# Processing of Inference Queries in Probabilistic Databases

V.Arthi[1], V.P.Sumathi[2]

**Abstract**

Real world applications like sensor network monitoring, that deal with wide existence of uncertain factors, employ relational databases to describe the probability distribution of all variables in its environment. Such probability distribution data finds extensive usage through inference queries. In practice, however, rather than a single probabilistic inference query, applications pose multiple but usually similar probabilistic interference queries to the system. An environment that involves frequent inference queries on relational databases provides a possibility of applying 'Computation sharing' logic among different queries. CTP is introduced in databases for probabilistic inference queries. Such an approach provides two opportunities for computation sharing. First opportunity is an existence of many common variables that needed to be eliminated during the evaluation of different queries. Second opportunity refers to the variables appearing frequently in the queries that can be cached and reused in later queries. The materialized views are used to cache the intermediate results of the previous inference queries, which might be shared with the following queries, and consequently reduce the time cost. When a similar query comes and requests the elimination of the same variables, the inference query can reuse these cached materialized views to avoid re-eliminating the common variables again. Variables that appear frequently in the query workload can be efficiently processed by caching and reusing the intermediate computation results. The work focuses on how a sequence of Probabilistic Inference queries can be efficiently processed with the application of computation sharing logic, which can highly optimize query performance.

*Keywords*: Probabilistic inference, Variable elimination, Clique tree propagation, Query optimization

## I. Introduction

Data uncertainty becomes a popular topic in database and data mining communities due to the wide existence of uncertainty in many real applications, such as sensor network monitoring, object identification, location-based services (LBS), and moving object tracking. In these applications, databases are often employed to describe the probability distribution of all variables X in the system. Due to the intrinsic property of uncertainty, many interesting queries have been raised for different purposes. Among them, probabilistic inference queries are frequently used, e.g., in decision support systems [3]. Formally, an inference query is to compute the marginal probability distribution P(Q) of a subset variables $Q \subseteq X$ from the probability distribution of all the variables. For example, consider a database storing the probability distribution of all sensors (variables) in a sensor network. An inference query is to compute the marginal probabil-

[1]*Research Scholar, Department of Computer Science, Kumaraguru College of Technology, Coimbatore.*
*Email: arty2406@gmail.com*

[2]*Assistant Professor, Department of Computer Science, Kumaraguru College of Technology, Coimbatore.*
*Email: sumathi_subbu2001@yahoo.co.in*

ity distribution of some sensors A and B, i.e. P(A,B); or given the value of A=a, to compute the marginal probability of B, i.e., P(B|A=a) in order to know the effect of A (e.g., humidity) on B (e.g., temperature). A straightforward approach to evaluate an inference query is conducting a relational query with group by operation. However, this approach has to join all the joinable tables together first, which is not efficient. To improve the evaluation efficiency, a variable elimination algorithm in Bayesian networks was implemented in relational databases [3]. In fact, the Bayesian network itself is widely used as a compact representation of the probability distribution of all the variables [6] and it can be naturally represented and stored in relational databases [8]. Selecting a suitable set of views that minimizes the total cost associated with the materialized views and is the key component in data warehousing. [7] gives the results of proposed tree based materialized view selection algorithm for query processing. In distributed environment where database is distributed over the nodes on which query should get executed and also plays an important role. One simple criterion would be to select a set of materialized view that minimizes the overall execution time of the workload of queries. Materializing a view causes it to be refreshed every time a change is made to the base tables that it references. It can be costly to rematerialize the view each time a change is made to the base tables that might affect it. So it is desirable to propagate the changes incrementally. Therefore, in this paper, we mainly focus on probabilistic inference problems over the probability distribution of all the variables represented by Bayesian networks in databases.

In practice, however, rather than a single probabilistic inference query, users may frequently pose multiple probabilistic interference queries to the system. For example, again in the sensor networks, a sequence of inference queries are often posed to continuously monitor data distributions in different areas. The above discussion indicates that compared to single probabilistic inference query, efficiently answering multiple frequent inference queries is more practical and useful. However, previous works on probabilistic inference in databases seldom address the issues of optimizing the computation sharing among different queries. [8][9] provide a framework of implementing probabilistic inferences in relational databases, but their work do not address the efficiency issue with respect to large-scale databases.[3] presents a broad class of aggregate queries, called Marginalize a Product Function (MPF), and implement the variable elimination (VE) algorithm in relational databases. Specifically, the VE algorithm optimizes the inference query by pushing down the aggregates in the joining tree of the whole inference query. However, the VE algorithm has no computation sharing among different queries. Even when a similar inference query, e.g., P(A), arrives after another query, e.g., P(A,B), in a sequence of inference queries, the VE-based approach has to reconstruct the query for the new P(A). To summarize, previous works for answering a single probabilistic inference query in databases is not efficient for a sequence of queries since it does not support computation sharing among different queries.

