false & false & 0.2 \\ \hline\end{array} \rangle)$$

$$= \{ \ \langle D = true \wedge E = true, 0.5216 \rangle ,$$

$$\langle D = true \wedge E = false, \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.1304 * 0.7 = 0.09128 \\ false & 0.1304 * 0.3 = 0.03912 \\ \hline\end{array} \rangle ,$$

$$\langle D = false \wedge E = true, 0.0684 \rangle ,$$

$$\langle D = false \wedge E = false,$$

F	$Value$
$true$	$0.2796 * 0.8 = 0.22368$
$false$	$0.2796 * 0.2 = 0.05592$

$\rangle$ }

We split the resulting rules into R_1 and R_2 corresponding to the contexts $D = true$ and $D = false$, respectively, and add the two sets of rules:

$$R_1 = \{ \ \langle E = true, 0.5216 \rangle, \langle E = false,$$

F	$Value$
$true$	0.09128
$false$	0.03912

$\rangle\}$

$$R_2 = \{ \ \langle E = true, 0.0684 \rangle, \langle E = false,$$

F	$Value$
$true$	0.22368
$false$	0.05592

$\rangle\}$

$$R_1 \oplus_g R_2 = \{ \ \langle E = true, 0.5216 + 0.0684 = 0.59 \rangle,$$

$$\langle E = false,$$

F	$Value$
$true$	$0.09128 + 0.22368 = 0.31496$
$false$	$0.03912 + 0.05592 = 0.09504$

$\rangle$ }

The rule base becomes:

$$R = R^- \cup (R_1 \oplus_g R_2) = \{ \ \langle E = true, 0.59 \rangle,$$

$$\langle E = false,$$

F	$Value$
$true$	0.31496
$false$	0.09504

$\rangle$,

$$\langle E = true,$$

F	$Value$
$true$	0.4
$false$	0.6

$\rangle$ }

Eliminating the variable E, we have:

$$R^- = \{\}$$

$$R^+ = \{\langle E = true, 0.59\rangle\}$$

$$R^* = \{\ \langle E = false,\ \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.31496 \\ false & 0.09504 \\ \hline\end{array}\ \rangle,$$

$$\langle E = true,\ \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.4 \\ false & 0.6 \\ \hline\end{array}\ \rangle\ \}$$

$$R_{\prime}^+ = absorb(R^+, \langle E = false,\ \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.31496 \\ false & 0.09504 \\ \hline\end{array}\ \rangle)$$

$$= \{\ \langle E = true, 0.59\rangle,\ \langle E = false,\ \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.31496 \\ false & 0.09504 \\ \hline\end{array}\ \rangle\ \}$$

$$R_{\prime\prime}^+ = absorb(R_{\prime}^+, \langle E = true,\ \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.4 \\ false & 0.6 \\ \hline\end{array}\ \rangle)$$

$$= \{\ \langle E = true,\ \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.59 * 0.4 = 0.236 \\ false & 0.59 * 0.6 = 0.354 \\ \hline\end{array}\ \rangle,$$

$$\langle E = false,\ \begin{array}{|c|c|}\hline F & Value \\ \hline true & 0.31496 \\ false & 0.09504 \\ \hline\end{array}\ \rangle\ \}$$

$$R_1 = \{ \ \langle TRUE, \begin{array}{|l|l|} \hline F & Value \\ \hline true & 0.236 \\ false & 0.354 \\ \hline \end{array} \rangle \}$$

$$R_2 = \{ \ \langle TRUE, \begin{array}{|l|l|} \hline F & Value \\ \hline true & 0.31496 \\ false & 0.09504 \\ \hline \end{array} \rangle \}$$

$$R_1 \oplus_g R_2 = \{ \ \langle TRUE, \begin{array}{|l|l|} \hline F & Value \\ \hline true & 0.236 + 0.31496 = 0.55096 \\ false & 0.354 + 0.09504 = 0.44904 \\ \hline \end{array} \rangle \ \}$$

After elimination, there is only one rule left. The probabilities of F are calculated to be $P(F = true) = 0.55096$ and $P(F = false) = 0.44904$.

Chapter 5

Empirical Evaluation of *cve-tree*

5.1 Experiment Setup

In order to evaluate our proposed algorithm, we constructed a number of parameterized classes of networks that display context specific independence. [1] We compare *cve-tree* in terms of the time and space required to do probability inference with *Hugin* (Jensen et al., 1990) and the query-based contextual variable elimination algorithm (Poole and Zhang, 2002).

The following parameters are used for building the random networks:

- n: the number of variables in the network
- s: the number of contextual splits (so there are $n + s$ generalized rules for the network in the initial rule base)
- p: the probability that a parent variable that is not in the context of a generalized rule will be included in the table of the generalized rule

[1] Note that we do not expect *cve-tree* to work well for random networks. *cve-tree* is designed for networks that display context specific independence. Thus, it is fair to compare the algorithm on these networks.

- *obs*: the number of observed variables

Given the n variables, a total ordering is imposed. For each variable, we build a decision tree whose leaves correspond to the contexts of the generalized rules of the variables. We randomly choose a leaf and a variable. If the variable is a possible split for the leaf (i.e. that variable is a predecessor in the total ordering of the variable that the leaf is for, and the context corresponding to the leaf had not committed to a value of that variable), we split the leaf on that variable. This process is repeated until we have a total of $n+s$ leaves. Then, for each leaf, we build a generalized rule using the leaf as the context. For the table of the generalized rule, we include the variable that the leaf is for, and each of its predecessor is included with probability p. The probabilities in the tables are assigned random numbers. Thus, the number of rules in the initial rule base is controlled by the parameter s, and the size of the tables is controlled probabilistically by p. We also randomly choose variables to be observed for the networks. The number of observed variables is controlled by the parameter *obs*.

The following parameters were used to make the random networks in our experiments:

- number of variables in the network: $n = 20$
- probability of including a predecessor variable in the table of the rule: $p = 0.2$
- number of splits: $0 \leq s \leq 10$
- number of observed variables: $0 \leq obs \leq 10$.

Each set of parameters (n, p, s) was used to make 10 different random networks and each network was accompanied with the 11 different observations (determined

by *obs*), for a total of 1210 random networks. Notice that $s = 0$ corresponds to networks that exhibit no contextual specific independence. All experiments were performed on an Intel Pentium 4, 2.0GHz processor with 256KB CPU cache and 1GB RAM, running RedHat Linux 7.2.

5.2 Space Comparisons

Figure 5.1 gives a comparison of the space required by *Hugin* and *cve-tree* to do probabilistic inference for the random networks described above. The space is measured by the combined size (the number of floating point values) of all the tables in the clique tree after the tree is consistent. For *cve-tree*, this is calculated by summing up all the table size in the rules for each clique. The *Hugin* architecture that we use in this experiment compacts the tables in the cliques if there are observed variables. This is done to make the comparison fairer and to not confound the saving by compacting with the saving by context.

A major advantage of *cve-tree* can be seen in the much smaller total table sizes that *cve-tree* requires compared to those required by *Hugin*. The amount of savings that *cve-tree* provides over *Hugin* depends on the amount of context specific independence that a particular network exhibits, the number of observed variables, and whether the observed variables appear in the contexts or in the tables. This savings of space can be substantial, especially for large networks that exhibit some context specific independence. Figures 5.2 to 5.5 show the space comparisons by the number of *splits* that the random networks were created with. The cause for the difference in space requirements between *cve-tree* and *Hugin* is that *Hugin* always multiplies the factors passed to a clique while *cve-tree* may just keep the factors without multiplying them. With no context specific independence present,

