Answering Frequent Probabilistic Inference Queries in Databases

Shaoxu Song, *Student Member*, *IEEE*, Lei Chen, *Member*, *IEEE*, and Jeffrey Xu Yu, *Senior Member*, *IEEE*

Abstract—Existing solutions for probabilistic inference queries mainly focus on answering a single inference query, but seldom address the issues of efficiently returning results for a sequence of frequent queries, which is more popular and practical in many real applications. In this paper, we mainly study the computation caching and sharing among a sequence of inference queries in databases. The clique tree propagation (CTP) algorithm is first introduced in databases for probabilistic inference queries. We use the materialized views to cache the intermediate results of the previous inference queries, which might be shared with the following queries, and consequently reduce the time cost. Moreover, we take the query workload into account to identify the frequently queried variables. To optimize probabilistic inference queries with CTP, we cache these frequent query variables into the materialized views to maximize the reuse. Due to the existence of different query plans, we present heuristics to estimate costs and select the optimal query plan. Finally, we present the experimental evaluation in relational databases to illustrate the validity and superiority of our approaches in answering frequent probabilistic inference queries.

Index Terms—Probabilistic inference, variable elimination, clique tree propagation.

1 Introduction

Recently, 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 [1], object identification [2], location-based services (LBS) [3], and moving object tracking [4]. In these applications, databases are often employed to describe the probability distribution of all variables $\mathbf{X}$ 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 [5]. Formally, an inference query is to compute the marginal probability distribution $P(Q)$ of a subset variables $Q \subseteq \mathbf{X}$ 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 probability 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 *jointable* tables together first, which is not efficient. To improve the evaluation efficiency, a variable elimination algorithm in *Bayesian networks* is implemented in relational databases [5]. In fact, the *Bayesian network* itself is widely used as a compact representation of the probability distribution of all the variables [6], [7] and it can be naturally represented and stored in relational databases [8]. 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. Given another example of decision support in enterprise's supply chain database [5], a manager of manufactures may frequently pose different queries to databases to draw up a monthly production plan, e.g., to query the probability of parts p_a and p_b being ordered together in the last month, or the probability of parts p_b and p_c together in August, or even the probability of p_a, p_b, and p_c together in every month of the last year.

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. Wong et al. [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. Bravo and Ramakrishnan [5] present a broad

- *S. Song and L. Chen are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: {sshaoxu, leichen}@cse.ust.hk.*
- *J.X. Yu is with the Department of System Engineering and Engineering Management, The Chinese University of Hong Kong, William M.W. Mong Engineering Building, Shatin, NT, Hong Kong. E-mail: yu@se.cuhk.edu.hk.*

Manuscript received 15 Dec. 2008; revised 24 July 2009; accepted 7 Dec. 2009; published online 24 Aug. 2010.
Recommended for acceptance by S. Chakravarthy.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2008-12-0661.
Digital Object Identifier no. 10.1109/TKDE.2010.146.

class of aggregate queries, called Marginalize a Product Function (MPF), and implement the *variable elimination* (VE) [10] 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 (see Section 2.3). 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 proposed work 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 a subset variables $Q \subseteq \mathbf{X}$ from the probability distribution of all variables $\mathbf{X}$, that is, to eliminate variables $\mathbf{X} - Q$ from $\mathbf{X}$. 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* $\mathbf{X} - Q$ *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 $\mathbf{X} - Q$ of different queries, we study the approach of treating inference as message propagation [11], [12]. The *clique tree propagation* (CTP) [12], also known as *junction tree propagation* [13], [14], 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 $\mathbf{X} - Q$ 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. We also discuss the correctness of a probabilistic inference on the probability distribution with cached query variables. By analyzing the updating operations of cached variables, we further reduce the times of discarding the cached variables with high frequency. To further reduce the query time cost in databases, we derive the heuristics of evaluating the cost in advance, so we can choose the query plan with the minimum estimated cost.

Our contributions in this paper are listed as follows:

- We transform the message propagation in probabilistic inferences to joining tree queries in databases by using materialized views in relational databases. To the best of our knowledge, this is the first paper to evaluate 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.[1]
- We explore the workload statistics to find the frequent variable(s) among different queries, and apply this workload information in the query optimization. Our CTP caching approach optimizes the message propagation by caching the *frequent* query variables in the materialized views, and thus maximize the computation reuse of the frequent query variables in a sequence of inference queries.
- We study the estimation strategies of query plans, and propose the minimum propagating join cost estimation in relational databases.

The rest of the paper is organized as follows: In Section 2, we introduce probabilistic inference queries in relational databases which further motivates our work. Section 3 discusses the issues of implementing clique tree propagation in relational databases. Then, we present our frequent variable caching optimization of CTP in relational databases in Section 4, and discuss the query plan estimation strategies in Section 5. Section 6 reports the experimental results. We discuss the related work in Section 7 and conclude in Section 8.

2 Probabilistic Inference Query

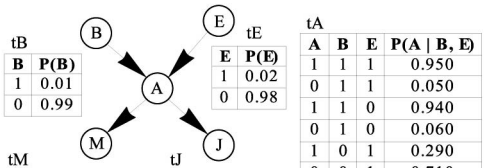
This section discusses a framework of probabilistic inference queries in relational databases. We start with relational tables storing the *Bayesian networks* [6], [7], which are widely used as a compact representation of the probability distribution of all the variables.

2.1 Probabilistic Inference

Consider a set of discrete random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. A *Bayesian network* [6], [7] 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 [8] 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

1. Here, Q_k is a query in the query workload, and Q_{k+1} is the next query after Q_k in the workload.



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 databases.

example, the tableau tA corresponding to 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, i.e.,

$$P(B,E,A,J,M) = P(B)P(E)P(A|B,E)P(J|A)P(M|A). \quad (1)$$

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 2^5 tuples, while the Bayesian network representation needs only 20 tuples in all the factors. For the detailed knowledge of Bayesian networks, please refer to [6], [7].

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 $Q \subseteq \mathbf{X}$ 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 $P(B|M=1)$.