In fact, when a sequence of inference queries are posed, there exist two opportunities of computation sharing among the evaluation of different queries. Recall that the inference query is to compute the marginal probability of subset variables $Q \subseteq X$ from the probability distribution of all variables X, that is, to eliminate variables X - Q from X. Thus, the first opportunity is that there may exist many common variables needed to be eliminated during the evaluation of different queries. The computation of eliminating these common variables (not included in the queries) can be cached and shared among the queries. The second opportunity refers to the variables appearing frequently in the queries. We can cache the query computation spent on these frequent query variables for the possible reusing by later queries. Corresponding to the two opportunities, there exist two challenges to answer frequent inference queries efficiently, which are 1) how to detect and organize the elimination of common variables with respect to X - Q in relational databases, which enables reusing among different inference queries. 2) How to optimize the inference queries by further reusing these frequently queried variables in regards to Q. Motivated by the challenges of sequences of probabilistic inference queries, in this paper, we study the computation caching and sharing among different inference queries in relational databases. In order to share the computation of eliminating common variables in X - Q of different queries, we study the approach of treating inference as message propagation. The *clique tree propagation* (CTP) [10], also known as junction tree propagation , is based on the same principle as VE except with a sophisticated caching strategy. We implement CTP in relational databases as follows: The results of eliminating the common variables in X - Q are cached as materialized views in relational databases. When a similar query comes and requests the elimination of the same variables, the inference query can reuse these cached materialized views to avoid re-eliminating the common variables again. Moreover, with respect to the frequent queried variables in Q, we cache and reuse the intermediate computation results with query variables that appear frequently in the query workload. Those frequent variables in the workload have a high probability of appearing again in the subsequent queries according to workload statistics, and thus can reuse the cached results.

The correctness of a probabilistic inference on the probability distribution is discussed with cached query variables. By analysing the updating operations of cached variables, we further reduce the times of discarding the cached variables with high frequency.

We transform the message propagation in probabilistic inferences to joining tree queries in databases by using materialized views in relational databases. The paper evaluates the CTP as relational queries with materialized views. This approach enables the computation sharing for the current query $Q_{k+1}$ from the results of the previous query $Q_k$.

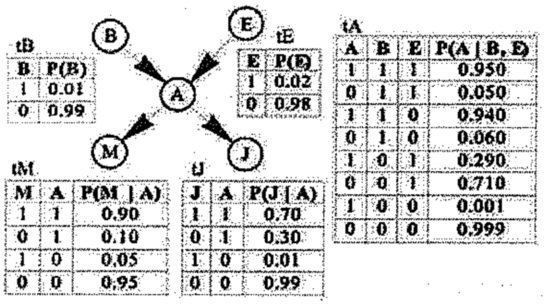
We explore the workload statistics to find the frequent variable(s) among different queries, and apply this workload information in the query optimization. This CTP caching approach optimizes the message propagation by caching the frequent query variables in the materialized views, and thus maximizes the computation reuse of the frequent query variables in a sequence of inference queries.

## II. Proposed System Model

Here a framework of probabilistic inference queries in relational databases is considered. We start with relational tables storing the Bayesian networks [6], which are widely used as a compact representation of the probability distribution of all the variables.

### A. Probabilistic Inference Query

Consider a set of discrete random variables $X = \{X_1 . . . X_n\}$. A Bayesian network [6] is a compact graphical representation for the joint distribution of all the variables. Specifically, a Bayesian network is a directed acyclic graph (DAG), where each node represents a random variable and is associated with a tableau of the conditional probabilities given its parents, called a *factor*. By conducting the product join of all the factors in a Bayesian network, we get the joint probability distribution $P(X_1 \ldots X_n)$ over all the variables. We show an example of Bayesian network in Fig. 1.Consider binary random variables B,E,A,J,M. Each node (a factor) is associated with a tableau in the figure, for example tA corresponding to a node A represents the conditional probabilities P(A|B,E) of A given variables B,E. The product join of all the factors is the joint distribution, ie,



tB

| B | P(B) |
|---|---|
| 1 | 0.01 |
| 0 | 0.99 |

tE

| E | P(E) |
|---|---|
| 1 | 0.02 |
| 0 | 0.98 |

tA

| A | B | E | P(A \| B, E) |
|---|---|---|---|
| 1 | 1 | 1 | 0.950 |
| 0 | 1 | 1 | 0.050 |
| 1 | 1 | 0 | 0.940 |
| 0 | 1 | 0 | 0.060 |
| 1 | 0 | 1 | 0.290 |
| 0 | 0 | 1 | 0.710 |
| 1 | 0 | 0 | 0.001 |
| 0 | 0 | 0 | 0.999 |

