Efficiently Answering Probabilistic Threshold Top-k Queries on Uncertain Data (Extended Abstract)

Ming Hua† Jian Pei† Wenjie Zhang‡ Xuemin Lin‡

†*Simon Fraser University, Canada*
{mhua, jpei}@cs.sfu.ca

‡*The University of New South Wales & NICTA*
{zhangw, lxue}@cse.unsw.edu.au

***Abstract*— In this paper, we propose a novel type of probabilistic threshold top-k queries on uncertain data, and give an exact algorithm. More details can be found in [4].**

I. Probabilistic Threshold Top-k Queries

We consider uncertain data in the *possible worlds* semantics model [1], [5], [7], which is also adopted by some recent studies on uncertain data processing, such as [8], [2], [6].

Generally, an *uncertain table* T contains a set of (uncertain) tuples, where each tuple $t \in T$ is associated with a *membership probability* value $Pr(t) > 0$. When there is no confusion, we also call an uncertain table simply a table.

A *generation rule* on a table T specifies a set of exclusive tuples in the form of $R : t_{r_1} \oplus \cdots \oplus t_{r_m}$ where $t_{r_i} \in T$ $(1 \leq i \leq m)$ and $\sum_{i=1}^{m} Pr(t_{r_i}) \leq 1$. The rule R constrains that, among all tuples $t_{r_1}, \ldots, t_{r_m}$ involved in the rule, at most one tuple can appear in a possible world.

As [8], [2], we assume that each tuple is involved in at most one generation rule. For a tuple t not involved in any generation rule, we can make up a trivial rule $R_t : t$. Therefore, conceptually, an uncertain table T comes with a set of generation rules $\mathcal{R}_T$ such that each tuple is involved in one and only one generation rule in $\mathcal{R}_T$. We write $t \in R$ if tuple t is involved in rule R. The *probability* of a rule is the sum of the membership probability values of all tuples involved in the rule, denoted by $Pr(R) = \sum_{t \in R} Pr(t)$.

The *length* of a rule is the number of tuples involved in the rule, denoted by $|R| = |\{t | t \in R\}|$. A generation rule R is a *singleton rule* if $|R| = 1$. R is a *multi-tuple rule* if $|R| > 1$. A tuple is *dependent* if it is involved in a multi-tuple rule, otherwise, it is *independent*.

For a subset of tuples $S \subseteq T$ and a generation rule R, we denote the tuples involved in R and appearing in S by $R \cap S$. A *possible world* W is a subset of T such that for each generation rule $R \in \mathcal{R}_T$, $|R \cap W| = 1$ if $Pr(R) = 1$, and $|R \cap W| \leq 1$ if $Pr(R) < 1$. We denote by $\mathcal{W}$ the set of all possible worlds.

Clearly, for an uncertain table T with a set of generation rules $\mathcal{R}_T$, the number of all possible worlds is $|\mathcal{W}| = \prod_{R \in \mathcal{R}_T, Pr(R)=1} |R| \prod_{R \in \mathcal{R}_T, Pr(R)<1} (|R| + 1)$. The number of possible worlds on a large table can be huge.

Each possible world is associated with an *existence probability* $Pr(W)$ that the possible world happens. Following with the basic probability principles, we have $Pr(W) = \prod_{R \in \mathcal{R}_T, |R \cap W|=1} Pr(R \cap W) \prod_{R \in \mathcal{R}_T, R \cap W = \emptyset} (1 - Pr(R))$. Apparently, for a possible world W, $Pr(W) > 0$. Moreover, $\sum_{W \in \mathcal{W}} Pr(W) = 1$.

A *top-k query* $Q^k(P, f)$ contains a *predicate* P, a *ranking function* f, and an integer $k > 0$. When Q is applied on a set of certain tuples, the tuples satisfying predicate P are ranked according to ranking function f, and the top-k tuples are returned. For tuples t_1, t_2, $t_1 \preceq_f t_2$ if t_1 is ranked higher than or equal to t_2 according to ranking function f. $\preceq_f$, called the *ranking order*, is a total order on all tuples.

Since a possible world W is a set of tuples, a top-k query Q can be applied to W directly. We denote by $Q^k(W)$ the top-k tuples returned by a top-k query Q on a possible world W. $Q^k(W)$ contains k tuples.