2.2 Inference as Relational Query

The Bayesian networks can be naturally represented and stored in relational databases [8]. 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 )
```

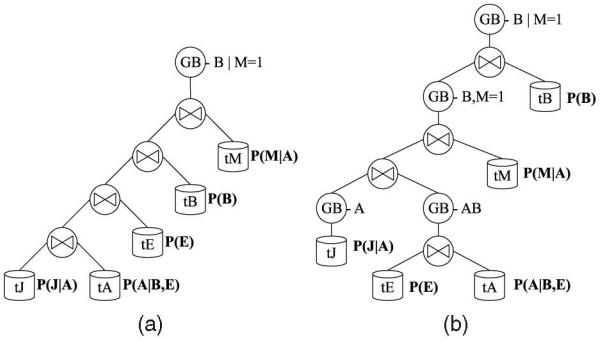


Fig. 2. Query optimization in the joining tree.

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., $P(B|M=1)$.

2.3 Query Optimization with VE

A naïve 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. For example, in Fig. 2a, we show the joining tree for computing the materialized view of the joint distribution. Then, GB operation aggregates the results to generate marginal probability of the query.

In the database literature, techniques have been proposed to optimize the aggregation queries on joining trees. Chaudhuri and Shim [15], [16] present transformations to push GROUP BY operation down into the joining tree. Since GROUP BY operation reduces the cardinality of a subquery 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* [10] algorithm, which also studies the aggregation of variables one by one on the factors rather than the joint distribution. This is not surprising due to the similarity and correspondence between Bayesian networks and relational databases [17]. Bravo and Ramakrishnan [5] implement the VE algorithm in relational databases, and study a cost-based ordering heuristic for variable elimination. We introduce the VE algorithm briefly as follows:

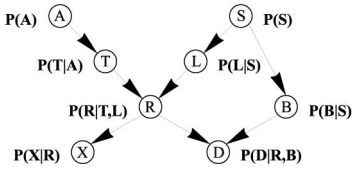


Fig. 3. Bayesian network.

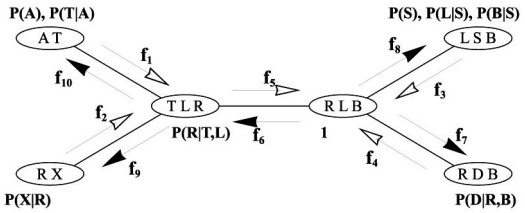


Fig. 4. Clique tree propagation.

2.3.1 Variable Elimination

We first consider the elimination of one variable from the joint distribution. 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. For example, to eliminate variable E, we first join all the tables containing E together, i.e., $tA \bowtie tE$ (note that `tA` corresponding to the conditional probabilities $P(A|B, E)$ of A given variables B, E) and then group and aggregate the results by the remaining variables A, B.

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. 2b, the first variable E is eliminated by joining all the relations with E, i.e., $tE \bowtie tA$, 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.

So far, we have introduced the techniques for a single inference query. In the rest of this paper, we consider a sequence of inference queries $Q_1, Q_2, \ldots, Q_{k-1}, Q_k, Q_{k+1}, \ldots$, and study the techniques for computation sharing among these queries and the corresponding optimization issues.

3 CTP in Database

In this section, we study the inference query Q_{k+1} 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 subquery results in the joining tree could be reused among different queries. For example, the computation of elimination variable E in Fig. 2b is exactly the same for the inference query $P(B, M)$ and $P(B, A)$; thus, the elimination results can be shared and reused between these two queries.

In order to cache these intermediate query results, we employ the *clique tree propagation* [12] 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 recomputation. Consequently, the inference is conducted as *message propagation* in the clique tree. The detailed definitions of clique tree and message propagation for sharing the intermediate results are given as follows.

3.1 CTP Preliminary

We consider a more complicated Bayesian network in Fig. 3 to illustrate the CTP. The clique tree of this Bayesian network is given in Fig. 4 with the corresponding messages.

Definition 1 (Clique tree). *A clique tree is an undirected tree, where each node represents a set of variables, i.e., a clique.*

Cliques in a clique tree should be *variable connected*,[2] which indicates that the subgraph of cliques containing a given variable should be connected. Specifically, if a variable appears in two cliques, then it must appear in all the cliques on the path between those two cliques. For example, consider variable L in the clique tree in Fig. 4. Let (AT), (TLR),... denote all the clique nodes in the tree. All the cliques with variable L (TLR), (RLB), and (LSB) should be a connected subgraph of the clique tree. Note that the variable-connected property is ensured in the clique tree construction algorithm, which has been studied in the CTP literature [12]. During the clique tree construction, each factor in the Bayesian network is assigned to a clique node which contains corresponding variables. Therefore, both the clique nodes and the messages in the clique tree consist of *functions* of variables. The messages with functions propagated between cliques are defined as follow:

Definition 2 (Message propagation). *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:*

$$f\big(C \cap (C' \cup Q)\big) = \sum_{C \setminus (C' \cup Q)} \prod_i f_i \prod_j g_j, \tag{2}$$

where f_i denotes all functions sent by the messages toward C but not from C', and g_i represents all the functions attached to the clique C.

We discuss the features of messages in the clique tree during the inference query processing. According to Definition 2, there are some messages that do not contain any

2. Also known as running intersection property.

query variable. For example, given a query $Q_k = \{L\}$ in Fig. 4, message f_2 passed from clique (RX) to (TLR) is $f_2(R)$ without the query variable L. On the other hand, for the same query $Q_k = \{L\}$, message f_9 passed from clique (TLR) to (RX) should be $f_9(R, L)$ which contains the query variable L. Therefore, we mainly have two kinds of messages in the clique tree after Q_k, with or without query variables. If the new query Q_{k+1} has the same variable setting as Q_k in a specific message f_i, then this message can be reused directly in Q_{k+1}. Otherwise, we have to discard the old one and rebuild the message for the new query Q_{k+1}. For example, message f_9 for $Q_{k+1} = \{T\}$ should be $f_9(R, T)$ and cannot reuse the cached message $f_9(R, L)$ of the previous query Q_k.

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. For example, let the query variable be $Q = \{L\}$. We select a clique C_Q that contains Q, for instance node (TLR), and use it as a *pivot* in the inference $P(L)$. Then, the messages propagated from leaves to the pivot (TLR) are computed by

$$f_1(T) = \sum_A P(A)P(T|A),$$
$$f_2(R) = \sum_X P(X|R),$$
$$f_3(L, B) = \sum_S P(S)P(L|S)P(B|S),$$
$$f_4(R, B) = \sum_D P(D|R, B),$$
$$f_6(L, R) = \sum_B f_3(L, B)f_4(R, B).$$

Note that the function attached in the clique (RLB) is an identity function, that is **1**. Thus, message f_6 collects the functions of f_3, f_4, and **1**.

Finally, we compute the inference query results, by collecting the functions in the pivot clique and the messages sent to the pivot directly. Let $f_i(Q, X)$ be all the functions sent to C_Q by the messages, and $g_j(Q, X)$ be all the functions attached in pivot C_Q. Then, the marginal probability of Q is

$$P(Q) = \sum_X \prod_i f_i(Q, X) \prod_j g_j(Q, X). \quad (3)$$

For example, the function collection of pivot (TLR) for query $Q = \{L\}$ consists of

$$P(L) = \sum_{T,R} f_1(T)f_2(R)f_6(L, R)P(R|T, L). \quad (4)$$

For other forms of inference queries, such as the posterior query $P(Q|E = e)$, we only need to add further constraints of the evidence $E = e$ which is similar to VE.

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 f_i again, we can reuse the cached message result and stop propagation in the corresponding subtree.

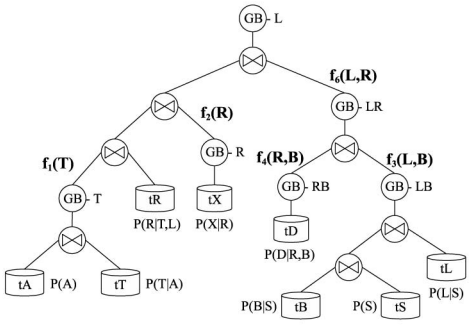


Fig. 5. Joining tree of $Q_k = \{L\}$.

3.2 Relational Framework for CTP

We now present the framework for clique tree propagation in relational databases, which is also discussed in [8]. Essentially, the computation of the messages in (2) is transformed to the relational queries in databases. Thereby, the whole process of computing all the messages propagated toward the pivot clique corresponds to a joining tree with relational tables. For example, consider the joining tree for inference query $Q_k = \{L\}$, e.g., $P(L)$. Again, let node (TLR) be the pivot in the clique tree. As shown in Fig. 5, where $\bowtie$ denotes join operation and GB is group by operation in databases, message f_i passed toward the pivot can be computed step by step in the joining tree. Specifically, we first compute messages $f_1(T), f_2(R)$ that eliminate variable A, X, respectively, and then messages $f_3(L, B), f_4(R, B)$ that eliminate S, D, respectively. According to the clique tree structure, message $f_6(L, R)$ is computed by collecting the results of $f_3(L, B), f_4(R, B)$, which eliminates variable B. The formulas for computing all these messages f_i have been given in Section 3.1, and their corresponding SQL implementations in databases are also introduced in Section 3.3. Finally, marginal probability $P(L)$ is calculated by collecting the functions in message $f_1(t), f_2(R), f_6(L, R)$ passed to the pivot and the function $P(R|T, L)$ in the pivot clique (TLR), i.e., (4). Note that this is exactly the joining tree of VE algorithm with the same ordering $<A, X, S, D, B, T, R>$.

Then, we consider the next query $Q_{k+1} = \{A, R\}$ such as $P(A, R)$. In the case of the VE algorithm, we have to reconstruct another joining tree (in Fig. 6) without any computation reuse for the new query $P(A, R)$. With the clique tree propagation, rather than entirely rebuilding the joining, we only need to modify parts of the messages with changes in query variables, and reuse all the remaining unchanged messages.

For query $P(A, R)$, we can use node (AT) in Fig. 4 as the pivot, which contains one of the query variables A. According to the principle of message propagation, the messages should be passed toward the pivot. Therefore, in the clique in Fig. 4, the message should be passed from the (TLR) to (AT), i.e., f_{10} rather than f_1. The remaining messages f_2, f_3, f_4, and f_6 are passed toward the pivot

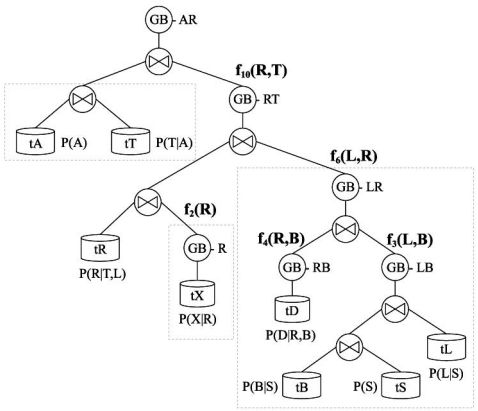


Fig. 6. Joining tree of $Q_{k+1} = \{A, R\}$.

without any change, and can be shared and reused in the current query. As shown in Fig. 6, we can reuse part of results in the previous query (in rectangles with dotted line), and only need to compute the new message f_{10} passed from (TLR) to (AT):

$$f_{10}(R,T) = \sum_{L} f_2(R) f_6(L,R) P(R|T,L).$$

Note that the original message f_{10} cannot be used directly which eliminates R, since R is the query variable. Instead, we update message f_{10} which eliminates L and reserves R for the new query.

Therefore, by using the CTP, rather than reconstructing the joining tree entirely for each query, we enable the computation caching and sharing of joining trees among different queries. Similar to VE, the posterior inference with evidence $E = e$, for example, $P(A|R = 1)$, can be naturally implemented by simply putting an additional `WHERE E = e` condition in the SQL statement, which is also evaluated in our experiments.

3.3 Transforming CTP to Relational Query

We consider the implementation issues of the message caching and propagation in relation databases. 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.

3.3.1 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., $\prod_j g_j$ in (2). Thus, this product function can be precomputed 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, i.e., vLSB $\leftarrow P(S) \bowtie P(B|S) \bowtie P(L|S)$.