tM

| M | A | P(M \| A) |
|---|---|---|
| 1 | 1 | 0.90 |
| 0 | 1 | 0.10 |
| 1 | 0 | 0.05 |
| 0 | 0 | 0.95 |

tJ

| J | A | P(J \| A) |
|---|---|---|
| 1 | 1 | 0.70 |
| 0 | 1 | 0.30 |
| 1 | 0 | 0.01 |
| 0 | 0 | 0.99 |

Fig.1. Bayesian networks as relational database

$$P(B,E,A,J,M) = P(B)P(E)P(A|B,E)P(J|A)P(M|A)$$

Therefore, the Bayesian network is a compact representation of the joint distribution. In fact, the joint probability distribution in this example can be in the size of 25 tuples, while the Bayesian network representation needs only 20 tuples in all the factors. In this paper, we study probabilistic inference queries in Bayesian networks within relational database environments. A probabilistic inference is a process of computing marginal probability P(Q) to an inference query $Q \subseteq X$ based on the joint distribution. For example, calculate marginal probability P(B,M) from the joint distribution.

$$P(B,M) = \sum_{E,A,J} P(B,E,A,J,M)$$

The general form of posterior query is P(Q|E=e), where Q denotes the query variables and E represents the evidence variables with observed values e correspondingly. For instance, an inference with evidence can be P(B|M=1).

**B. Inference as Relational Query**

The Bayesian networks can be naturally represented and stored in relational databases. Specifically, we transform each factor to a relational table. Other than variable attributes in a factor (relation), we introduce an extra attribute p to represent the probability value. For example, the corresponding relation of factor P(A|B,E) is tA (A,B,E,p),where p denotes the probability of P(A|B,E). According to properties of Bayesian networks, the joint distribution is specified by joining all the relations of factors, and can be represented by a relational database view. For example, P(B,E,A,J,M) in (1) corresponds to the view:

```
CREATE VIEW joint AS (SELECT B, E, A, J, M,
tB.p * tE.p * tA.p * tJ.p * tM.p AS p FROM tB, tE, tA,
tJ, tM WHERE tB.B=tA.B AND tE.E=tA.E AND
tM.A=tA.A AND tJ.A=tA.A )
```

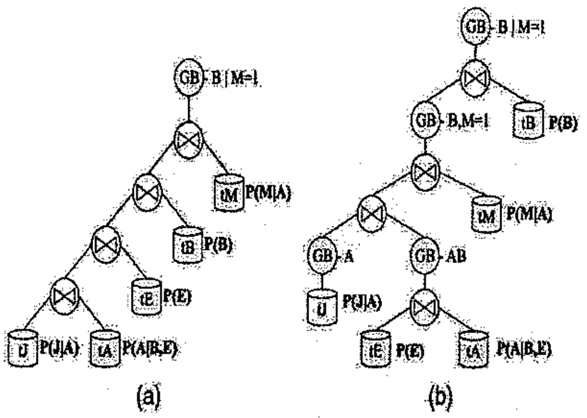


Fig.2. Query optimization in the joining tree

Consequently, the marginal probability of an inference query, P(Q), can be computed by aggregating the joint distribution to eliminate all the other variables not in Q, i.e.,

SELECT Q, SUM(P) FROM joint GROUP BY Q.

Note that the posterior probabilistic inference is a special case of a general inference query; it can be handled by adding an extra WHERE constraint to the above aggregate query. For example, the inference query P(B|M=1) can be conducted by the query

SELECT B, SUM(p)/q FROM joint WHERE M=1 GROUP BY B

where q is P(M=1) that can be computed by the previous query of marginal probability. Thus, this predicate computes the marginal probability distribution of variable B when M=1 is observed, i.e., P(B|M=1).

## III. Query Optimization with VE

A naive implementation of an inference query is to materialize a joint view by joining all the tables together, and then, perform GROUP BY (GB) operation to aggregate and eliminate the variables not in the query.[1][2] present transformations to push GROUP BY operation down into the joining tree. Since GROUP BY operation reduces the cardinality of a sub query result, an early conduction of GROUP BY can potentially save the cost of subsequent joins. Interestingly, there is a similar strategy in the literature of probabilistic inference in Bayesian networks, called the variable elimination algorithm. This is not surprising due to the similarity and correspondence between Bayesian networks and relational databases We first consider the elimination of one variable from the joint distribution [3]. Let $P(X_1,X_2, \ldots,X_m)$ be a joint distribution. Eliminating $X_1$ from P is to compute

$$P(X_2,\ldots,X_m) = \sum_{X_1} P(X_1,X_2,\ldots,X_m).$$

Since the joint distribution is represented by a set of factors in Bayesian networks, we only need to multiply all the factors containing $X_1$ and aggregate the results to eliminate $X_1$. In terms of relational databases, all the tables that include $X_1$ are product-joined, and the results are aggregated and grouped by the variables that have not been eliminated so far.

