Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach[*]

Ming Hua   Jian Pei
Simon Fraser University
Burnaby, BC, Canada
{mhua, jpei}@cs.sfu.ca

Wenjie Zhang   Xuemin Lin
The University of New South Wales & NICTA
Sydney, Australia
{zhangw,lxue}@cse.unsw.edu.au

ABSTRACT

Uncertain data is inherent in a few important applications such as environmental surveillance and mobile object tracking. Top-k queries (also known as ranking queries) are often natural and useful in analyzing uncertain data in those applications. In this paper, we study the problem of answering probabilistic threshold top-k queries on uncertain data, which computes uncertain records taking a probability of at least p to be in the top-k list where p is a user specified probability threshold. We present an efficient exact algorithm, a fast sampling algorithm, and a Poisson approximation based algorithm. An empirical study using real and synthetic data sets verifies the effectiveness of probabilistic threshold top-k queries and the efficiency of our methods.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query processing

General Terms

Algorithm, Performance, Experimentation

Keywords

Uncertain Data, Probabilistic Threshold Top-k Queries, Query Processing

1. INTRODUCTION

In a few emerging important applications such as environmental surveillance using large scale sensor networks, uncertainty is inherent in data due to various factors like incompleteness of data, limitations of equipment, and delay or loss

RID	Loc.	Time	Sensor-id	Duration	Conf.
$R1$	A	6/2/06 2:14	$S101$	25 min	0.3
$R2$	B	7/3/06 4:07	$S206$	21 min	0.4
$R3$	B	7/3/06 4:09	$S231$	13 min	0.5
$R4$	A	4/12/06 20:32	$S101$	12 min	1.0
$R5$	E	3/13/06 22:31	$S063$	17 min	0.8
$R6$	E	3/13/06 22:28	$S732$	11 min	0.2

Table 1: Panda presence records.

in data transfer. In those applications, top-k queries (or as known as ranking queries) are often natural and useful in analyzing uncertain data.

EXAMPLE 1 (MOTIVATION). *Sensors are often used to detect presence of endangered, threatened, or special concern risk categories of animals in remote or preserved regions. Due to limitations of sensors, detections cannot be accurate all the time. Instead, detection confidence is often estimated. Table 1 lists a set of synthesized records of presence of pandas detected by sensors. Once a sensor detects a suspect of presence, it records the duration that the suspect stays in the detection range of the sensor.*

In some locations where the targets are active, multiple sensors are deployed to improve the detection quality. Two sensors in the same location (e.g., $S206$ and $S231$, as well as $S063$ and $S732$ in Table 1) may detect the presence of a suspect at the (approximately) same time, such as records $R2$ and $R3$, as well as $R5$ and $R6$. In such a case, if the durations detected by multiple sensors are inconsistent, at most one sensor can be correct.

The uncertain data in Table 1 carries the possible world semantics [1, 16, 23, 8]. The data can be viewed as the summary of a set of possible worlds. The possible worlds are governed by some underlying generation rules which constrain the presence of tuple instances. In Table 1, the fact that $R2$ and $R3$ cannot be true at the same time can be captured by a generation rule $R2 \oplus R3$. Another generation rule is $R5 \oplus R6$. Table 2 shows all possible worlds and their existence probability values.

Top-k queries can be used to analyze uncertain data. For example, a scientist may be interested in the top-2 longest durations that a suspect stays in a location at a time. In different possible worlds the answers to this question are different. The third column of Table 2 lists the top-2 records in all possible worlds according to the duration attribute.

It is interesting to examine the probability that a tuple is in the top-2 lists of all possible worlds, as shown in Ta-

Possible world	Probability	Top-2 on Duration
$W1 = \{R1, R2, R4, R5\}$	0.096	$R1, R2$
$W2 = \{R1, R2, R4, R6\}$	0.024	$R1, R2$
$W3 = \{R1, R3, R4, R5\}$	0.12	$R1, R5$
$W4 = \{R1, R3, R4, R6\}$	0.03	$R1, R3$
$W5 = \{R1, R4, R5\}$	0.024	$R1, R5$
$W6 = \{R1, R4, R6\}$	0.006	$R1, R4$
$W7 = \{R2, R4, R5\}$	0.224	$R2, R5$
$W8 = \{R2, R4, R6\}$	0.056	$R2, R4$
$W9 = \{R3, R4, R5\}$	0.28	$R5, R3$
$W10 = \{R3, R4, R6\}$	0.07	$R3, R4$
$W11 = \{R4, R5\}$	0.056	$R5, R4$
$W12 = \{R4, R6\}$	0.014	$R4, R6$

Table 2: The possible worlds of Table 1.

RID	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$
Probability	0.3	0.4	0.38	0.202	0.704	0.014

Table 3: The top-2 probability values of records in Table 1.

ble 3. The method to calculate the probability values will be discussed in Section 2. To answer the top-2 query on the uncertain data, it is helpful to find the tuples whose probability values in the top-2 lists are at least p, where p is a user-specified probability threshold. In this example, if $p = 0.35$, then $\{R2, R3, R5\}$ should be returned. ∎

Example 1 demonstrates the probabilistic threshold top-k queries which will be studied in this paper. Generally, different from the top-k queries on certain data, probabilistic top-k queries on uncertain data pose several interesting challenges that will be addressed in this study.