```
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.

3.3.2 Materialize the Message

We can also use the materialized views to represent the messages. Recall that each message f_i is also a function of variables given by (2). 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, i.e., $C \cap (C' \cup Q) = \{L, R, Q\}$, 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.

3.3.3 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 $P(A) \bowtie P(T|A)$ 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. That is, $f_i(C \cap (C' \cup Q_k)) = f_i(C \cap (C' \cup Q_{k+1}))$, for example message f_2, f_3, f_4, and f_6 in Figs. 5 and 6. Otherwise, we need to recompute message f_i for the new query Q_{k+1}.

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 f_6, we can directly use the materialized

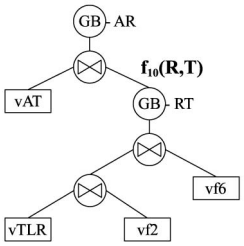


Fig. 7. Joining tree of $Q_{k+1} = \{A, R\}$ with views.

view vf6 of message f_6. In other words, we only need to compute the new messages passed to the current pivot clique with the variables of Q_{k+1}, that is f_{10} instead of f_1 in Fig. 4.

3.3.4 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', that is $C \cap (C' \cup Q)$ as shown in Definition 2, 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 $\mathcal{T}_f$ rooted in f. For example, in Fig. 5, $f_6.V_p$ is $\{L, R\}$, and $f_6.V_e$ is $\{B, D, S\}$.

We present Algorithm 1 for transforming the clique tree propagation to the relational query plan. Rather than recomputing all the messages and reconstructing the joining tree, we only need to update the messages in some paths to propagate query variables. That is, message f_i with $f_i.V_e \cap Q \neq \emptyset$. Specifically, if query Q requests the variables in the eliminated variable set of message f_i, i.e., $f_i.V_e \cap Q \neq \emptyset$, then the message needs to be recomputed to include the requested query variables. Otherwise, f_i can be reused and attached to the joining tree directly, e.g., message f_6 in Fig. 7, and it is not necessary to recompute the subtree under f_i. The joining tree for the CTP algorithm can be generated recursively.