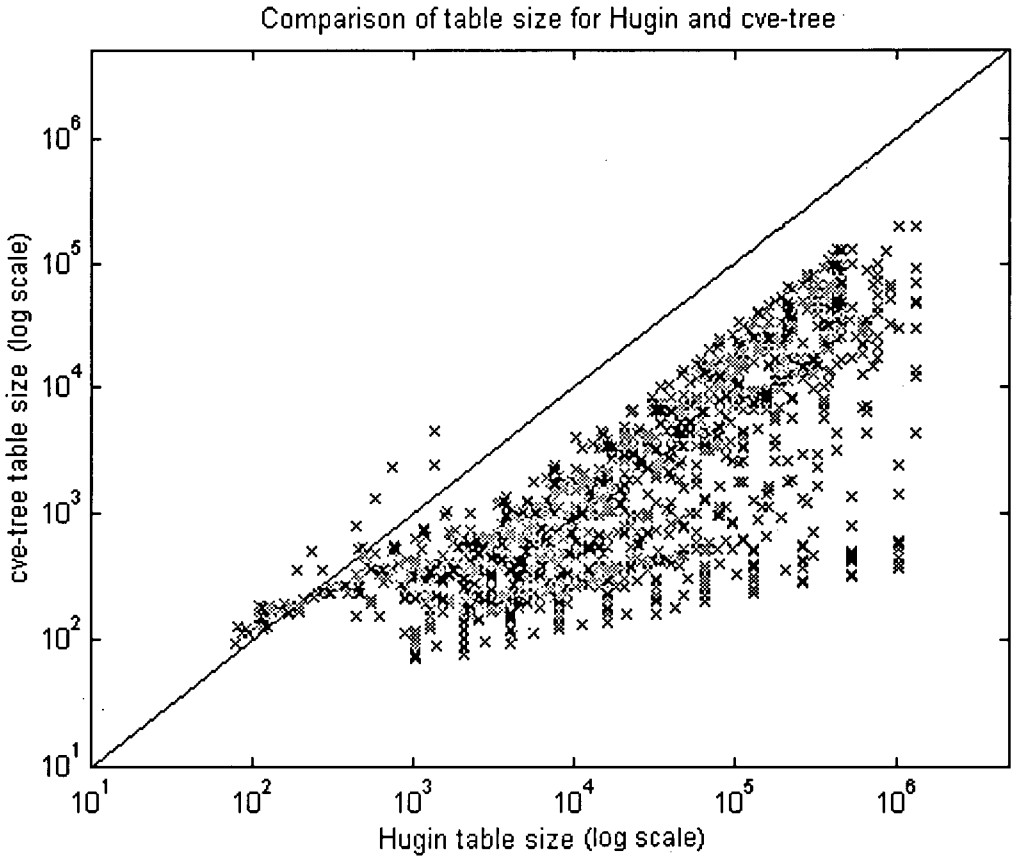


Figure 5.1: Comparison of table size of consistent clique trees

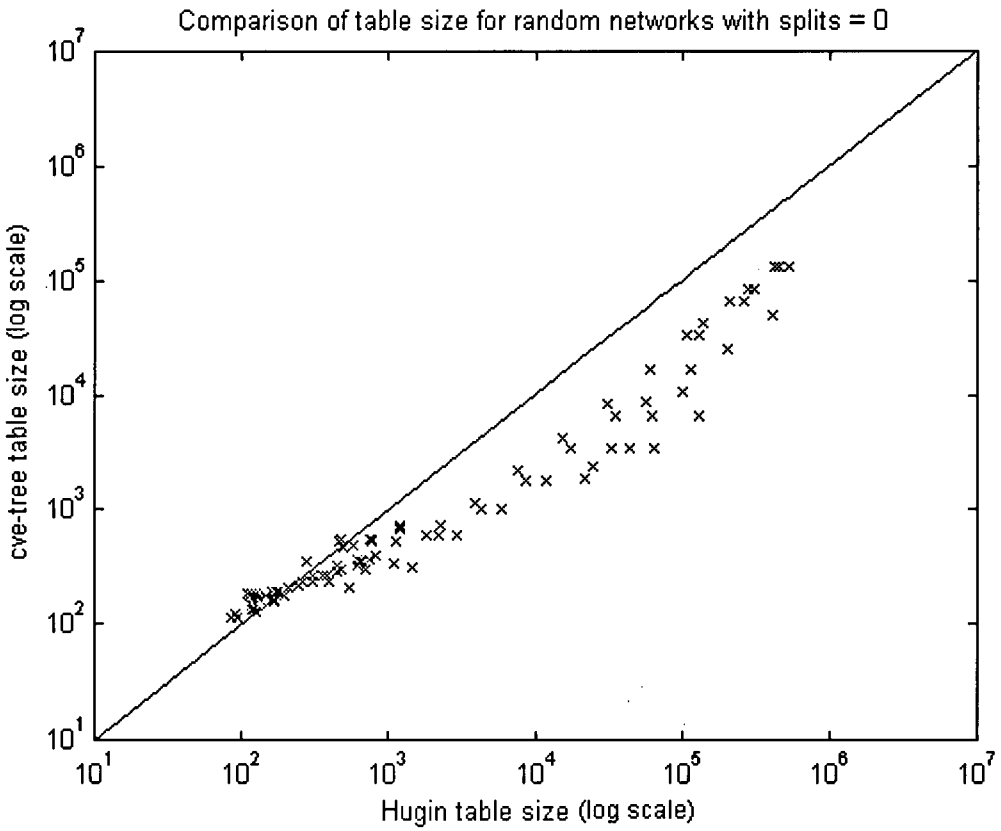


Figure 5.2: Comparison of table size of consistent clique trees from networks with $splits = 0$

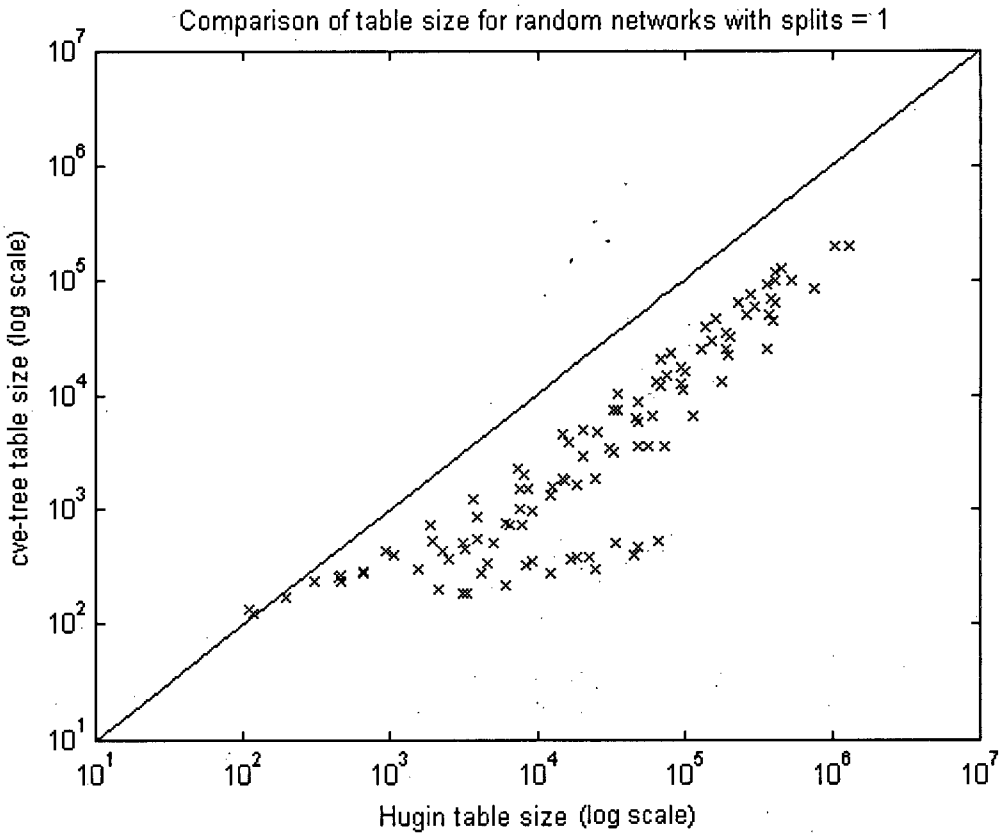


Figure 5.3: Comparison of table size of consistent clique trees from networks with $splits = 1$

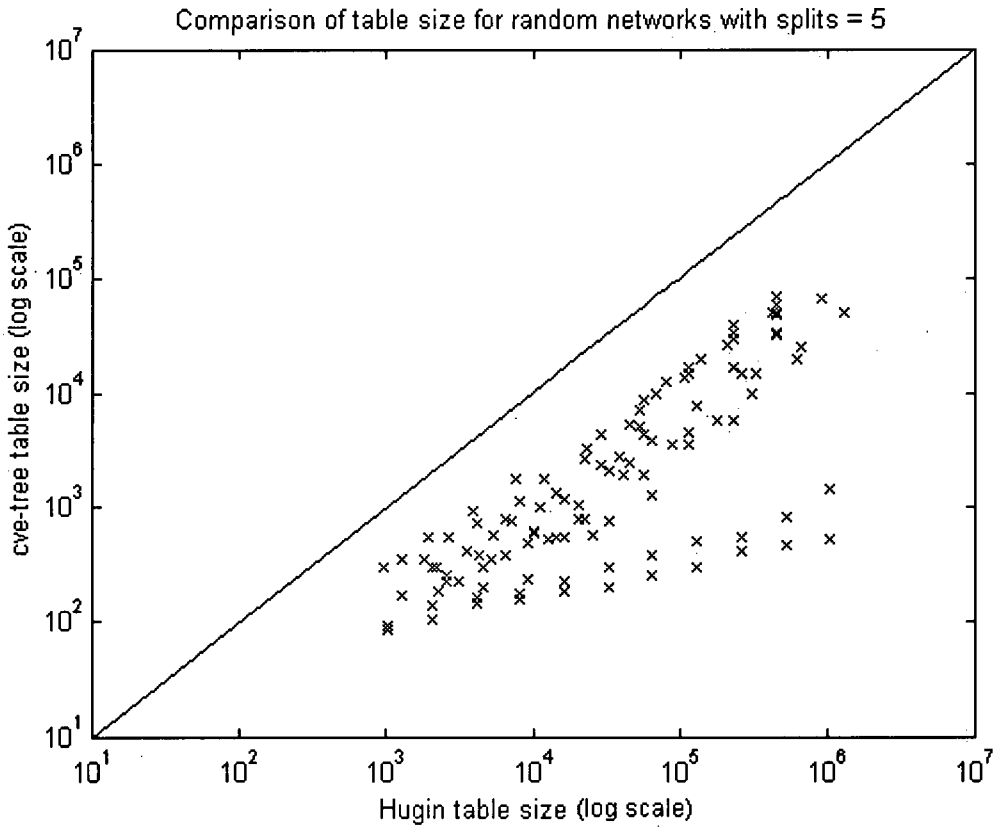


Figure 5.4: Comparison of table size of consistent clique trees from networks with *splits* = 5