A *probabilistic threshold top-k query* (*PT-k* query for short) on an uncertain table T consists of a top-k query Q and a *probability threshold* p $(0 < p \leq 1)$. For each possible world W, Q is applied and a set of k tuples $Q^k(W)$ is returned. For a tuple $t \in T$, the *top-k probability* of t is the probability that t is in $Q^k(W)$ in all $W \in \mathcal{W}$, that is, $Pr^k_{Q,T}(t) = \sum_{W \in \mathcal{W}, t \in Q^k(W)} Pr(W)$. When Q and T are clear from context, we often write $Pr^k_{Q,T}(t)$ as $Pr^k(t)$ for the interest of simplicity.

The *answer set* to a PT-k query is the set of all tuples whose top-k probability values are at least p. That is, $Answer(Q, p, T) = \{t | t \in T, Pr^k_Q(t) \geq p\}$. We are interested in how to compute efficiently the answer set for a PT-k query on an uncertain table.

II. An Exact Algorithm

Hereafter, by default we consider a top-k query $Q^k(P, f)$ on an uncertain table T. $P(T) = \{t | t \in T \wedge P(t) = true\}$ is the set of tuples satisfying the query predicate. $P(T)$ is also an uncertain table where each tuple in $P(T)$ carries the same membership probability as in T. Moreover, a generation rule R in T is projected to $P(T)$ by removing all tuples from R that are not in $P(T)$. Then, the problem of answering the PT-k query is to find the tuples in $P(T)$ whose top-k probability values pass the probability threshold.

Apparently, $Answer(Q, p, T) = Answer(Q, p, P(T))$. We only need to consider $P(T)$ in answering a top-k query.

A. The Dominant Set Property

For a tuple $t \in P(T)$ and a possible world W such that $t \in W$, whether $t \in Q^k(W)$ depends only on how many other tuples in $P(T)$ ranked higher than t appear in W. Technically,

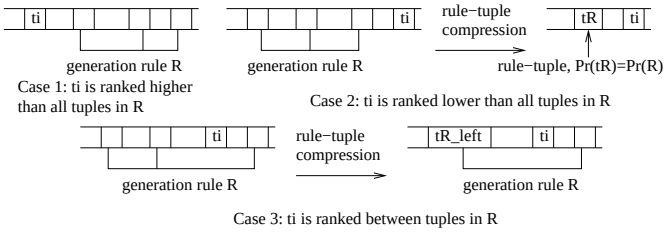

Fig. 1. Computing $Pr^k(t_i)$ for one tuple t_i.

for a tuple $t \in P(T)$, the *dominant set* of t is the subset of tuples in $P(T)$ that are ranked higher than t, i.e., $S_t = \{t' | t' \in P(T) \wedge t' \prec_f t\}$.

Theorem 1 (The dominant set property): For a tuple $t \in T$, $Pr^k_{Q,T}(t) = Pr^k_{Q,S_t}(t)$. ■

Our PT-k query answering algorithm scans the tuples in $P(T)$ in the ranking order, and derives the top-k probability of a tuple t based on the tuples preceding t in the ranking order. Generation rules are handled by the rule-tuple compression technique. The probability threshold is used to prune tuples whose top-k probability values fail the threshold.

B. The Basic Case

We first consider the basic case where all tuples are independent. Let $L = t_1 \cdots t_n$ be the list of all tuples in table $P(T)$ in the ranking order. Then, in a possible world W, a tuple $t_i \in W$ $(1 \leq i \leq n)$ is ranked at the j-th $(j > 0)$ position if and only if exactly $(j-1)$ tuples in the dominant set $S_{t_i} = \{t_1, \ldots, t_{i-1}\}$ also appear in W.

The *position probability* $Pr(t_i, j)$ is the probability that tuple t_i is ranked at the j-th position in possible worlds. Moreover, the *subset probability* $Pr(S_{t_i}, j)$ is the probability that j tuples in S_{t_i} appear in possible worlds.