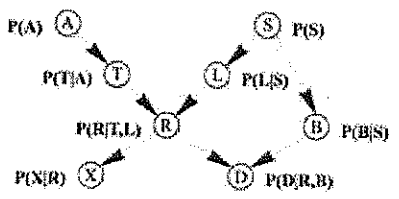


Fig.3.Bayesian Network

Given an ordering of variables for elimination, we can build the joining tree of the inference query. For instance, consider the ordering <E,J,A> for query P(B|M=1).As shown in Fig. 2, the first variable E is eliminated by joining all the relations with E, i.e., tE is product joined with tA and conducting GROUP BY with the remaining variables A,B. As a second step, variable J is eliminated. Finally, we eliminate A and GROUP BY B with a further WHERE M=1 condition to generate the results. Note that the evidence condition M=1 can be pushed down in the joining tree in order to reduce the intermediate results.

## IV. CTP IN DATABASES

Here the inference query $Q_k$ is studied by reusing the computation results in the previous query $Q_k$. Recall that the variable elimination answers one query at a time, and has no computation sharing among different queries. However, some of the sub query results in the joining tree could be reused among different queries.

In order to cache these intermediate query results, we employ the clique tree propagation to compute marginal probability, which enables the computation sharing among different inference queries. In addition to following the similar principle of VE, the clique tree propagation utilizes a smart caching strategy. Intuitively, we use the clique tree to cache the intermediate results (named messages) of eliminating some variables in the VE algorithm. When a new query arrives, it is possible to reuse the messages cached in the clique tree to avoid re computation. Consequently, the inference is conducted as message propagation in the clique tree.

- A clique tree is an undirected tree, where each node represents a set of variables, i.e., a clique.
- Message propagation is defined as Given a set of query variables Q, the message passed from the clique C to C' aggregates all the variables in C but not in C' and Q.
- In CTP Preliminary, a pivot is a clique node selected in the clique tree which usually contains some or all of the variables of query Q. During the inference query processing, we consider all the messages that are passed toward the pivot.

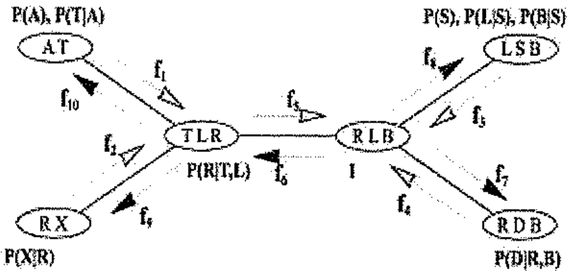


Fig.4. Clique tree propagation

Instead of discarding all the intermediate results in VE, all the clique nodes and messages $f_i$ in the clique tree are cached after processing the current query. If the variables of a following query is a subset of variables contained in a clique node, e.g., a query with variable A belongs to clique (AT) in Fig. 4, then we can directly compute the inference results from this clique. Otherwise, the messages have to be propagated in order to collect all the query variables, e.g., a query with variables S,D which cannot be covered by a single clique. When the following queries request the same cached message fi again, we can reuse the cached message result and stop propagation in the corresponding subtree.

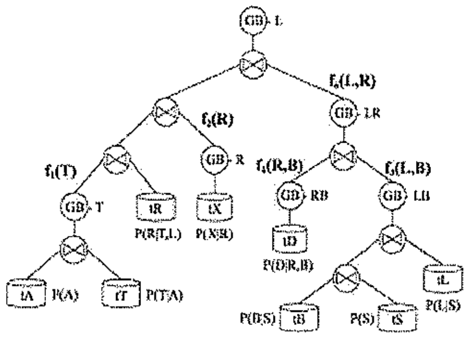


Fig.5.CTP applied to databases, Joining tree of $Q_k = \{L\}$

The Relational Framework for CTP, Transforming CTP to Relational Query are defined in the system. In Transforming CTP to Relational Query process having the four steps, such as Materialize the Clique, Materialize the Message, Sharing among Queries, Transformation Algorithm.

### A. Transforming CTP to Relational Query

The implementation issues of the message caching and propagation in relation databases are considered. Recall that the factors are stored as relational tables in databases. Similarly, in order to cache and reuse the intermediate results among different queries, we utilize the materialized views in databases for the clique tree propagation. Specifically, we store the cliques and messages $f_i$ as materialized views to enable the computation caching and sharing. Given a query Q, we implement the message propagation in the clique tree as the joining tree with materialized views.