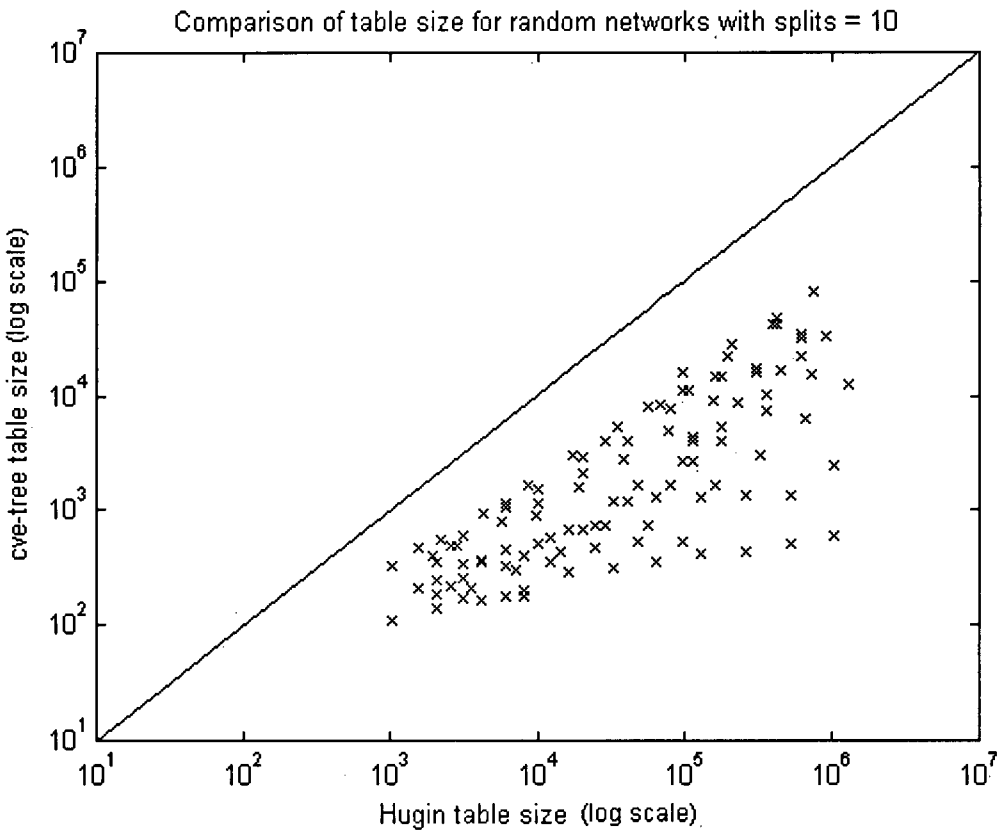


Figure 5.5: Comparison of table size of consistent clique trees from networks with $splits = 10$

Hugin may take less space than *cve-tree* or the other way around depending on the particular set of factors involved. For $splits = 0$ (Figure 5.2), even though the networks exhibit no context specific independence, the space savings for *cve-tree* over *Hugin* is statistically significant when analyzed with the *matched pairs t test.* With $splits = 1$ (Figure 5.3), the space required by *cve-tree* is less than *Hugin* for almost all networks. The difference in space is even more significant for networks with larger amounts of context specific independence. Figures 5.4 and 5.5 show the space difference for random networks with $splits = 5$ and $splits = 10$, respectively.

5.3 Time Comparisons

The entire process of clique tree propagation can be viewed as three stages: setting up the initial clique tree, propagating messages, and getting posterior probabilities. We compare the amount of time spent in these stages by the different algorithms as well as the total time they take to query 5 unobserved variables. Figures 5.6 to 5.9 show the distribution of the times for 3 algorithms: *Hugin*, *cve-tree*, and the query-based contextual variable elimination (*cve*) (Poole and Zhang, 2002), over the 1210 randomized networks as described above. Notice that these figures show the overall time distributions over the 1210 randomized networks, so the difference of the curves at a point may not correspond to the actual time difference for any particular network.

Figure 5.6 shows the time distribution for the *Hugin* and *cve-tree* algorithms to set up the initial clique tree for the 1210 random networks. *Hugin* takes a predominantly large amount of time at this stage because much of the work for this algorithm is done here. It involves initializing all the factors to the cliques by multiplying the conditional probabilities in the Bayesian networks and the observations

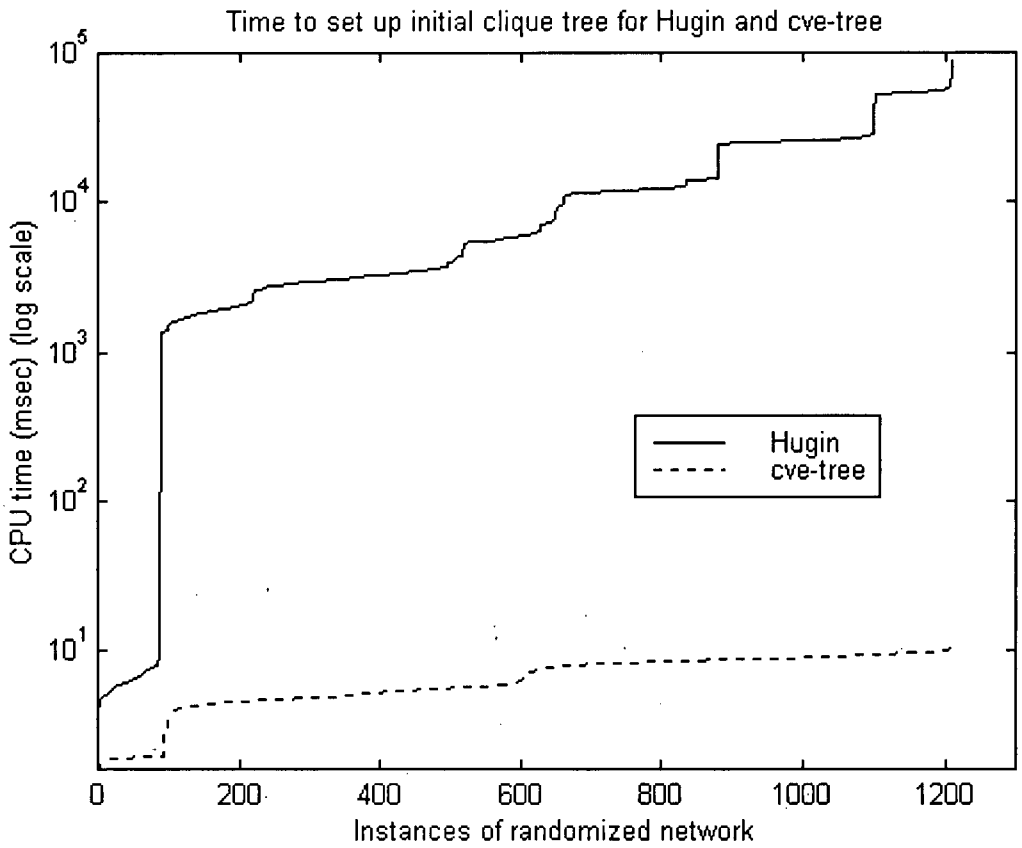


Figure 5.6: Distribution of time to set up the initial clique trees

into the corresponding cliques. On the other hand, *cve-tree* uses minimal time at this stage because all it does is to assign the rules in the initial rule base to the appropriate clique and update the rules with the observed variables; no table multiplication is performed. Note that this set up stage is not applicable to the query-based *cve* algorithm, as *cve* does not make use of a clique tree.

Figure 5.7 shows the time distribution for message propagation. The message propagation time is the time to do the two sweeps of message passing to make the clique tree consistent. Again, the message propagation time does not apply to the query-based *cve* algorithm. In general, for simpler networks, the message propagation time is higher for *cve-tree* than *Hugin*, mainly due to the overhead of the rule multiplications. But for more complicated networks with context specific independence, *cve-tree* in general takes less time than *Hugin* in the message propagation stage.