Trivially, we have $Pr(\emptyset, 0) = 1$ and $Pr(\emptyset, j) = 0$ for $0 < j \leq n$. Then, $Pr(t_i, j) = Pr(t_i) Pr(S_{t_{i-1}}, j-1)$. Apparently, the top-$k$ probability of t_i is given by

$$Pr^k(t_i) = \sum_{j=1}^{k} Pr(t_i, j) = Pr(t_i) \sum_{j=1}^{k} Pr(S_{t_{i-1}}, j-1) \quad (1)$$

Particularly, when $i \leq k$, we have $Pr^k(t_i) = Pr(t_i)$.

Theorem 2: In the basic case, for $1 \leq i, j \leq |T|$, $Pr(S_{t_i}, 0) = Pr(S_{t_{i-1}}, 0)(1 - Pr(t_i)) = \prod_{j=1}^{i}(1 - Pr(t_i))$, $Pr(S_{t_i}, j) = Pr(S_{t_{i-1}}, j-1)Pr(t_i) + Pr(S_{t_{i-1}}, j)(1 - Pr(t_i))$. ■

C. Handling Generation Rules

In the basic case, Theorem 2 can be used to compute the top-k probability values for all tuples in time $O(kn)$, where n is the number of tuples in the uncertain table. However, a general case may contain some multi-tuple generation rules.

1) Rule-Tuple Compression: Consider $P(T) = t_1 \cdots t_n$ in the ranking order, i.e., $t_i \preceq_f t_j$ for $i < j$. Let us compute $Pr^k(t_i)$ for a tuple $t_i \in P(T)$. A multi-tuple generation rule $R : t_{r_1} \oplus \cdots \oplus t_{r_m}$ $1 \leq r_1 < \cdots < r_m \leq n$ can be handled in one of the following cases (see Figure 1 for illustration.)

Case 1: $t_i \preceq_f t_{r_1}$, i.e., t_i is ranked higher than or equal to all tuples in R. According to Theorem 1, R can be ignored.

Case 2: $t_{r_m} \prec_f t_i$, i.e., t_i is ranked lower than all tuples in R. R is called *completed* with respect to t_i. At most one tuple in R can appear in a possible world. According to Theorem 1, we can combine all tuples in R into a *rule-tuple* t_R with membership probability $Pr(R)$.

Corollary 1 (Rule-tuple compression): For a tuple $t \in P(T)$ and a multi-tuple rule R, if $\forall t' \in R$, $t' \prec_f t$, then $Pr^k_{Q,T}(t) = Pr^k_{Q,T(R)}(t)$ where $T(R) = (T - \{t | t \in R\}) \cup \{t_R\}$, tuple t_R takes any value such that $t_R \prec_f t$, $Pr(t_R) = Pr(R)$, and other generation rules in T remain the same in $T(R)$. ■

Case 3: $t_{r_1} \prec_f t_i \preceq_f t_{r_m}$, i.e., t_i is ranked in between tuples in R. R is called *open* with respect to t_i. Among the tuples in R ranked better than t_i, let $t_{r_{m_0}} \in R$ be the lowest ranked tuple i.e., $r_{m_0} = \max_{l=1}^{m}\{r_l < i\}$. The tuples involved in R can be divided into two parts: $R_{left} = \{t_{r_1}, \ldots, t_{r_{m_0}}\}$ and $R_{right} = \{t_{r_{m_0}+1}, \ldots, t_{r_m}\}$. $Pr^k(t_i)$ is affected by tuples in R_{left} only and not by those in R_{right}.

Two subcases may arise. First, if $t_i \notin R$, similar to Case 2, we can compress all tuples in R_{left} into a rule-tuple $t_{r_1,\ldots,r_{m_0}}$ where membership probability $Pr(t_{r_1,\ldots,r_{m_0}}) = \sum_{j=1}^{m_0} Pr(t_{r_j})$, and compute $Pr^k(t_i)$ using Corollary 1.

Second, $t_i \in R$, i.e., $t_i = t_{r_{m_0}+1}$. In a possible world where t_i appears, any tuples in R cannot appear. Thus, to determine $Pr^k(t_i)$, according to Theorem 1, we only need to consider the tuples ranked higher than t_i and not in R, i.e., $S_{t_i} - \{t' | t' \in R\}$.