Algorithm 1. Joining Tree for CTP original

```
1: procedure GJOININGTREE(C, C', f)
2:     let C_1, C_2, ..., C_k be the neighbors of C except C'
3:     let f_1, f_2, ..., f_k be the corresponding messages
4:     f.addchild(C)
5:     for i ← 1, k do                    ▷ k = 0 if C is a leaf node
6:         if f_i.V_e ∩ Q ≠ ∅ then
7:             GJOININGTREE(C_i, C, f_i)
8:         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. Note that message f in the current state is cached in a previous query $Q_x, x = k, k-1, k-2, \ldots$ where message f is most recently updated. Therefore, in the original clique tree propagation, the caching strategy is *most recently used* (MRU).

4 CTP Optimization

In this section, we optimize the clique tree propagation by considering the most frequently queried variables in a sequence of inference queries $Q_k, , Q_{k-1}, Q_{k-2}, \ldots$. From the discussion in the previous section, we know that the original CTP caches the most recently updated messages in the materialized views. 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.

Specifically, in Section 4.1, we first explore the workload to find out those variables appearing frequently in queries. The associations among the query variables from the workload statistics [18] can be used to find the most frequent variables. Then, in Section 4.2, we study the strategies for caching the frequent variables in both the messages and the pivots in the clique tree. We further analyze the message updating strategies to reduce the times of recomputing the messages. Finally, we present the query processing of CTP with caching frequent variables in the clique tree in Section 4.3. Table 1 lists the frequently used notations in this section.

TABLE 1
Notations

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

4.1 Exploring Workload

We first study the variables appearing frequently in the workload, i.e., the most frequently used variables. Specifically, we explore and learn the associations of variables from the workload. Let $\mathbf{Q} = \{Q_1, Q_2, \ldots\}$ be a sequence of queries, where each query Q_k consists of several query variables. We can explore the query workload as follows:

Definition 3 (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.*

Those variables with high *occurrence frequency* in the workload are important in the computation sharing. Moreover, we can also cache some variables that appear together frequently, which might be queried again and thus reused according to the workload statistics.

Definition 4 (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.*

These two statistics can be computed by applying the algorithm of pairwise association rules [19]. Specifically, in the real implementation, we scan the workload data to compute the occurrence and co-occurrence of variables, respectively. In fact, the *occurrence frequency* and *co-occurrence association* correspond to the frequent 1-item sets and the frequent 2-item sets mining, respectively.

Definition 5 (Variable association). *Let A, B be two query variables. The* variable association *of A, B is*

$$\phi(A,B) = \frac{f(A,B)}{N}, \tag{5}$$

where N denotes the total number of queries in the workload, and f(A, B) means the frequency of A, B appearing together in the same queries, i.e., co-occurrence*(A, B).*

Here, to be consistent, we also define the association of a variable with itself to be $\phi(A,A) = \frac{f(A,A)}{N} = \frac{f(A)}{N}$, i.e., $occurrence(A)$.

In the inference query processing, the query variables are propagated in the messages toward the pivot. It is desired that the variables with high associations should have short distance of propagating them toward the pivot. The distance equal to zero means that two variables appear in the same clique in the clique tree. Thus, the variable associations can be utilized in the optimization of both caching query variables and pivot selection.

4.2 Caching Frequent Variables

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.

4.2.1 Message Updating

Given a query Q, some of the messages need to be updated to propagate the query variables toward the pivot. Rather than the original variables passed from one clique to another in the initialization, some new variables requested by the query are also propagated in the message. Therefore, we study the following property of variables that are actually propagated in a query.

Theorem 1. *Let f be the message passed from C to C', and let $f.V_r$ be the actual variables propagated from C to C' that are requested by a query Q. Then, we have*

$$f.V_p \subseteq f.V_r \subseteq (f.V_p \cup f.V_e). \tag{6}$$

Proof. Here, $f.V_e$ denotes all the variables *eliminated* in the previous steps before C. Therefore, if the variables in $f.V_e$ are requested by query Q, then we should not eliminate these variables in current and previous message propagation steps. The actual set of variables propagated in message f should be $f.V_r = f.V_p \cup (f.V_e \cap Q)$. Thus, the minimum set of $f.V_r$ is $f.V_p$ (i.e., no variables are requested by query Q), while the maximum set is $f.V_p \cup f.V_e$ (i.e., all the variables eliminated in the previous steps are requested by the query). In other words (in relational terms), it is allowed for some query variables in the joining tree $\mathcal{T}_f$ to appear in the message of f. The maximum set of variables in f should be all the variables appearing in $\mathcal{T}_f$. And, those variables not in $\mathcal{T}_f$ will never appear in message f. □

Now, we consider a sequence of queries. The variables actually propagated in a message change among different queries. The caching messages with requested query variables in a certain state of the clique tree may contain the variables in many other queries, probably with high frequency in the workload. Thus, we can *cache* these frequently requested query variables in the messages, and reuse these caching messages in the following queries. For example, in Fig. 4, suppose that variable A appears frequently with L, R in the queries in workload. In other words, message f_1 with query variable A is requested frequently. Therefore, rather than caching message $f_1(T)$, we can further cache the frequently query variable A into the message, i.e., $f_1(T, A)$.

Thus, we have two operations to add or remove variables in the messages: the *message merge* operation and the *message purge* operation. 1) If the cached variables $f.V_c$ do not contain all the variables requested by the query, i.e., $f.V_c \not\supseteq f.V_r$, then the *message merge* operation recomputes the message views to include all the new requested query variables. 2) If the cached variables $f.V_c$ contain all the variable requested by the query, i.e., $f.V_c \supseteq f.V_r$, then the *message purge* operation aggregates the message views to eliminate all the cached variables that are not requested by the query.

4.2.2 Reserving Frequent Variables

Since we cache the frequent query variables in messages, some of the frequent variables may not be requested by the current query. According to the original CTP, we have to discard the message and recompute a message by collecting all the propagated functions. We now discuss the correctness of propagation directly on these messages with these extra unused frequent query variables, to avoid discarding cached messages.

We consider the message whose cached variables are the superset of the variables requested by the current query, i.e., $f.V_c \supset f.V_r$. In terms of inference techniques, we need to identify and eliminate these irrelevant variables by using *message purge* operation. Fortunately, 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 $f.V_r$, and aggregates all the other variables automatically. Therefore, although variables in different queries coexist in the messages of a certain clique tree status, these variable variances will not affect the query results. To summarize, when the message purge operation is needed in the updating, it is not necessary to discard and recompute the message. Instead, we can reserve and directly use the messages with extra unused frequent variables.

4.2.3 Caching in Pivot

As shown in (3), all the messages passed toward the 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.

Let $\mathcal{P}.V_c$ be the set of variables cached in pivot $\mathcal{P}$, and Q be the set of query variables in a query. If $Q \subseteq \mathcal{P}.V_c$, then the results can be computed directly by eliminating (aggregating) variables $\mathcal{P}.V_c - Q$ in $\mathcal{P}$. As discussed above, this

aggregation can be performed efficiently by facilitating GROUP BY operator in relational databases.

4.3 Optimizing CTP Query

We now 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.

4.3.1 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. Consider two sets of variables V_1 and V_2. The association of V_1 and V_2 is given by

$$\Phi(V_1, V_2) = \sum_{A \in V_1, B \in V_2} \phi(A, B), \tag{7}$$

where $\phi(A, B)$ is the association of variables A and B studied in the query workload statistics. Then, the association score of the current cached variables to the passed variables is $\Phi(f.V_c, f.V_p)$, which is preferred to be high.

The intuition is that the cached variables are expected to be reused as much as possible in the query workload. Those cached query variables $f.V_c$ should have a high probability to appear together with the originally passed variables $f.V_p$ in the query workload, i.e., high association scores. Let $f.V_r$ be the requested query variables of the current query Q. If $\Phi(f.V_r, f.V_p) > \Phi(f.V_c, f.V_p)$, then we will replace the current cached variables $f.V_c$ by the new requested variables $f.V_r$, i.e., $f.V_c = f.V_r$.

A natural extension is the incremental caching strategies in the pivot. Let $\mathcal{P}.V_c$ be the variables cached in pivot $\mathcal{P}$ and let Q be a query. If $\Phi(Q, Q) > \Phi(\mathcal{P}.V_c, \mathcal{P}.V_c)$, then the cached variables will be replaced by the query, i.e., $\mathcal{P}.V_c = Q$.

4.3.2 Query Algorithm

We present Algorithm 2 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., $f.V_c \neq f.V_r$. In this study, we further prune the joining subtree where the requested messages are already available, that is the message purge operation where $f.V_c \supset f.V_r$. 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, i.e., $f_i.V_r \not\subseteq f_i.V_c$.

Algorithm 2. Joining Tree for CTP caching