### B. Materialize the Clique

The clique is the minimum unit in the clique tree, and also in the reuse of joining trees among queries. During the inference query processing, we always use the product function of all the functions attached in the clique, i.e. Àj gj. Thus, this product function can be pre-computed and cached in a materialized view for the computation sharing. For example, consider the clique (LSB) in Fig. 4. In the query, we always use the product function P(S)P(B|S)P(L|S) in that clique. Therefore, we can store the product function as a materialized view for the clique

```
CREATE MATERIALIZED VIEW vLSB AS (
SELECT L, S, B, tL.p * tS.p * tB.p AS p
FROM tL, tS, tB
WHERE tL.S=tS.S AND tB.S=tS.S )
```

By applying similar strategies, each clique can be materialized by a view. Then, the message propagation is conducted on these materialized (clique) views, rather than the original tables of factors.

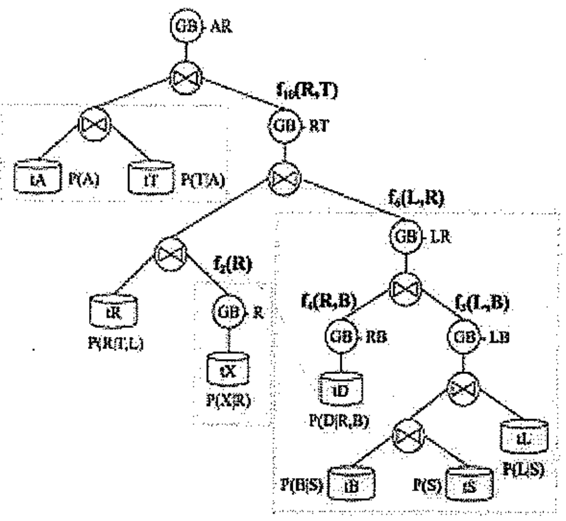


Fig.6. Joining tree of $Q_{k+1}$ = {A,R}

### C. Materialize the Message

The materialized views are used to represent the messages. Recall that each message $f_i$ is also a function of variables. The computation of the message can be implemented by relational queries, and we want to share the message query results among different inference queries. Thereby, we use the materialized views to store the query results of the messages, i.e., the materialized (message) views. Again, in the example in Fig. 4, we consider message $f_5$ passed from clique (TLR) to (RLB).Given a query Q, message $f_5$ collects all the functions attached in the clique view vTLR and the functions sent to the clique (TLR), that is, $f_1$ and $f_2$.

```
CREATE MATERIALIZED VIEW vf5 AS (
SELECT L, R, Q, vTLR.p * vf1.p * vf2.p AS p
FROM vTLR, vf1, vf2
WHERE vTLR.T=vf1.T AND vTLR.L=vf2.R
GROUP BY L, R, Q)
```

According to (2), all the variables in C but not in C' and Q should be eliminated and aggregated. In this example, only those variables in (RLB) and Q, will be reserved in message $f_5$. Note that evidence conditions like L=1 can also be represented in the materialized view, e.g., by WHERE L=1. In other words, the same as the VE algorithm, we can also push down the WHERE L=1 condition in the joining tree of CTP in query optimization settings.

### D. Sharing among Queries

Now, we consider the next query $Q_{k+1}$ by sharing with the previous query $Q_k$. The messages are passed from the leaf cliques toward the new pivot with $Q_{k+1}$.First, all the materialized views of cliques in $Q_k$ can be reused directly, for example the clique (AT) with P(A) join with

P(T|A) in both the queries of Figs. 5 and 6.Moreover, message $f_i$ that have the same variable settings in both query $Q_k$ and $Q_{k+1}$ can be reused as well.

Therefore, rather than rebuilding the entire joining tree for query $Q_{k+1}$, we construct the joining tree on the materialized views of the previous query $Q_k$. The joining tree with reused materialized views will be small in size. For example, in Fig. 7, we show the joining tree with caching and sharing views for the query in Fig. 6. All the requested clique node functions are reused from the materialized views, such as vAT and vTLR. Moreover, the same setting messages $f_2$,$f_3$,$f_4$, and $f_6$ can also be reused. Since messages $f_3$,$f_4$ are contained in message f6,we can directly use the materialized view $vf_6$ of message $f_6$.

### E. Transformation Algorithm

Consider message f passed from the clique C to C'. Let $f.V_p$ be the set of variables passed from C to C', and let $f.V_e$ be the set of variables eliminated in the previous steps before C. In terms of relational databases, $f.V_p$ is the set of variables in GROUP BY operation of f, while $f.V_e$ are all the other variables in the joining tree $T_f$ rooted in f. For example, in Fig. 5, $f_6.V_p$ is {L,R}, and $f_6.V_e$ is {B,D,S}. The joining tree for the CTP algorithm can be generated recursively.