Corollary 2 (Tuple in rule): For a tuple $t \in R$ such that $P(t) = true$ and $|R| > 1$, $Pr^k_{Q,T}(t) = Pr^k_{Q,T'}(t)$ where uncertain table $T' = (S_{t_i} - \{t' | t' \in R\}) \cup \{t\}$. ■

For a tuple t and its dominant set S_t, we can check t against the multi-tuple rules one by one. Each multi-tuple rule can be handled by one of the above three cases, and dependent tuples in S_t can be either compressed into some rule-tuples or removed due to the involvement in the same rule as t. After the rule-tuple compression, the resulted set is called the *compressed dominant set* of t, denoted by $T(t)$. Based on the above discussion, for a tuple $t \in P(t)$, all tuples in $T(t) \cup \{t\}$ are independent, $Pr^k_{Q,T}(t) = Pr^k_{Q,T(t)\cup\{t\}}(t)$. We can apply Theorem 2 to calculate $Pr^k(t)$ by scanning $T(t)$ once.

We can sort all tuples in $P(T)$ into a sorted list L in the ranking order. For each tuple t_i, by one scan of the tuples in L before t_i, we obtain the compressed dominant set $T(t_i)$ where all tuples are independent. Then, we can compute $Pr^k(t_i)$ on $T(t_i) \cup \{t_i\}$ using Theorem 2. In this way, the top-k probability for all tuples can be computed in $O(n^2)$ time where n is the number of tuples in the uncertain table.

2) Scan Reduction by Prefix Sharing: Equation 1 indicates that to compute $Pr^k(t_i)$ using subset probability $Pr(S_{t_{i-1}}, j)$, the order of tuples in $S_{t_{i-1}}$ does not matter. This gives us the space to order the tuples in compressed dominant sets of different tuples so that the prefixes and the corresponding subset probability values can be shared as much as possible.

Consider the list $L = t_1 \cdots t_n$ of all tuples in $P(T)$ and a tuple t_i in L. Two observations may help the reordering.

First, for a tuple t that is independent or is a rule-tuple of

a completed rule with respect to t_i (Case 2 in Section II-C.1), t is in $T(t')$ for any tuple $t' \succ_f t_i$. Thus, t should be ordered before any rule-tuple of a rule open with respect to t_i (Case 3 in Section II-C.1).

Second, there can be multiple rules open with respect to t_i. Each such a rule R_j has a rule-tuple $t_{R_{j_{left}}}$, which will be combined with the next tuple $t' \in R_j$ to update the rule-tuple. Thus, if t' is close to t_i, $t_{R_{j_{left}}}$ should be ordered close to the rear so that the rule-tuple compression affects the shared prefix as little as possible. In other words, those rule-tuples of rules open with respect to t_i should be ordered in their next tuple indices descending order.

An *aggressive method* to reorder the tuples is to always put all independent tuples and rule-tuples of completed rules before rule-tuples of open rules, and order rule-tuples of open rules according to their next tuples in the rules. On the other hand, a *lazy method* always reuses the maximum common prefix in $T(t_{i-1})$, and reorders only the tuples not in the common prefix using the above two observations. We can show that the lazy method is always not worse than the aggressive method.

D. Pruning Techniques

Can we avoid retrieving or checking all tuples satisfying the query predicates? Some existing methods such as the well known TA algorithm [3] can retrieve in batch tuples satisfying the predicate in the ranking order. Using such a method, we can retrieve tuples in $P(T)$ progressively. Now, the problem becomes how we can use the tuples seen so far to prune some tuples ranked lower in the ranking order.

Hereafter, by default we consider a PT-k query using probability threshold p. We give three pruning rules which can determine that some tuples not checked yet cannot pass the probability threshold. The tuple retrieval method (e.g., an adaption of the TA algorithm [3]) uses the pruning rules in the retrieval. Once it can determine all remaining tuples in $P(T)$ fail the probability threshold, the retrieval can stop.

Please note that we still have to retrieve a tuple t failing the probability threshold if some tuples ranked lower than t may satisfy the threshold, since t may be in the compressed dominant sets of those promising tuples.