```
1: procedure GJOININGTREE(C, C', f)
2:     let C_1, C_2, ..., C_k be the neighbors of C except C'
3:     let f_1, f_2, ..., f_k be the corresponding messages
4:     f.addchild(C)
5:     for i ← 1, k do                    ▷ k = 0 if C is a leaf node
6:         f_i.V_r ← (f_i.V_e ∩ Q) ∪ f_i.V_p
7:         if f_i.V_r ⊈ f_i.V_c then
8:             GJOININGTREE(C_i, C, f_i)
9:             if Φ(f_i.V_r, f_i.V_p) > Φ(f_i.V_c, f_i.V_p) then
10:                f_i.V_c ← f_i.V_r
11:        f.child = f_i
```

The generated result is the minimum joining tree of messages that are needed to be updated for the current query Q. For example, we perform query $Q_2 = \{A, R\}$ with pivot (AT). Assume that variable R is already cached in message f_{10}. Then, there is no need to update any message for the query, whereas the traditional approach without query variable caching has to recompute at least one message f_{10} to propagate the query variable R. Therefore, our frequent variable caching strategy can maximize the computation reuse among frequent queries and thus improve the query performance.

5 Cost Estimation

So far, we have presented the clique tree propagation in relational databases when a certain pivot is given. Now, we discuss the pivot selection for inference queries. Rather than the traditional pivot selection approaches, we introduce the strategies to estimate the query plan cost in relational database computation. To minimize the time cost, we study the heuristics for selecting the pivot.

5.1 Pivot Selection

First, let us briefly review the problem of pivot selection. 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.

5.2 Query Plan Estimation

We derive the heuristics of pivot selection step by step. First, according to Algorithm 2, the CTP with caching frequent variables has an early termination scheme if the request query variables are already cached in the messages and pivots. Consequently, the corresponding propagation steps are reduced. Thus, the straightforward estimation of a query plan is the total length of steps in the propagating paths. In other words, one possible measure for the query plan cost is *Minimum Propagating Path*.

5.2.1 Minimum Propagating Path

The *minimum propagating path* estimation returns a pivot with the shortest propagating path for a certain query. To compute *Minimum Propagating Path* of a query plan, let's first define propagation distance, which is closely related to *Minimum Propagating Path*, for the clique tree propagation.

Definition 6 (Propagation distance). *Given a pivot $\mathcal{P}$, $distance(A, \mathcal{P})$ denotes the minimum path length of propagating variable A to pivot $\mathcal{P}$.*

The propagation distance is the minimum length of the path between pivot $\mathcal{P}$ and the clique that contains variable A, having possible $distance(A, \mathcal{P}) = \{0, 1, 2, \ldots\}$. Here, $distance = 0$ indicates that pivot $\mathcal{P}$ contains variable A exactly. The following theorem ensures that $distance(A, \mathcal{P})$ must exist between any variable A and pivot $\mathcal{P}$ with the shortest path.

Theorem 2. *Let C_A be any clique that contains the variable of A. Then, there must exist one and only one clique C'_A, which contains A as well, having the path length of $(C'_A, \mathcal{P})$ equal to $distance(A, \mathcal{P})$.*

Proof. First, according to the clique tree definition, the clique nodes should be connected without cycles. Thus, there must exist one and only one path $(C_A, \mathcal{P})$ between the clique C_A and the pivot clique $\mathcal{P}$. With the variable-connected features, a pivot C'_A can be found in the path of $(C_A, \mathcal{P})$ that all the clique nodes in (C_A, C'_A) contain variable A and no clique between $(C'_A, \mathcal{P})$ will contain A anymore. Refer to the definition of distance, the length of path $(C'_A, \mathcal{P})$ is exactly $distance(A, \mathcal{P})$, i.e., the shortest propagating path. □

Consider a pivot $\mathcal{P}$ with a set of variables $\mathcal{P}.V$ attached. For a query Q, all those variables not in the pivot, $Q \backslash \mathcal{P}.V$, need to be propagated to the pivot. According to Theorem 2, there must exist a shortest path for each requested variable to pivot $\mathcal{P}$. Thus, we estimate the corresponding query plan cost on pivot P by the propagation distance

$$\mathbb{IP} = \sum_{v \in Q \backslash \mathcal{P}.V} distance(v, \mathcal{P}). \tag{8}$$

Finally, we choose the clique node as the query pivot, whose query plan achieves the lowest estimating cost, i.e., $\min \mathbb{IP}$, the pivot with the *Minimum Propagating Path*.

The previous *minimum propagating path* estimation only counts the number of propagating steps, while the real computation cost depends on the propagated message size in each step. Thus, another possible cost measure is *Minimum Propagating Size*, which takes the number of propagated variables into account.

5.2.2 Minimum Propagating Size

The *minimum propagating path* estimation returns a pivot with the minimum number of propagated variables in a query plan.

Consider any clique node C in the current state of clique tree. Let f be the message passed from C to the clique C', and let Q be the current query. For message f, the set of variables $f.V_e$ denotes all the variables eliminated in the joining tree of f. If there are some variables $U = Q \cap f.V_e, U \neq \emptyset$, we need to recompute message f to propagate the query variable U. Therefore, in the pivot selection, it is desirable that the propagating variables will be fewer in each step of message updating. The recomputing cost is estimated by the size of $|U|$ and the variables passed in the original clique tree, i.e., the total number of propagated variables.

Definition 7 (Propagating size). *Consider message f propagated 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. Then, for a query Q, the propagating size of message f is given by*

$$size(f) = |f.V_p| + |f.V_e \cap Q|. \tag{9}$$

Again, according to Theorem 2, only those messages in the requested propagating paths of query variables are needed to be updated. Thus, we consider all messages f_i in these propagating paths. The *minimum propagating size* estimation computes $\mathbb{IP} = \sum_i size(f_i)$ for each pivot, and returns the pivot with a minimum $\mathbb{IP}$.

5.2.3 Minimum Propagating Join

The *minimum propagating path* estimation considers the propagation distances, and the *minimum propagating size* estimates the query plan cost by the propagating message size. Now, we study the optimization issues in relational databases. Note that a product operation of collecting all the messages (2) is required in each propagating step, which is implemented by a `JOIN GROUP BY` operator of all the message views in RDBMS. The join operation cost is important in relational query evaluation. Thus, instead of considering the propagation distances or the propagating message sizes only, we further estimate the join cost of propagating the messages.

Definition 8 (Propagating join cost). *Consider message f passed from the clique C to C'. Let f_j be the message propagated toward C but not from C', and let $C.V$ be the set of variables attached in the clique C. Then, for a query Q, the propagating cost of message f is given by*

$$join(f) = |C.V| \cdot \prod_j |f_j.V_r|, \tag{10}$$

where $f_j.V_r$ is the set of requested variables in message f_j.

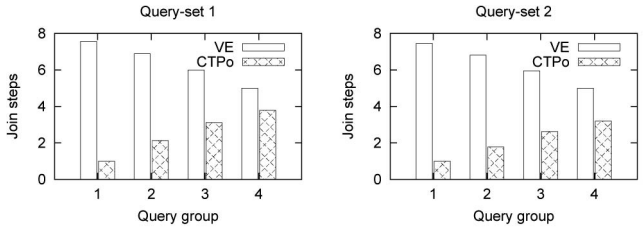


Fig. 8. JOIN-GROUP BY steps.