Figure 5.8 shows the distribution of the average time to get the probability of a query variable from a consistent clique tree for *Hugin* and *cve-tree*, or from the initial rule base for the query-based *cve* algorithm. This average time is obtained from summing up the time to get the posterior probability of each of the unobserved variables in the Bayesian network, divided by the number of unobserved variables. The time it takes for *Hugin* in this stage is very small comparatively, because all it needs to do is summing out the non-query variables from the marginal probability table contained in the clique. *cve-tree* takes a much longer time to get the probability, because it has to multiply the rules in a clique and eliminate the non-query variables from these rules. The overhead for maintaining variables in the contexts and splitting rules also plays a factor in the time it takes to get the probabilities. Since the query-based *cve* algorithm does all the work at this stage, its average time

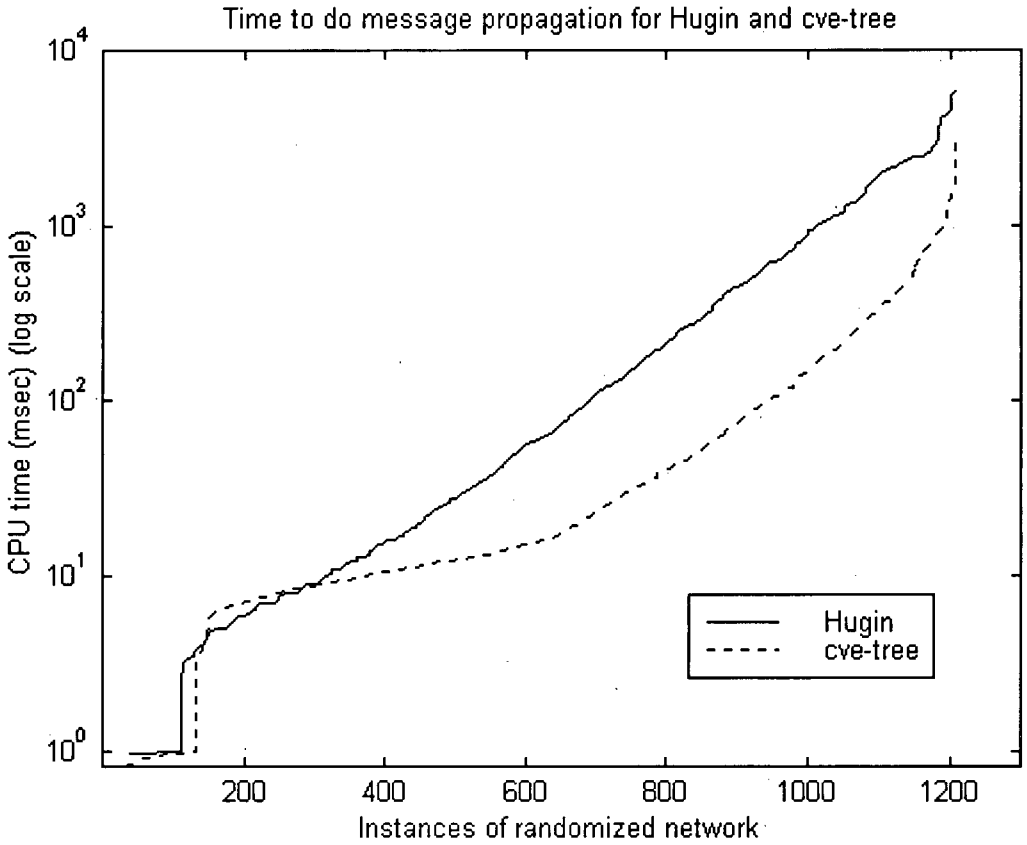


Figure 5.7: Distribution of message propagation time

to get the probability is higher than the other two algorithms.

As each algorithm does most of its work at different stages of the algorithm, it is difficult to compare which one is the most efficient. It all depends on the application, such as how many query variables there are and how much space is available to do the inference. If only one variable is to be queried, then the query based *cve* algorithm should be the choice because there is no point for setting up the clique tree that does probability inference for all the variables in the network simultaneously. However, if more than two variables are to be queried, the algorithm of choice would depend on the structure of the networks and the amount of context specific independence (and other additional structures) that can be exploited, as well as the physical constraints such as memory size.

Figure 5.9 shows the distributions of the average time that the 3 algorithms take to calculate the posterior probability of 5 unobserved variables for the 1210 randomized networks. This time is obtained by adding the set up time for the initial clique tree, the message propagation time, and 5 times the average get probability time. It is very clear from the figure that for most of the randomized networks, *cve-tree* does much better than *Hugin* when context specific independence appears in a Bayesian network. This is especially apparent for the more complicated networks for which *Hugin* takes a relatively long time to do inference. The distribution for *cve-tree* and *cve* are very close to each other. It appears that the query-based *cve* does better for simpler networks which takes less time to do inference, while *cve-tree* is better for more complicated networks. For simpler networks, the time taken by *cve-tree* to build the clique tree and make the clique tree consistent dominates. However, for the complicated networks, *cve-tree* only needs to do all the hard work once in the clique tree set up and message propagation stages, while *cve* must repeat

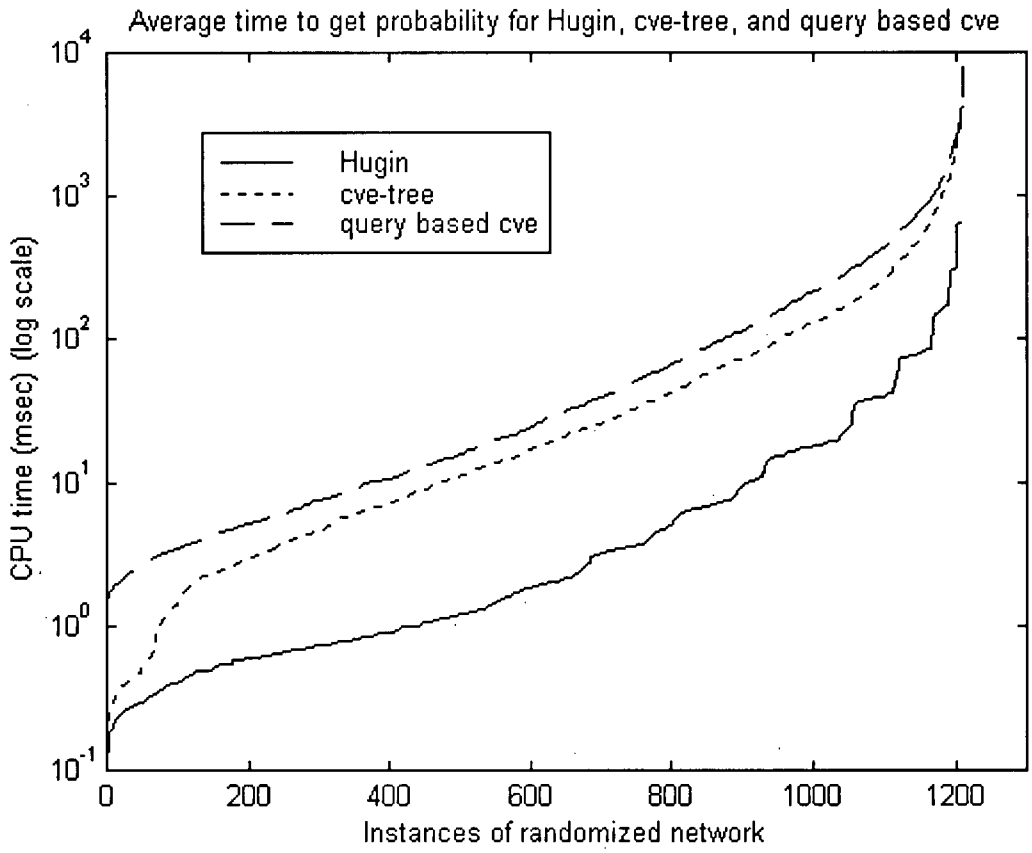


Figure 5.8: Distribution of the average time to get the probability of a query variable from a consistent clique tree

the process for each query.

Figures 5.10 to 5.13 show the amount of time that the *Hugin*, *cve-tree* and query-based *cve* algorithms take to query 5 unobserved variables for some individual random networks. In Figure 5.10, where the random networks do not exhibit context specific independence, *Hugin* does a little better than *cve-tree* for 5 of the networks, and does worse for the other 5. It seems that for networks that require relatively more time for *Hugin* (more than 10^3 *msec*), *cve-tree* is faster than *Hugin*, but for networks that require relatively less time for *Hugin* (less than 10^2 *msec*), *cve-tree* is a little slower than *Hugin*. This is because the networks that *Hugin* takes longer corresponds to clique trees that contain cliques and factors of many variables. Multiplication of these large factors is time costly. On the other hand, in *cve-tree*, the rules are not multiplied unless it is necessary. In addition, the time is reduced significantly for *cve-tree* and query-based *cve* when variables are observed, especially for those networks that take a long time to do inference. This is because the number of rules as well as the size of the rules are reduced with each variable being observed. For *Hugin*, the time reduction is not as significant, if it is reduced at all. The reason is that when variables are observed, the factors that contain those variables would have the corresponding entries changed to 0. The time that it takes to multiply the factors cannot be reduced as significantly. For the random networks with $splits > 0$, the graph of the time that the algorithms take to query 5 unobserved variables look very similar. Figures 5.11, 5.12, and 5.13 show the results for $splits = 1, 5$, and 10, respectively. In most cases, *cve-tree* and query-based *cve* are a lot faster than *Hugin*. The time difference is in general larger for a larger number of splits. Nevertheless, even when only a small amount of context specific independence is present in the network, *cve-tree* can take advantage of it and wins over *Hugin*.