**Algorithm 1.** Joining Tree for CTP original

```
procedure GJOININGTREE(C,C'. f)
let $C_1$,$C_2$, . . . ,$C_k$ be the neighbors of C except C'
let $f_1$,$f_2$, . . . ,$f_k$ be the corresponding messages
f.addchild(C)
for i $\leftarrow$ 1.k do          k =0 if C is a leaf node
if $f_i.V_e \cap Q$ "' $\phi$ then
GJOININGTREE($C_i$,C,$f_i$)
f.addchild($f_i$)
```

To summarize, we have presented the inference techniques by using the cached views in the current state clique tree after query $Q_k$. Therefore, in the original clique tree propagation, the caching strategy is most recently used (MRU).

## V. CTP OPTIMIZATION

TABLE 1

| Notations | Description |
|---|---|
| $f.V_p$ | Variables *passed* in $f$ in the original clique tree |
| $f.V_e$ | Variables *eliminated* in previous steps before $f$ |
| $f.V_r$ | Variables *requested* by $Q$ to be propagated in $f$ |
| $f.V_c$ | Variables *cached* in $f$ of the current state |

Here the clique tree propagation is optimized by considering the most frequently queried variables in a sequence of inference queries However, there might be some frequently queried variables in a sequence of inference queries. Heuristically, we would like to cache these frequent query variables in order to maximize the reuse of cached messages, i.e., the most frequently used (MFU) caching strategy. In this process, the following three steps preceded, Such as exploring Workload, Caching Frequent Variables, and Optimizing CTP Query. And the caching frequent variables consist of, Message Updating, Reserving Frequent Variables, and Caching in Pivot. Then the Optimizing CTP Query consists, Incremental Updating, Query Algorithm.

In Occurrence frequency, Let A be a query variable. Occurrence (A) describes the frequency or probability of variable A appearing in a query in a workload. In Co-occurrence association), Let A, B be two query variables. Co-occurrence (A, B) describes the frequency or probability of two variables appearing together in the same query. In the original clique tree propagation, the cached messages with different query variable settings to the current query have to be discarded, even though the messages contain the frequent query variables. In order to reserve the frequent query variables, we intend to avoid discarding the messages with unused frequent variables. To illustrate the frequent variable caching strategies in the messages, we first introduce the message updating operations of variables. In message updating we have two operations to add or remove variables in the messages: the message merge operation and purge operation.

Reserving Frequent Variables we consider the message whose cached variables are the superset of the variables requested by the current query. In terms of inference techniques, we need to identify and eliminate these irrelevant variables by using message purge operation. With features of GROUP BY operator, we do not need to take extra consideration to aggregate these variables. GROUP BY operation generates the messages with requested variables and aggregates all the other variables automatically. All the messages passed toward pivot are collected by a product operation. To reuse this computation step, we can further cache query variables in the pivot. Specifically, the probability functions of frequent queried variables are stored in the pivot according to the workload statistics. When another query is conducted on this pivot, if the requested query variables are already cached in the pivot, the results can be returned directly without gathering all the propagated messages again.

## Optimizing CTP Query

We discuss the inference query processing of message propagation, with the consideration of optimization by caching frequent query variables into both messages and pivots. We also study strategies of managing the cached variables incrementally according to variable associations from the workload statistics.

*Incremental Updating:* According to natural features of GROUP BY operator, the message purge operation is already included in message propagating steps. Now, we discuss the message merge in the message updating. As mentioned above, the message will be recomputed to include all the variables requested with query $f.V_r$. Then, the problem is whether or not we should replace the cached variables by new set $f.V_r$ in message f. Intuitively, those variables that are queried frequently in the workload should be cached in the message. To measure the priority of caching in the message, we define the association of two variable sets according to the variable association in the workload statistics.

*Query Algorithm* of generating joining trees for inference queries with our variable caching strategies in cliques and messages. Traditionally, we should update all the messages in the propagating path with different variable settings to the current query, i.e., $f.V_c$ ‘‘‘ $f.V_r$. In this study, we further prune the joining subtree where the requested messages are already available, that is the message purge operation. This property yields an early termination strategies in the construction of the joining tree. Consequently, we only need to generate the joining tree for the message merge operation.