Recall that if there is no variable in f_j requested by query Q, then $f_j.V_r = f_j.V_p$, i.e., *passed* variables; if the variables requested by query Q are already *cached* in f_j, then $f_j.V_r \subseteq f_j.V_c$; otherwise, $f_j.V_r$ are returned by the previous propagating steps of computing message f_j. Finally, the *minimum propagating join* estimation sums up all the propagating steps in a query plan, $\mathbb{IP} = \sum_i join(f_i)$, and chooses the pivot with the minimum query plan cost $\mathbb{IP}$.

Therefore, according to our heuristic analysis advanced step by step, the minimum propagating join estimation would like to be the most reasonable one, which considers join costs in relational databases. Our experimental evaluation in Section 6.3 also verifies that the minimum propagating join estimation outperforms the other ones.

6 Experimental Evaluation

This section reports the experimental evaluation of the proposed approaches. A probabilistic inference query as clique tree propagation is developed with the materialized view provided by Oracle. The experiments run on a PC with two cores 2.0 GHz CPU, and 2 GB memory.

We use the Bayesian network in Fig. 3 as database schema with total eight relational tables. Rather than the binary variables, we test various data sizes with the variables' domain sizes up to 20 (that is, we have 25.6 billion tuples at most in the joint probability distribution of all the variables). In data set 1, we simulate probability distribution for each relation following the normal distribution. For example, consider uncertain data in tB in Fig. 1. Probability $P(B)$ is populated under the normal distribution with the constraint that $\sum_B P(B) = 1$. Moreover, in the real world, not every event has a probability to occur. Some events' probability value may equal to 0, e.g., $P(A = 0, B = 1) = 0$, which affects the efficiency of database query evaluation. Therefore, without loss of generality, in the data set 2, we randomly select x percent of the events as 0 probability events, where x percent ranges from 0 to 50 percent for different factors.

We also simulate the workload data of query inputs. Query variables are populated with the appearance frequency following the uniform and normal distribution. Specifically, in the experiment, we simulate the query-set 1 with uniform distribution and query-set 2 in normal distribution with variances of 1.0. The query-set 2 with a variance of 1.0 in normal distribution means that there are more frequent query variables in the workload. For each query set, we have four groups of query workloads Q_1, Q_2, Q_3, and Q_4 which contain 1, 2, 3, and 4 variable(s), respectively, in a query. In fact, an inference query with more variables (i.e., query the joint distribution with most variables in databases) is unusual in practice. In each query workload Q_i, we have 100 queries and evaluate the average time cost of these queries.

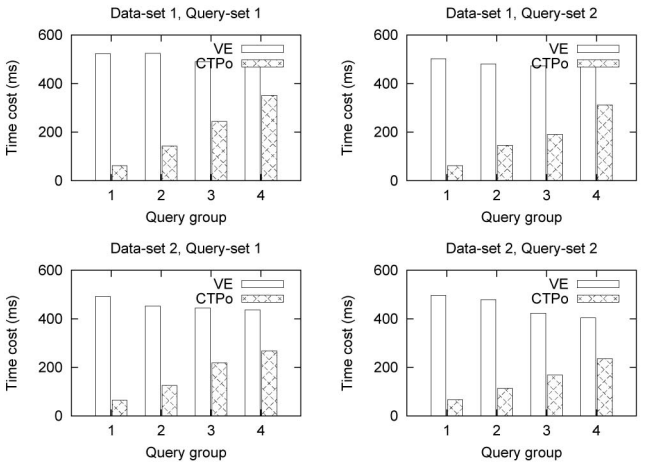


Fig. 9. Time cost.

6.1 VE versus CTP

In the first experiment, we study the Variable Elimination (VE) and the original Clique Tree Propagation (CTPo) without caching optimization. As shown in Fig. 8, the VE algorithm needs to join all the factors in the database, and has more JOIN-GROUP BY steps than those of CTPo. When there is only one query variable in query Q_1, the CTPo can always find a pivot with only one step of JOIN-GROUP BY operation. With the increase of query variables from Q_1 to Q_4, although the VE still needs to cover all the factors, the number of eliminating variables decreases. Therefore, the VE has fewer JOIN-GROUP BY operations when there are more variables in query Q_4. On the other hand, if more query variables are requested in the CTPo, then more messages with the requested variables are needed to be updated. Thus, the JOIN-GROUP BY operations of CTPo increase in query Q_4 with more variables. In fact, the worst case is that all the variables are requested by query, and entire message sets are needed to be recomputed. Fortunately, in the real application, it is unusual to query all the variables, i.e., the joint distribution of the whole database. As presented in Fig. 8, the original CTP requires significantly fewer steps of JOIN-GROUP BY operations than the VE, when the number of query variables is moderately large. Note that the numbers of VE steps are quite similar in all the query-sets, while the CTP algorithm has fewer steps in query-set 2 which has more frequent query variables. The difference between data set 1 and data set 2 is the 0 probability event, which does not affect the JOIN-GROUP BY steps. Therefore, we only show the result of data set 1 in Fig. 8, which is the same on data set 2.

Next, we present the time cost of VE and CTPo as well. As shown in Fig. 9, the time cost of these two approaches is quite similar to their corresponding JOIN-GROUP BY steps. The original CTP approach achieves lower time cost than the VE algorithm in all the four groups of queries. The results implicate that the reuse of computation among different queries by storing cliques and messages in the materialized views works well in relational databases.

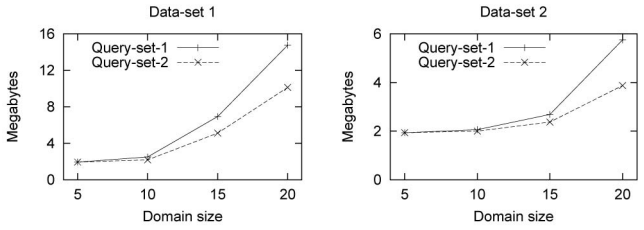


Fig. 10. Space cost evaluation in data size.

Furthermore, there are 0 probability events in the data set 2, which are not needed to evaluate in the query processing. Thus, the query shows low time cost in the data set 2.

We also observe the space cost of materialized data in the CTP approaches. The materialized storage only depends on the data set size, i.e., the domain size. Note that there is only one version of materialized data maintained in the CTP. Thus, different from the traditional query optimization with materialized views, we have no choice of different materialized data to achieve the best trade-off between space and efficiency. Fig. 10 shows the storage of materialized data in different data domain sizes. Similar to the probabilistic data sets themselves in this experiment, the sizes of materialized storage also increase in a nonlinear fashion with respect to the domain sizes. For example, the data size of $P(ABC)$ with domain size 5 for each variable is 5^3, while it increases to 10^3 when domain size changes to 10. Consequently, as shown in Fig. 10, we have similar space increases of materialized data as the data set itself on various domain sizes. An interesting result can be found that the query-set 2 shows lower space cost. Recall that the query-set 2 has more frequent query items, that is, there is a higher probability of reusing the cached data. Those materialized data with low reusing rate are then discarded; thus, the space cost is smaller.

6.2 CTP Optimization Evaluation

In this experiment, we apply our CTP caching optimization strategies by storing the most *frequently* queried variables in both cliques and messages, rather than the most *recently* queried variables in the original CTP. The original CTP (CTPo) and the CTP with caching optimization (CTPc) are compared in Figs. 11 and 12. We also show the CTP with minimum propagating join estimation (CTPe) in the same figures which will be discussed specifically in Section 6.3.

Again, we only show the JOIN-GROUP BY steps of data set 1 in Fig. 11, which are the same as the results on data set 2. Since we cache the frequent variables in the cliques and messages, the CTPc can reduce the JOIN-GROUP BY steps largely comparing with the original CTPo. Note that, in query-set 1, the JOIN-GROUP BY steps of CTPc in Q_4 are