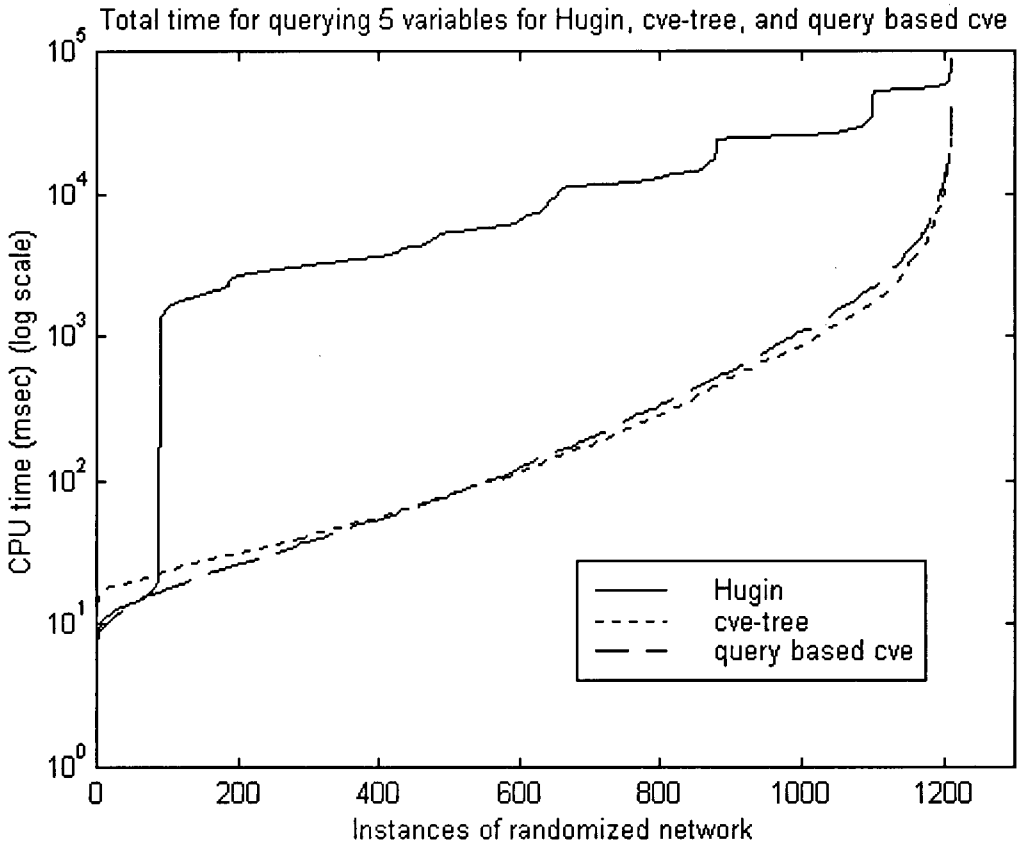


Figure 5.9: Distribution of the total time for calculating the probability of 5 unobserved variables

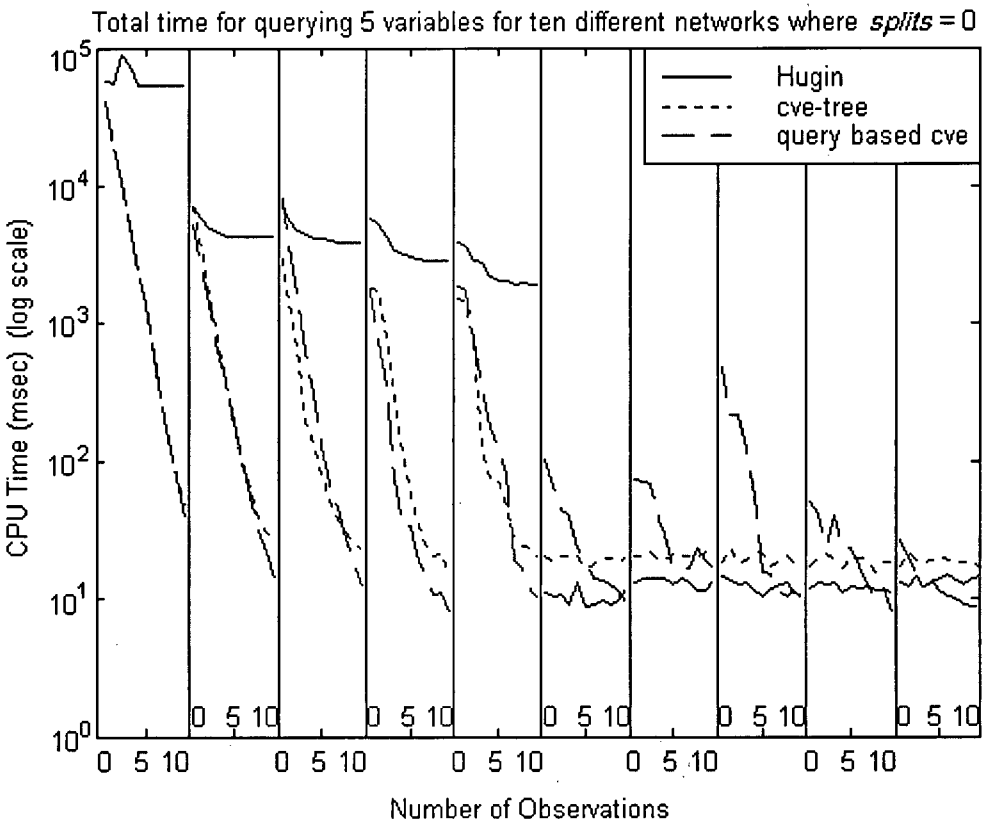


Figure 5.10: Total time for calculating the probability of 5 unobserved variables for 10 random networks with $splits = 0$ (ie. no context specific independence)

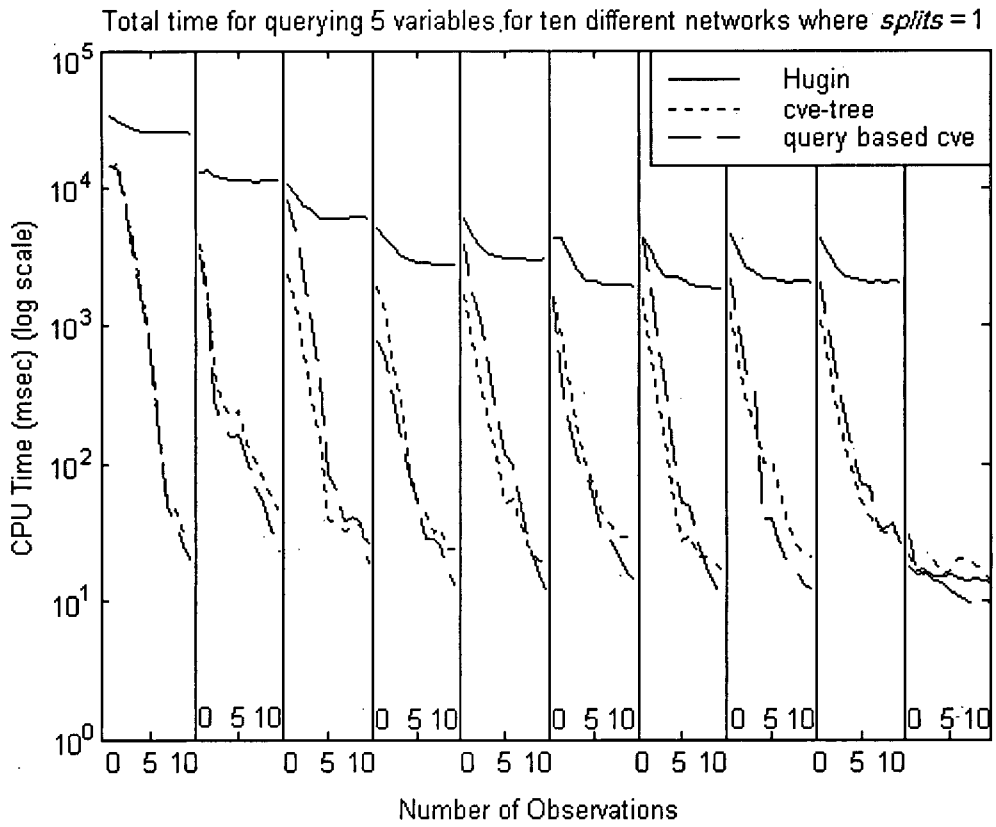


Figure 5.11: Total time for calculating the probability of 5 unobserved variables for 10 random networks with *splits* = 1