**Algorithm 2.** Joining Tree for CTP caching

```
procedure GJOININGTREE(C,C',f)
let $C_1,C_2,...,C_k$ be the neighbors of C except C'
let $f_1,f_2,...,f_k$ be the corresponding messages
f.addchild(C)
for i $\leftarrow$ 1,k do        k = 0 if C is a leaf node
$f_i.V_r \leftarrow (f_i.V_e \cap Q) \cup f_i.V_p$
if $f_i.V_r \not\subset f_i.V_c$ then
GJOININGTREE($C_i$,C,$f_i$)
if $\phi(f_i.V_r, f_i.V_p) > \phi(f_i.V_c, f_i.V_p)$ then
$f_i.V_c \leftarrow f_i.V_r$
```

f.child = $f_i$

*Pivot selection* : To minimize the time cost, we study the heuristics for selecting the pivot. In the inference propagation, all the messages are passed toward the pivot. In order to reduce the propagation cost, the selected pivot is expected to contain more query variables. Note that there might not exist any clique that covers all the query variables. Traditionally, in this case, we select the clique node that has the most query variables as the pivot. However, since we cache the frequent query variables in the messages, the messages with some of the current query variables might already be available in the clique tree. Therefore, the number of query variables in a pivot clique node is no longer a good criterion for estimating the query plan cost. For instance, consider a query Q = {X, T,L} in the example in Fig. 4. Suppose that message $f_9$ caches variables T and L in the previous queries, while $f_2$ does not contain X. According to the traditional criterion, node (TLR) contains most query variables and can be selected as the pivot.

Thus, message $f_2$ needs to be recomputed to propagate the query variable X. However, since $f_9$ contains the query variables T and L, we can reuse $f_9$ directly without updating the message if the clique (RX) is selected as the pivot. In other words, the time cost of the query plan also depends on the number of query variables that are already cached in the corresponding propagating path. The challenge now is how to evaluate the reuse capability of selecting different pivots, in order to maximize the sharing among queries. Therefore, we study the following query plan estimation strategies to evaluate a query plan with a specific pivot.

## VI. Experimenal Evaluation

This section reports the experimental evaluation of the proposed approach. A probabilistic inference query as clique tree propagation is developed with the materialized view provided by the Oracle. We use a Bayesian network as database schema with totally 5 relational tables such as details, x-ray details, cancer, smoker, patient. Each of these table is a factor in the Bayesian network. The tables contain the necessary fields according to their description. The dataset that we have considered consists of 10,000 records to diagnose the probabilistic value of how many persons are affected by cancer. We need not want the details on how many are affected by cancer but we need to calculate the extent to which one record depends on the other and how far we can derive inferences from the details that are present. In the dataset we stimulate probability distribution for each relation following the normal distribution. We simulate the workload data of query inputs.

We read the set of input queries from different textpads and calculate the time period for variable elimination and clique tree propagation. In VE first we display the elimination result and query result. Elimination result consists of those variables that are not present in the query and the query result consists of the probabilistic values of the query that is being proposed to the system. Here we note down the execution time for the query result in VE. Then, on executing the CTP for the same set of queries and we see for the execution time of the results. First CTP with original is calculated then followed by that we calculate the CTP with caching. The CTP applied to original set of query will have lesser execution time compared to VE. This is because in CTP we generate a set of materialized views according to the query that is being asked. Materialized views are formed for each of the query that is passed from textpad. If suppose successive queries contain the same fields of the previous query along with other new fields, then the materialized view is updated from the previous one. Successively on considering the CTP with caching, the execution time results obtained will be lesser compared to the CTP with original.

The CTP with caching is calculated using the materialised views obtained from the previous results of CTP with original. Finally on analysing these we derive that CTP with caching processes the queries and displays the results with least execution time compared to other two approaches. For example, we consider the following Bayesian network :

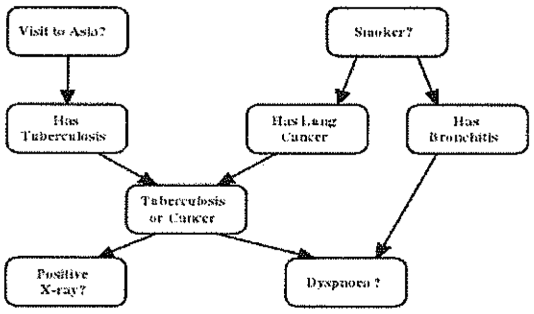


Fig.7.Bayesian Network for Cancer

Here we read a set of queries from a textpad and perform VE as the initial step. The output obtained is eliminated result which contains variables that are not present in the query and query result which is the result for the set of queries that have been passed as input to the system. Next, CTP is performed for the same set of queries. Since CTP employs clique tree propagation, the execution time of CTP will be lesser compared to the variable elimination. As the next step, CTP with caching is performed. This is executed on the original set of materialized views formed previously.

On analysing the execution time of VE and original CTP we derive at a conclusion that CTP original consumes execution time for the same set of queries passed in VE. Again, on analysing CTP original and CTP with inference, the second one consumes less time for execution which is comparably lesser than other two. This is because CTP caching is based on the materialized views. Since the results of previous queries are reused, this consumes a lesser time compared to the CTP without caching. On plotting a graph for VE, CTP and CTP with caching we can easily analyse the efficiency of our work as it reduces the execution time and memory for the probabilistic inference queries.