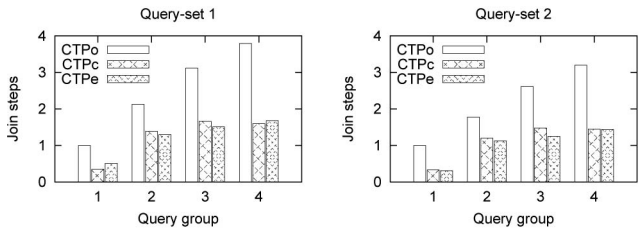


Fig. 11. JOIN-GROUP BY steps.

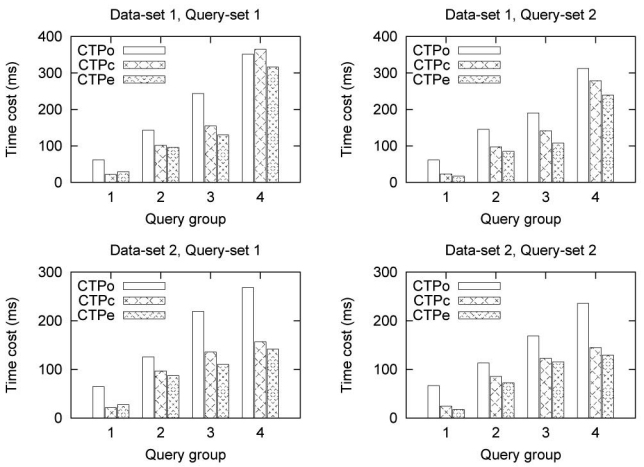


Fig. 12. Time cost.

fewer than Q_3. The reason is that the workload statistics of Q_i are different from each other, and the caching variables are updated incrementally according to the workload statistics. Those groups of queries with more frequent variables can share more computation with fewer propagating steps, in other words, fewer JOIN-GROUP BY steps.

Fig. 12 illustrates the efficiency of CTPc with caching frequent variables, which achieves lower time cost than CTPo. Similarly, the CTPo approach can also avoid the 0 probability events and shows low time cost in data set 2. Although CTPc has a small JOIN-GROUP BY steps in Q_4 in query-set 1, the corresponding time cost is still high. It is obvious that the time cost of each JOIN-GROUP BY is various. A query plan with shorter propagating steps might contain more query variables and require more JOIN-GROUP BY cost.

6.2.1 Scalability

Next, we study the scalability of our approach under different data sizes. In this experiment, we conduct the query (e.g., the query group 3) under four different domain sizes of variables in databases. Since we have eight variables in this database, the domain size with 5 means 5^8 (about 0.39 million) number of instances in the joint distribution

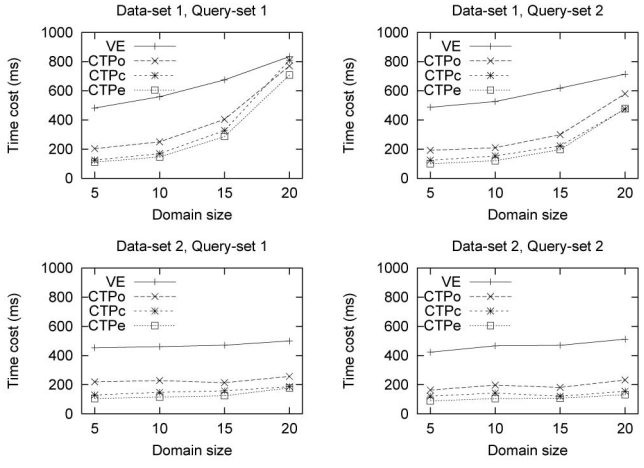


Fig. 13. Scalability in data size.

TABLE 2
Query Plan Estimations

	JOIN-GROUP BY Steps				Time Cost (ms)			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
CTPe path	0.75	1.36	1.75	2.09	145	277	841	742
CTPe size	1.26	1.49	1.78	2.09	253	304	827	733
CTPe join	**0.31**	**1.13**	**1.25**	**1.44**	**57**	**229**	**283**	**467**

view, and 20 denotes 20^8 (25.6 billion) in size of the joint distribution. As shown in Fig. 13, our CTP approaches achieve lower time cost and scale well under different data sizes. The CTPc with the variable caching strategy shows better performance than the original CTP, and the CTPe with cost estimation achieves the best performance.

6.3 Query Plan Estimation

In the last experiment, we evaluate the different strategies of query plan estimation, including the *minimum propagating path* (CTPe path), the *minimum propagating size* (CTPe size), and the *minimum propagating join* (CTPe join). Before performing a query, the query plan estimation for the pivot selection is conducted first. In the experiments, we find that the time cost of estimating processing is tiny comparing with the relational query cost and can be ignored. Due to the space limitation, we only present the results in query-set 2 in Table 2.

The time cost of query results by the minimum propagating join estimation (CTP *min join*) is the best in all these four groups of queries. This result verifies our analysis in Section 5. The reason is that the JOIN-GROUP BY operator plays the most expensive role in the query evaluation. Thus, the queries with CTP *min join* estimation can achieve lower time cost.

Note that the CTP *min path* does not have the minimum number of JOIN-GROUP BY steps in the results. The underlying reason is that the caching variables are updated incrementally, and the CTP *min path* estimation only greedily chooses the minimum propagating path for the current query. Consequently, after several queries with different pivots in these estimation approaches, the cached variables are various. Thus, the estimations for the next query are different as well due to the different historical caching data.

7 Related Work

Promising applications motivate the marriage of database and uncertainty community according to their similarity and correspondence on the theoretical foundation [17]. On the one hand, probabilistic models [20] are used to evaluate the selectivity of queries in database [21]. On the other hand, database techniques can also be utilized in the problems in uncertainty community, e.g., conducting probabilistic inference efficiently in database.

7.1 Probabilistic Inference as Propagation

Probabilistic inference queries can be directly answered in original Bayesian networks [6], [7] by using algorithms such as variable elimination (ve) [10], [22]. Shafer [11] describes a scheme of probabilistic inference as message propagation, i.e., the Shenoy-Shafer propagation architecture [23]. The above direct inference computing techniques like VE can be adopted to compute the messages in the Shenoy-Shafer propagation architecture. Specifically, the clique tree propagation (CTP) [12] and lazy propagation [13] algorithms combine the Shenoy-Shafer message propagation scheme with VE for message computation, i.e., based on the same principle as VE except with an advanced message caching scheme. Note that the variables of an arbitrary inference query may not be the subset of any clique in the propagation tree. Thereby, messages are collected (and multiplied) from a set of cliques whose union covers all the query variables. Instead of propagating in the original clique tree, Xu [24] builds new clique nodes which cover all the query variables. Since the clique tree structure (as well as the relational schema in database implementation) changes frequently when processing different queries by Xu's approach, we implement the propagation scheme of the original CTP [12]. Before the propagation computation of messages starts, Butz et al. [25] identify those messages that are not necessarily to be physically computed due to the explored independence of corresponding variables. Our approach in Section 4, however, identifies those messages with common query variables between two different queries in the workload. Moreover, Madsen [26] implements arc reversal (AR) [27] and symbolic probabilistic inference (SPI) [28] algorithms for the propagation-based inference in addition to the VE-based CTP. In our current work, since the state-of-the-art work [5] of probabilistic inference in databases uses VE instead of AR, in order to conduct fair comparison with this previous work, we choose to implement CTP, which is based on the same principle as VE.

7.2 Probabilistic Inference in Relational Database

The connections between relational databases and Bayesian networks have already been noticed [29], [30]. Wong et al. [17] present the correspondence between relational databases and Bayesian networks. Specifically, a joint distribution corresponds to a relation in databases. The marginalization and multiplication operations correspond to the projection and natural join operations in databases, respectively. The probabilistic conditional independency corresponds to embedded multivalued dependency in databases [31]. In addition, Wong et al. [8] propose a method for implementing a probabilistic inference by transforming a Markov network into a relational database. As a sequel work, Wong et al. [9] extend their work to support probabilistic reasoning in Bayesian networks in relational databases. Bravo and Ramakrishnan [5] present a broad class of aggregate queries, Marginalize a Product Function, and implement the VE algorithm in relational databases with the consideration of query optimization issues. The VE algorithm answers one query at a time without computation sharing among different queries, while in this study, we provide computation reuse among a sequence of queries.

7.3 Optimizing Queries Using Materialized Views

Chaudhuri et al. [32] first study the incorporation of materialized views within the query optimization, by using