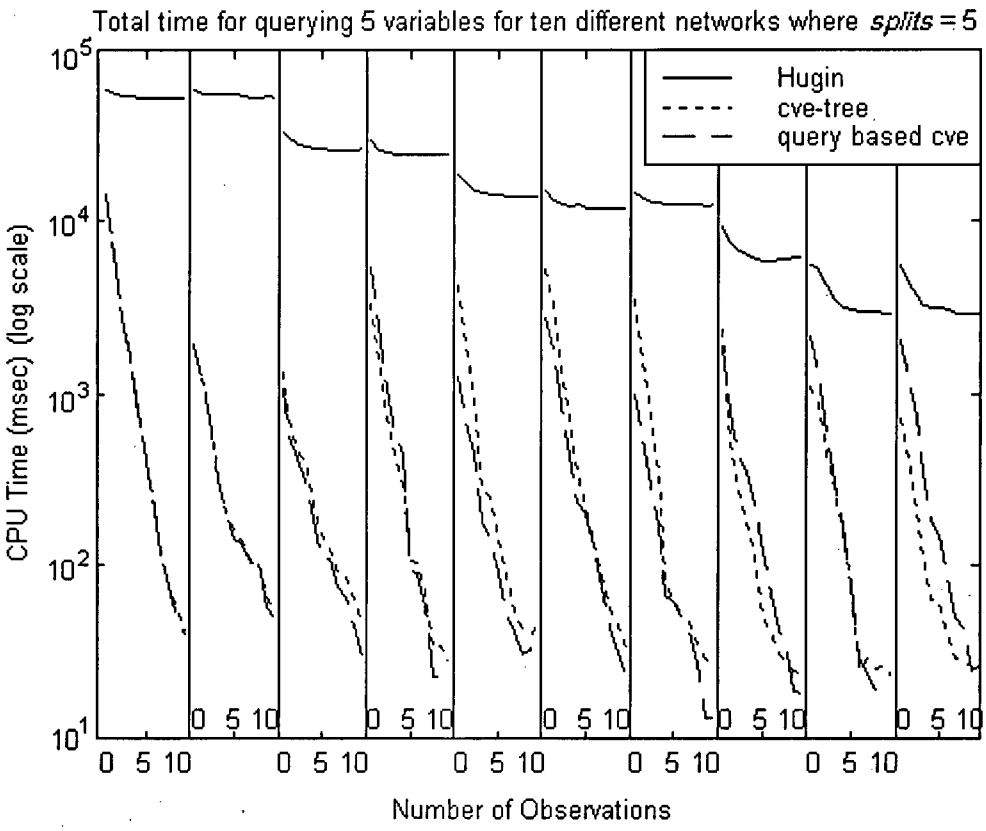


Figure 5.12: Total time for calculating the probability of 5 unobserved variables for 10 random networks with $splits = 5$

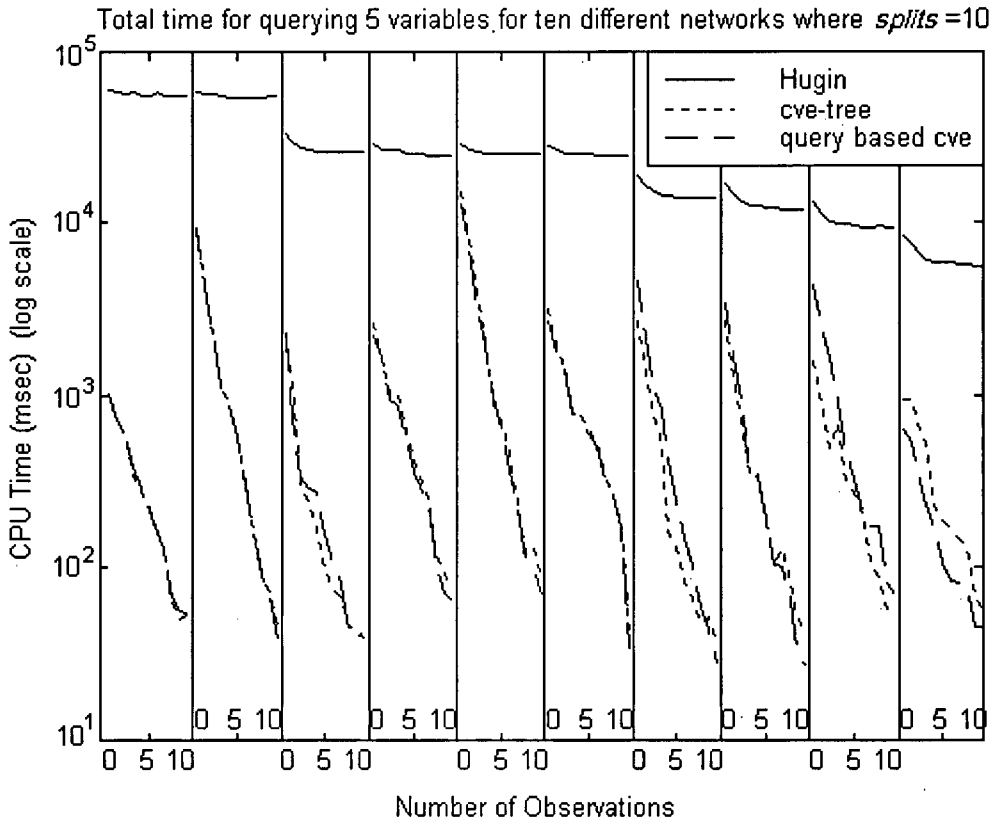


Figure 5.13: Total time for calculating the probability of 5 unobserved variables for 10 random networks with $splits = 10$

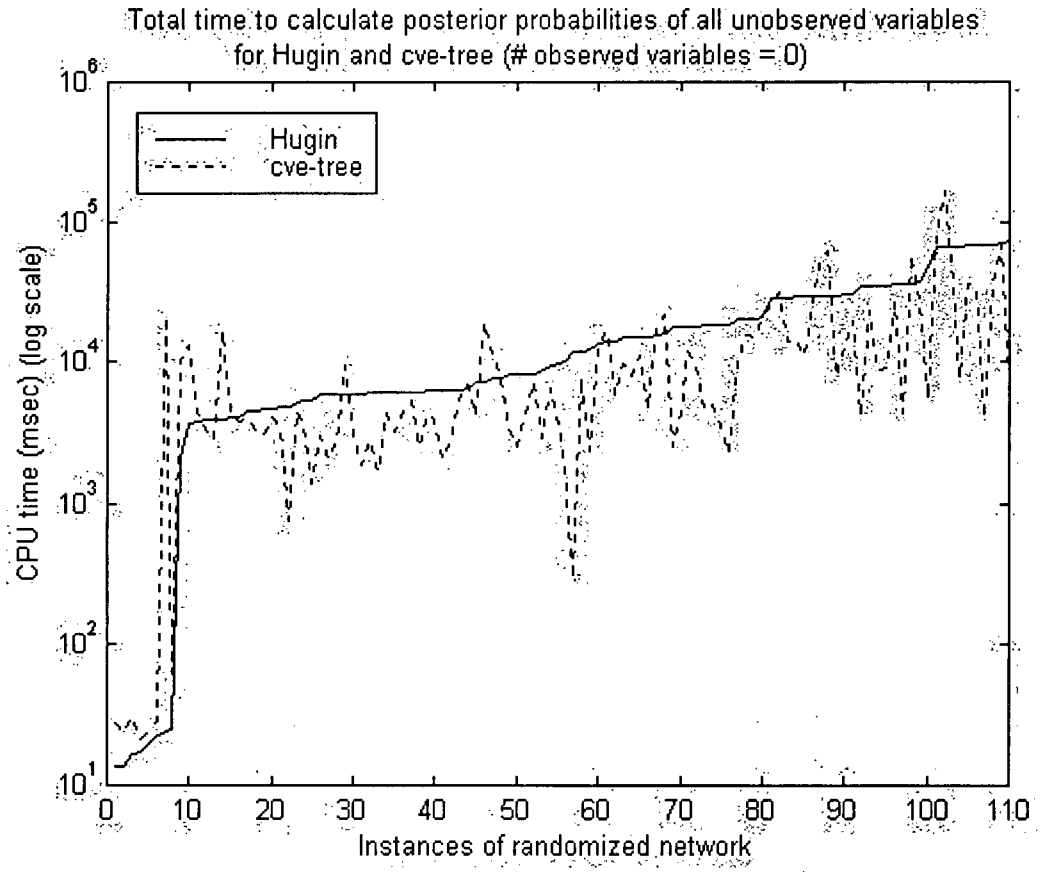


Figure 5.14: Total time for calculating the probabilities of all of the 20 unobserved variables for random networks with no observed variables