## VII. FUTURE WORK

### Belief Propagation for Frequent Probabilistic Inference Queries

Belief Network is defined. BN is a compact representation of a joint distribution over a set of random variables X. A BN is structured as a directed acyclic graph (DAG) whose vertices are the random variables and the directed edges represent dependency relationship among the random variables. The evidence in a Bayesian network consists of variables that have been instantiated. The junction tree algorithm propagates beliefs (or posteriors) over a derived graph called a junction tree. A junction tree is generated from a BN by means of moralization and triangulation.

In many applications belief propagation over the junction tree is used, this is a two-phase procedure: evidence collection and evidence distribution. For the evidence collection phase, messages are collected from the leaf vertices all the way up to a designated root vertex. For the evidence distribution phase, messages are distributed from the root vertex to the leaf vertices. Message passing can be viewed as the atomic operation for belief propagation, both for evidence collection and distribution. This produces the more reliable answering Frequent Probabilistic Inference Queries in Databases.

## VIII. Conclusion

We study the frequent probability inference queries in relational databases. Rather than reconstructing the joining tree for each query, we focus on the approaches that enable the caching and computation sharing among the frequent queries in relational databases. First, we transform the inference query of clique tree propagation (CTP) to the relational query of joining tree. Moreover, to further maximize the sharing among a sequence of queries, a variable caching optimization scheme is also proposed to cache those frequent query variables in both the cliques and messages. Our CTP caching optimization approach not only shares the messages when the query matches the cached frequent variables, but also reduces the times of discarding the messages with frequent variables during the message updating. The experimental results demonstrate the effectiveness of our caching and sharing strategies among the frequent queries.

## References

[1] S. Chaudhuri and K. Shim, "Including Group-By in Query Optimization", *Proc. Int'l Conf. Very Large Databases (VLDB)*,pp. 354-366, 2000.

[2] S. Chaudhuri and K. Shim, "Optimizing Queries with Aggregate Views", *Proc. Int'l Conf. Extending Database Technology (EDBT)*,pp. 167-182, 2000.

[3] H.C. Bravo and R. Ramakrishnan, "Optimizing MPF Queries Decision Support and Probabilistic Inference", *Proc. ACM SIGMOD*, pp. 701-712, 2007.

[4] S.Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, "Optimizing Queries with Materialized Views", *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 190-200, 2005.

[5] J. Goldstein and P.-_A. Larson, "Scalable Solution in Optimizing Queries Using Materialized Views", *Proc. ACM SIGMOD*, pp. 331-342, 2001.

[6] F.V. Jensen, *Introduction to Bayesian Networks*. Springer-Verlag, 2000.

[7] P.P.Karde and Dr.V.M.Thakare, "Selection of Materialized View Using Query Optimization in Database Management : An Efficient Methodology", *International Journal of Database Management Systems*, Vol.2,No.4,November 2010.

[8] S.K.M. Wong, C.J. Butz, and Y. Xiang, "A Method for Implementing a Probabilistic Model as a Relational Database", *Proc. Conf. Uncertainty in Artificial Intelligence (UAI)*, pp. 556-564, 2002.

[9] S.K.M. Wong, D. Wu, and C.J. Butz, "Probabilistic Reasoning in Bayesian Networks: A Relational Database Approach", *Proc. Conf.Artificial Intelligence (AI)*, pp. 583-590, 2003.

[10] N.L. Zhang and L. Yan, "Independence of Causal Influence and Clique Tree Propagation," *Int'l J. Approximate Reasoning*, vol. 19,nos. 3/4, pp. 335-349, 1998.

[11] C.J. Butz, H. Yao, and S. Hua, "A Join Tree Probability Propagation Architecture for Semantic Modeling", *J. Intelligent Information Systems*, vol. 33, pp. 145-178, 2008.

[12] F.V. Jensen and F. Jensen, "Optimal Junction Trees", *Proc. Conf. Uncertainty in Artificial Intelligence (UAI)*, pp. 360-366,2008.

[13] R.D. Shachter, B. D'Ambrosio, and B.D. Favero, "Symbolic Probabilistic Inference in Belief Networks," *Proc. Nat'l Conf.Artificial Intelligence (AAAI)*, pp. 126-131,2000.

[14] W.X. Wen, "From Relational Databases to Belief Networks," *Proc.Conf. Uncertainty in Artificial Intelligence (UAI)*, pp. 406-413, 2003.

[15] J. Goldstein and P.-_A. Larson, "Optimizing Queries Using Materialized Views: A Practical, Scalable Solution," *Proc. ACM SIGMOD*, pp. 331-342, 2001.

***Author's Profile :***

V.P.Sumathi is currently working as an Assistant Professor in Computer Science Department at Kumaraguru College of Technology, Coimbatore. Sheis pursuing her Ph.D. Her area of interest is Query Optimization in Data Mining.