the cost-based dynamic programming. Goldstein and Larson [33] also develop an approach to optimize the queries with materialized views, where general queries with selections, joins and group by are studied. Chirkova and Li [34] find a set of views with minimum sizes, which can compute the answers to the query. In this paper, however, we only have one version of materialized data determined by the CTP. Thus, we have no choice of different versions of materialized views to trade off between the space and time cost. Our pivot selection strategies choose the best query plan on the available materialized data with the minimum estimated cost.

8 Conclusions

In this paper, 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. Finally, in order to select the pivot with the lowest query cost, we study query plan estimation strategies. The experimental results demonstrate the effectiveness of our caching and sharing strategies among the frequent queries.

Acknowledgments

The authors thank Prof. Nevin L. Zhang at the Hong Kong University of Science and Technology for the kind discussion in the VE and CTP problems. Shaoxu Song and Lei Chen are supported by Hong Kong RGC NSFC RGC Joint Project under Grant No. N_HKUST612/09, and NSFC Grant Nos. 60736013, 60933011, 60873022, and 60903053. Jeffrey Xu Yu is supported by Hong Kong RGC Project Grant No. 419008.

References

[1] A. Faradjian, J. Gehrke, and P. Bonnet, "GADT: A Probability Space ADT for Representing and Querying the Physical World," *Proc. Int'l Conf. Data Eng.*, 2002.

[2] C. Böhm, A. Pryakhin, and M. Schubert, "The Gauss-Tree: Efficient Object Identification in Databases of Probabilistic Feature Vectors," *Proc. Int'l Conf. Data Eng.*, 2006.

[3] M.F. Mokbel, C.-Y. Chow, and W.G. Aref, "The New Casper: Query Processing for Location Services without Compromising Privacy," *Proc. Int'l Conf. Very Large Data Bases (VLDB '06)*, 2006.

[4] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Querying Imprecise Data in Moving Object Environments," *IEEE Trans. Knowledge and Data Eng.*, vol. 16, no. 9, pp. 1112-1127, Sept. 2004.

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

[6] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.

[7] F.V. Jensen, *Introduction to Bayesian Networks*. Springer-Verlag, 1996.

[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, 1995.

[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 D. Poole, "Exploiting Causal Independence in Bayesian Network Inference," *J. Artificial Intelligence Research*, vol. 5, pp. 301-328, 1996.

[11] G. Shafer, *Probabilistic Expert Systems*. Soc. for Industrial and Applied Math., 1996.

[12] 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.

[13] A.L. Madsen and F.V. Jensen, "Lazy Propagation: A Junction Tree Inference Algorithm Based on Lazy Evaluation," *Artificial Intelligence*, vol. 113, nos. 1/2, pp. 203-245, 1999.

[14] F.V. Jensen and F. Jensen, "Optimal Junction Trees," *Proc. Conf. Uncertainty in Artificial Intelligence (UAI)*, pp. 360-366, 1994.

[15] S. Chaudhuri and K. Shim, "Including Group-By in Query Optimization," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 354-366, 1994.

[16] S. Chaudhuri and K. Shim, "Optimizing Queries with Aggregate Views," *Proc. Int'l Conf. Extending Database Technology (EDBT)*, pp. 167-182, 1996.

[17] S.K.M. Wong, C.J. Butz, and D. Wu, "On the Implication Problem for Probabilistic Conditional Independency," *IEEE Trans. Systems, Man, and Cybernetics, Part A*, vol. 30, no. 6, pp. 785-805, Nov. 2000.

[18] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum, "Probabilistic Ranking of Database Query Results," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 888-899, 2004.

[19] R. Agrawal, T. Imielinski, and A.N. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. ACM SIGMOD*, pp. 207-216, 1993.

[20] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, "Learning Probabilistic Relational Models," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI)*, pp. 1300-1309, 1999.

[21] L. Getoor, B. Taskar, and D. Koller, "Selectivity Estimation Using Probabilistic Models," *Proc. ACM SIGMOD*, pp. 461-472, 2001.

[22] N.L. Zhang, "Computational Properties of Two Exact Algorithms for Bayesian Networks," *Applied Intelligence*, vol. 9, no. 2, pp. 173-183, 1998.

[23] P.P. Shenoy and G. Shafer, "Axioms for Probability and Belief-Function Proagation," *Proc. Ann. Conf. Uncertainty in Artificial Intelligence (UAI)*, pp. 169-198, 1988.

[24] H. Xu, "Computing Marginals for Arbitrary Subsets from Marginal Representation in Markov Trees," *Artificial Intelligence*, vol. 74, no. 1, pp. 177-189, 1995.

[25] 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.

[26] A.L. Madsen, "Variations over the Message Computation Algorithm of Lazy Propagation," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 36, no. 3, pp. 636-648, June 2006.

[27] R.D. Shachter, "Evaluating Influence Diagrams," *Operations Research*, vol. 34, no. 6, pp. 871-882, 1986.

[28] 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, 1990.

[29] W.X. Wen, "From Relational Databases to Belief Networks," *Proc. Conf. Uncertainty in Artificial Intelligence (UAI)*, pp. 406-413, 1991.

[30] F.M. Malvestuto, "A Unique Formal System for Binary Decompositions of Database Relations, Probability Distributions, and Graphs," *Information Sciences*, vol. 59, nos. 1/2, pp. 21-52, 1992.

[31] S.K.M. Wong, "An Extended Relational Data Model for Probabilistic Reasoning," *J. Intelligent Information Systems*, vol. 9, no. 2, pp. 181-202, 1997.

[32] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, "Optimizing Queries with Materialized Views," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 190-200, 1995.

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

[34] R. Chirkova and C. Li, "Materializing Views with Minimal Size to Answer Queries," *Proc. Symp. Principles of Database Systems (PODS)*, pp. 38-48, 2003.

Shaoxu Song is currently working toward the PhD degree in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His research interests include data quality and data dependency. He is a student member of the IEEE.

Lei Chen received the BS degree in computer science and engineering from Tianjin University, China, in 1994, the MA degree from Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from University of Waterloo, Canada, in 2005. He is currently an assistant professor in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His research interests include uncertain databases, graph databases, multimedia and time series databases, and sensor and peer-to-peer databases. He is a member of the IEEE.

Jeffrey Xu Yu received the BE, ME, and the PhD degrees in computer science from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. He held teaching positions in the Institute of Information Sciences and Electronics, University of Tsukuba, Japan, and the Department of Computer Science, The Australian National University. Currently, he is a professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. He served as an associate editor of the *IEEE Transactions on Knowledge and Data Engineering*, and is serving as a VLDB Journal editorial board member, and an ACM SIGMOD information director. His current main research interest includes graph database, graph mining, keyword search in relational databases, XML database, and Web-technology. He has published more than 190 papers including papers published in reputed journals and major international conferences. He is a senior member of the IEEE.