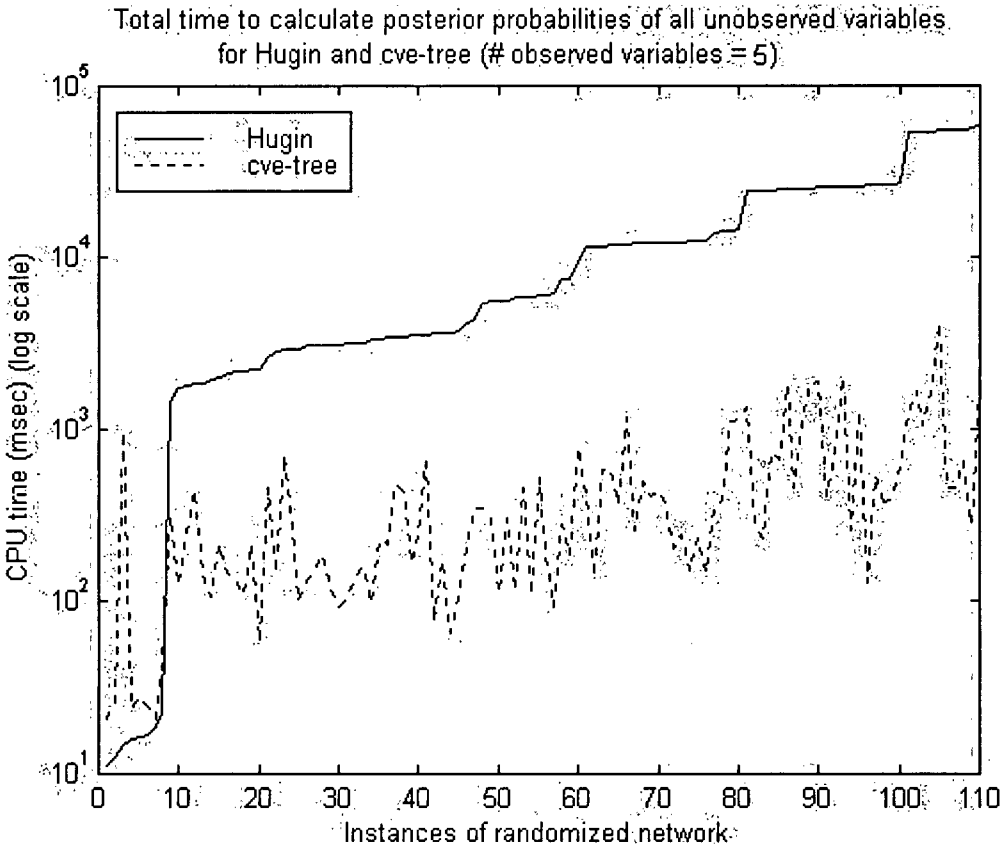


Figure 5.15: Total time for calculating the probabilities of all of the 15 unobserved variables for random networks with 5 observed variables

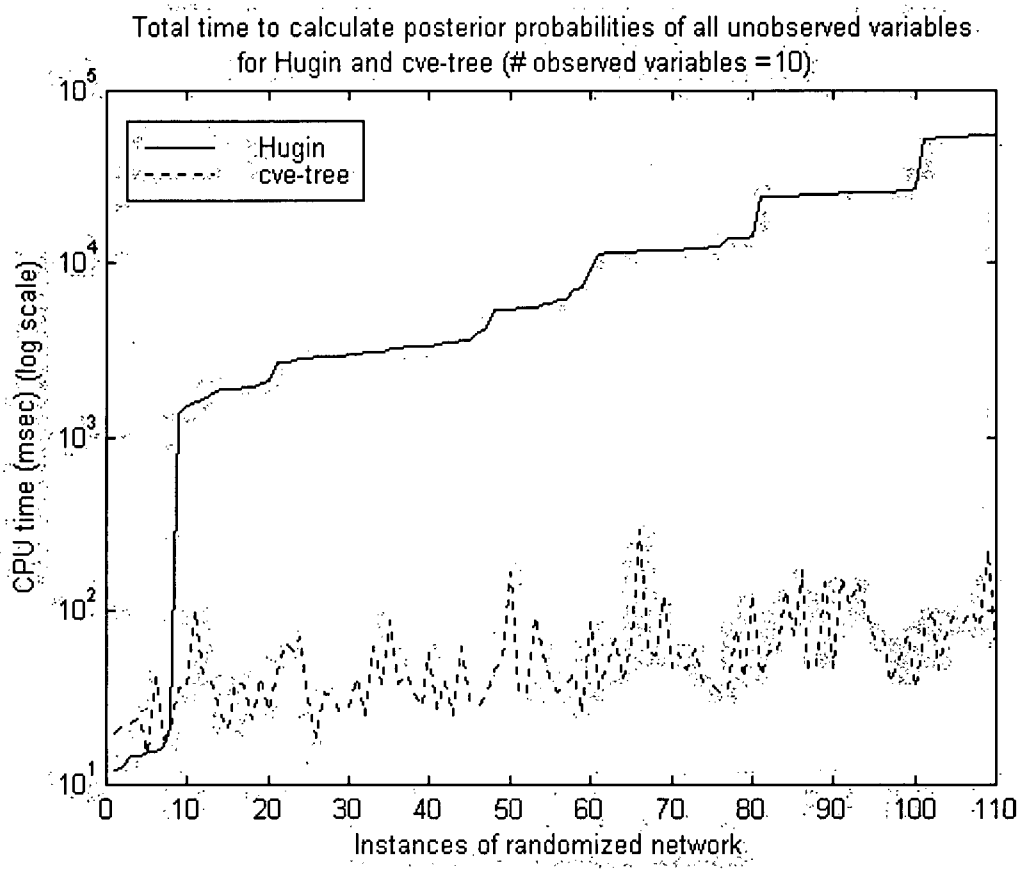


Figure 5.16: Total time for calculating the probabilities of all of the 10 unobserved variables for random networks with 10 observed variables

Figures 5.14 to 5.16 show the CPU time for *Hugin* and *cve-tree* to query all the unobserved variables in the random networks. Figure 5.14 shows the result for networks with no observation, corresponding to querying all 20 variables in the networks. *cve-tree* wins over *Hugin* for a majority of the cases. The networks for which *cve-tree* takes more time than *Hugin* are networks with little or no context specific independence. Figure 5.15 shows the result for networks with 5 variables observed, corresponding to querying the remaining 15 variables. Aside from a few networks that take a particularly little amount of time for *Hugin* (less than 10^2 *msec*), *cve-tree* is significantly faster than *Hugin* for all the other networks, including those that do not display context specific independence. The same result is seen for networks with 10 observed variables, as shown in Figure 5.16. As the number of observed variables increases, querying all unobserved variables' posterior probabilities for the two algorithms results in an even larger time difference. Our experiments show that the *cve-tree* alogrithm not only takes advantage of context specific independence in the networks, but also increases its efficiency as variables in the networks are observed.

5.4 The *Insurance* Network

In addition to the randomized networks, we experimented with the *insurance* network, a network for evaluating car insurance risks, from the Bayesian network repository at the Hebrew University of Jerusalem. [2] The *insurance* network consists of 27 variables, each with 2 to 5 domain values. Since the *insurance* network does not contain any context specific independence, we created modified versions of the *insurance* network by changing the numbers in the probability tables using *precision values*, as defined below. Two numbers in a probability table are considered equal if

[2] http://www.cs.huji.ac.il/labs/compbio/Repository/

their difference is less than the *precision value*. For example, if the precision value 0.1 is used, the table:

			$P(D\|A,B,C)$	
A	B	C	$true$	$false$
$true$	$true$	$true$	0.75	0.25
$true$	$true$	$false$	0.8	0.2
$true$	$false$	$true$	0.6	0.4
$true$	$false$	$false$	0.55	0.45
$false$	$true$	$true$	0.1	0.9
$false$	$true$	$false$	0.2	0.8
$false$	$false$	$true$	0.35	0.65
$false$	$false$	$false$	0.6	0.4

is split into the three rules:

$\langle A = true \wedge B = true,$

D	$Value$
$true$	0.775
$false$	0.225

$\rangle$

$\langle A = true \wedge B = false,$

D	$Value$
$true$	0.575
$false$	0.425

$\rangle$ }

$\langle A = false,$

B	C	D	$Value$
$true$	$true$	$true$	0.1
$true$	$true$	$false$	0.9
$true$	$false$	$true$	0.2
$true$	$false$	$false$	0.8
$false$	$true$	$true$	0.35
$false$	$true$	$false$	0.65
$false$	$false$	$true$	0.6
$false$	$false$	$false$	0.4

$\rangle$

The use of precision values generates context specific independence within the network that we can use to experiment with the *cve-tree* algorithm. Precision values

of 0.05, 0.1, 0.15, and 0.2 were applied to the *insurance* network in our experiment. Since no sensitivity analysis (Chan and Darwiche, 2001; Kjærulff and van der Gaag, 2000) has been done for these modifications, we do not expect any accuracy from the approximations. We are only interested in how this kind of approximations can be used with our algorithm to improve the efficiency of probability inference.