Theorem 3 (Pruning by membership probability): $Pr^k(t) \leq Pr(t)$. Moreover, for an independent tuple t, if $Pr^k(t) < p$, then (1) for any independent tuple t' such that $t \preceq_f t'$ and $Pr(t') \leq Pr(t)$, $Pr^k(t') < p$; and (2) for any multi-tuple rule R such that t is ranked higher than all tuples in R and $Pr(R) \leq Pr(t)$, $Pr^k(t'') < p$ for any $t'' \in R$. ■

To use Theorem 3, we maintain the largest membership probability p_{member} of all independent tuples and rule-tuples for completed rules checked so far whose top-k probability values fail the probability threshold. All tuples identified by the above pruning rule should be marked failed.

Theorem 4 (Pruning by tuples in the same rule): For tuples t, t' involved in the same multi-tuple rule R, if $t \preceq_f t'$, $Pr(t) \geq Pr(t')$, and $Pr^k(t) < p$, then $Pr^k(t') < p$. ■

Based on the above pruning rule, for each rule R open with respect to the current tuple, we maintain the largest membership probability of the tuples seen so far in R whose top-k probability values fail the threshold. Any tuples in R that have not been seen should be tested against this largest membership probability.

Our last pruning rule follows with that the sum of the top-k probability values of all tuples is k, i.e., $\sum_{t \in T} Pr^k(t) = k$.

Theorem 5 (Pruning by total top-k probability): Let A be a set of tuples whose top-k probability values pass the probability threshold p. If $\sum_{t \in A} Pr^k(t) > k - p$, then for every tuple $t' \notin A$, $Pr^k(t') < p$. ■

Input: an uncertain table T, a set of generation rules $\mathcal{R}_T$, a top-k query $Q^k(P, f)$, and a probability threshold p;
Output: $Answer(Q, p, T)$;
Method:

```
1: retrieve tuples in P(T) in the ranking order one by one,
   for each t_i ∈ P(T) do
2:   compute T(t_i) by rule-tuple compression;
3:   compute subset probability values and Pr^k(t_i);
4:   if Pr^k(t_i) ≥ p then output t_i;
5:   check whether t_i can be used to prune future tuples;
6:   if all remaining tuples in P(T) fail the probability
     threshold then exit;
   end for
```

Fig. 2. The exact algorithm.

The exact algorithm for PT-k query answering is shown in Figure 2. The complexity of the algorithm can be analyzed as follows. For a multi-tuple rule $R : t_{r_1} \oplus \cdots \oplus t_{r_m}$ where $t_{r_1}, \ldots, t_{r_m}$ are in the ranking order, let $span(R) = r_m - r_1$. When tuple t_{r_l} $(1 < l \leq m)$ is processed, we need to remove rule-tuple $t_{r_1,\ldots,r_{l-1}}$, and compute the subset probability values of the updated compressed dominant sets. When the next tuple not involved in R is processed, $t_{r_1,\ldots,r_{l-1}}$ and t_{r_l} are combined. Thus, in the worst case, each multi-tuple rule causes the computation of $O(2k \cdot span(R))$ subset probability values. Moreover, in the worst case where each tuple $P(T)$ passes the probability threshold, all tuples in $P(T)$ have to be read at least once. The time complexity of the whole algorithm is $O(kn + k \sum_{R \in \mathcal{R}_T} span(R))$.

As indicated by our experimental results, in practice the three pruning rules are effective. Often, only a very small portion of the tuples in $P(T)$ are retrieved and checked before the exact answer to a PT-k query is obtained.

REFERENCES

[1] S. Abiteboul *et al.* On the representation and querying of sets of possible worlds. In *SIGMOD'87*.

[2] O. Benjelloun *et al.* Uldbs: databases with uncertainty and lineage. In *VLDB'06*.

[3] R. Fagin *et al.* Optimal aggregation algorithms for middleware. In *PODS'01*.

[4] M. Hua *et al.* Efficient answering probabilistic threshold top-k queries on uncertain data (full version). Technical report TR 2007-26, School of Computing Science, Simon Fraser University.

[5] T. Imielinski and Jr. Witold Lipski. Incomplete information in relational databases. *Journal of ACM*, 31(4):761–791, 1984.

[6] J. Pei *et al.* Probabilistic skylines on uncertain data. In *VLDB'07*.

[7] A. D. Sarma *et al.* Working models for uncertain data. In *ICDE'06*.

[8] M. A. Soliman *et al.* Top-k query processing in uncertain databases. In *ICDE'07*.