Table 5.1 summarizes the rule bases of the *insurance* network (corresponding to a precision value of 0) and its approximations using precision values 0.05, 0.1, 0.15, and 0.2. The total time it takes to do inference for these networks and query 5 unobserved variables with *Hugin* and *cve-tree* are shown in Figure 5.17, where the number of observed variables ranges from 0 to 13. Given the same observed variables, the amount of time *Hugin* takes for the four networks is very similar. On the other hand, *cve-tree* performs differently with different precisions, which correspond to different amounts of context specific independence. As the figure shows, *cve-tree* is most efficient with a precision of 0.15 for this particular network. As table 5.1 shows, the network with precision value 0.2 has 7 more rules than the network with precision value 0.15, but the saving in the total number of entries in the rules' tables is only 11. This makes the overhead of maintaining the extra rules larger than the benefit obtained from the smaller table sizes. Thus, it takes more time to do inference with the network with precision value 0.2 than the network with precision value 0.15. As this experiment shows, not all splittings of probability tables into rules are beneficial to the inference time. It depends on the reduction in table sizes compared to the increase in the number of rules in the network.

Table 5.1: Summary of the *insurance* network and its approximations in terms of context specific independence and rule and table sizes

Precision Value	Number of Variables with Context Specific Independence	Total Number of Rules in Network	Total Number of Entries in the Rules' Tables
0	1	30	1407
0.05	2	34	1393
0.1	5	42	1339
0.15	6	43	1291
0.2	8	50	1280

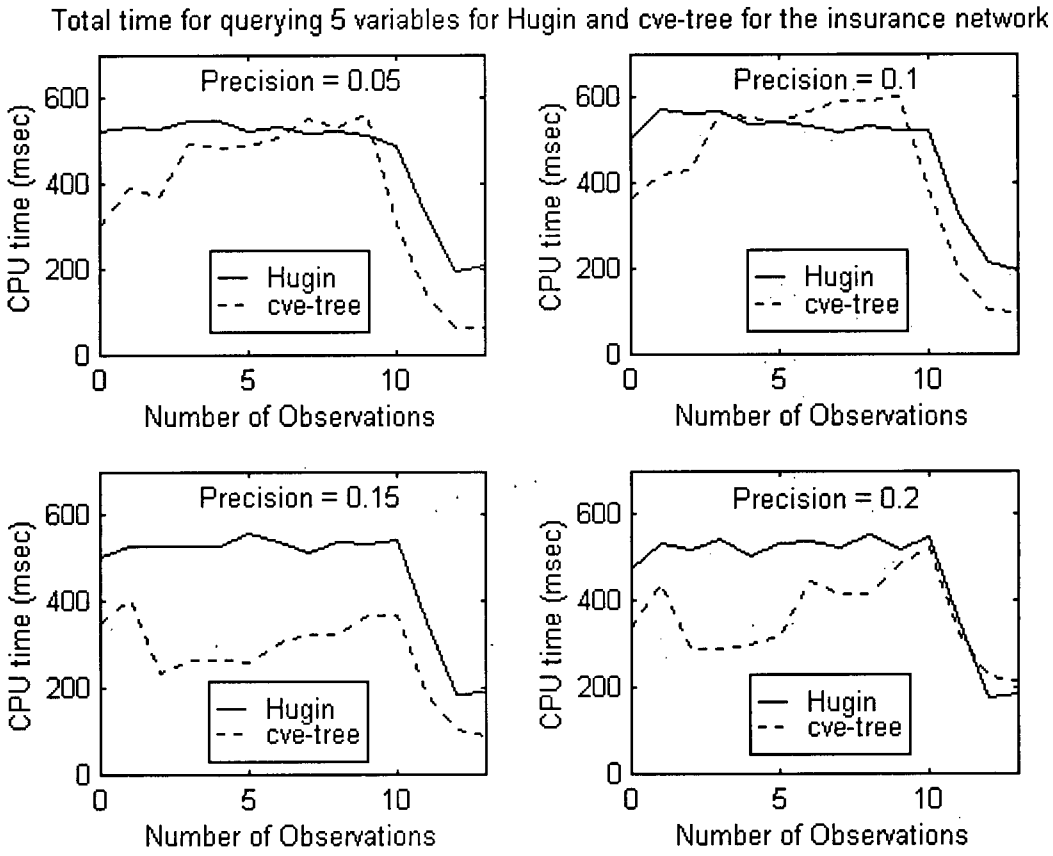


Figure 5.17: Comparison of the total time required to calculate the probability of 5 unobserved variables for 4 approximations of the *insurance* network

Chapter 6

Conclusion and Future Work

We have presented *cve-tree*, a clique tree propagation algorithm for exploiting context specific independence to compute posterior probabilities in Bayesian networks. With our experiments on randomized networks that exhibit various amounts of context specific independence and the *insurance* network, our algorithm shows advantages both in time and space over the *Hugin* architecture when some level of context specific independence appears in the Bayesian networks.

Although the *cve-tree* algorithm as described in this thesis uses the clique tree structure of the *Hugin* architecture, other clique tree structures such as the *Shenoy-Shafer* architecture (Shafer and Shenoy, 1990) can also be employed. The efficiency of *cve-tree* on other clique tree structures would be further explored in future research since the efficiency also depends on the clique tree used to carry out inference.

The combination of *cve-tree* with sensitivity analysis (Chan and Darwiche, 2001; Kjærulff and van der Gaag, 2000) could be explored in future research as an application for approximate inference. Sensitivity analysis techniques can be used to distinguish which probabilities in a Bayesian network are close enough to be

considered independent in the specific contexts while keeping the desired accuracy in the results. The combination of *cve-tree* with sensitivity analysis would require a cost-benefit analysis to determine if it is worthwhile to split a probability table into rules by trading off accuracy with time and space efficiency. The resulting Bayesian network would contain context specific independence that *cve-tree* can take advantage of in doing probability inference.

The main overhead of *cve-tree*, as compared to *Hugin*, is the more complicated methods for multiplying rules and eliminating variables. The amount of time needed for probabilistic inference directly depends on the number of rules in the network. Because of the overhead of rule maintenance, splitting a probability table into a set of rules that results in only a small reduction in table sizes may hinder the inference procedure rather than speeding it up. We would like to explore a heuristics that determines whether a probability table with context specific independence should be split into a set of rules with smaller tables or should be left as a single rule with the entire table. A reduction in table sizes would decrease the space and time in doing inference, whereas an increase in the number of rules would increase the overhead costs and the time to do inference. The heuristics for splitting a rule into a set of rules with smaller tables would be based on the reduction in table sizes as compared to the increase in the number of rules that result from the splitting.

Bibliography

Boutilier, C., Friedman, N., Goldszmidt, M. and Koller, D. (1996). Context-specific independence in Bayesian networks, *in* E. Horvitz and F. Jensen (eds), *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-96)*, Portland, OR, pp. 115–123.

Chan, H. and Darwiche, A. (2001). When do numbers really matter?, *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-01)*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 65–74.

Cooper, G. (1987). Probabilistic inference using belief networks is NP-hard, *Technical report*, Medical Computer Science Group, Stanford University.

Dechter, R. (1996). Bucket elimination: A unifying framework for probabilistic inference, *in* E. Horvitz and F. Jensen (eds), *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-96)*, Portland, OR, pp. 211–219.

Geiger, D. and Heckerman, D. (1996). Knowledge representation and inference in similarity networks and Bayesian multinets, *Artificial Intelligence* **82**: 45–74.

Heckerman, D. and Breese, J. (1994). A new look at causal independence, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pp. 286–292.

Huang, C. and Darwiche, A. (1996). Inference in belief networks: A procedural guide, *International Journal of Approximate Reasoning* **15**(3): 225–263.

Jensen, F. V., Lauritzen, S. L. and Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations, *Computational Statistics Quarterly* **4**: 269–282.

Kjærulff, U. (1990). Triangulation of graphs - algorithms giving small total state space, *Technical Report R 90-09*, Department of Mathematics and Computer Science, Strandvejen, DK 9000 Aalborg, Denmark.

Kjærulff, U. and van der Gaag, L. (2000). Making sensitivity analysis computationally efficient, *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 317–325.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems, *Journal of the Royal Statistical Society, Series B* **50**(2): 157–224.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA.

Poole, D. and Zhang, N. (2002). Exploiting contextual independence in probabilistic inference, *Technical report*, Department of Computer Science, University of British Columbia.

Shafer, G. and Shenoy, P. (1990). Probability propagation, *Annals of Mathematics and Artificial Intelligence* **2**: 327–352.

Zhang, N. (1998). Inference in Bayesian networks: The role of context-specific independence, *Technical Report HKUST-CS98-09*, Department of Computer Science, Hong Kong University of Science and Technology.

Zhang, N. and Poole, D. (1994). A simple approach to Bayesian network computations, *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, Banff, Alberta, Canada, pp. 171–178.

Zhang, N. and Poole, D. (1996). Exploiting causal independence in Bayesian network inference, *Journal of Artificial Intelligence Research* **5**: 301–328.

Zhang, N. and Poole, D. (1999). On the role of context-specific independence in probabilistic reasoning, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, pp. 1288–1293.

Zhang, N. and Yan, L. (1997). Independence of causal influence and clique tree propagation, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 481–488.