Challenge 1: What does a probabilistic top-k query mean? A top-k query on certain data returns the best k results in the ranking function. However, top-k queries on uncertain data may be formulated differently to address different application interests.

For example, in [26], an *uncertain top-k query* (*U-TopK query* for short) returns a list of k records (i.e., a record vector of length k) that has the highest probability to be the top-k list in all possible worlds. In Table 1, $\langle R5, R3 \rangle$ should be returned. An *uncertain k ranks query* (*U-KRanks query* for short) returns a list of k records such that the i-th record has the highest probability to be the i-th best record in all possible worlds. In the case of Table 1, $\langle R5, R5 \rangle$ should be returned since $R5$ has the highest probability to be ranked first in all possible worlds, and also has the highest probability to be ranked second in all possible worlds. Simultaneously, in [22], given a query Q, the top-k records that have the highest probabilities to satisfy Q in all possible worlds are extracted. For example, on Table 1, if the query is to find a record where a suspect stays for more than 12 minutes, then, the top-2 records are $R3$ and $R5$. Since the ranking is on the probability to satisfy the query, $R1$ is not selected in the top-2 answers though it has a longer duration than both $R3$ and $R5$.

Our contribution. We address an application scenario other than the recent proposals in [26, 22]. Given a probability threshold p, a *probabilistic threshold top-k query* in our model, as elaborated in Example 1, finds the set of records where each takes a probability of at least p to be in the top-k lists in the possible worlds. The new type of queries can find tuples like $R2$ in Example 1 which has a high probability to appear in the top-k lists, but may not be in the top vector of tuples or be the top in some positions.

Challenge 2: How can a probabilistic threshold top-k query be answered efficiently? A naïve method can examine the top-k list in every possible world, derive the top-k probability for each uncertain record, and select the ones passing the threshold. However, it can be very costly on a large data set where the number of possible worlds can be huge.

Can the methods for U-Top$k$ and U-kRanks queries [26] be extended to answer probabilistic threshold top-k queries? The algorithms in [26] scan the tuples in the ranking descending order and materialize all the possible states based on the tuples seen so far. A state is a set of possible worlds which share the top-i tuples and the rest tuples are unknown. The number of states needs to be maintained is exponential in the number of tuples searched. In U-Top$k$ query evaluation, the scan stops when there is a top-k tuple vector whose probability is greater than the upper bound of all other candidates. In U-kRanks query answering, the scan stops when the current answer for each rank is higher than the upper bound of the rest tuples.

The above algorithms cannot be extended to answer probabilistic threshold top-k queries efficiently because both U-Top$k$ queries and U-kRanks queries are "rank sensitive", which require materializing all the possible states. On the other hand, a probabilistic threshold top-k query finds the tuples with a high probability to be in the top-k lists regardless of their exact ranks. Thus, there is no need to maintain as many possible states as in U-Top$k$ query and U-kRanks query answering. More efficient algorithms are feasible for answering probabilistic threshold top-k queries.

Our contribution. We develop efficient methods to tackle the problem. First, we give an exact algorithm which exploits a few interesting techniques to avoid unfolding all possible worlds. It finds the exact top-k probability for each tuple by scanning the sorted list of all tuples only once. The rule-tuple compression technique is proposed to handle generation rules. Several pruning techniques are proposed to further improve the efficiency using the probability threshold. Second, we devise a sampling method which can quickly compute an approximation with quality guarantee to the answer set by drawing a small sample of the uncertain database. Third, we investigate the statistical properties of the top-k probability, and derive a general scan stopping condition for query answering algorithms. Moreover, a Poisson approximation based method is proposed to answer probabilistic threshold top-k queries in linear time.

The rest of the paper is organized as follows. In Section 2, we formulate probabilistic threshold top-k queries. We review the related work in Section 3. We develop an exact algorithm in Section 4, and devise a sampling method in Section 5. The statistical properties of the top-k probability, a general stopping condition of query answering algorithms, and a Poisson approximation based algorithm are discussed in Section 6. A systematic empirical evaluation is reported in Section 7. The paper is concluded in Section 8.

2. PROBABILISTIC THRESHOLD TOP-K QUERIES

We consider uncertain data in the *possible worlds* semantics model [1, 16, 23, 8], which has been extensively adopted by some of the studies on uncertain data processing, such as [26, 4, 21].

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.

Following [26, 4], 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 as $R \cap S$. 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 from the basic probability principles, we have

$$Pr(W) = \prod_{R \in \mathcal{R}_T, |R \cap W|=1} Pr(R \cap W) \prod_{R \in \mathcal{R}_T, R \cap W=\emptyset} (1 - Pr(R)) \tag{1}$$

Apparently, for a possible world W, $Pr(W) > 0$. Moreover, $\sum_{W \in \mathcal{W}} Pr(W) = 1$.

A *top-k query* $Q_{P,f}^k$ 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 and 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. We often denote a top-k query by Q^k or Q when the predicate and the ranking function are unimportant in our discussion.

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^k 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,

$$Pr_{Q,T}^k(t) = \sum_{W \in \mathcal{W}, t \in Q^k(W)} Pr(W) \tag{2}$$

When Q and T are clear from context, we often write $Pr_{Q,T}^k(t)$ 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,

$$Answer(Q, p, T) = \{t | t \in T, Pr_{Q,T}^k(t) \geq p\}.$$

We are interested in how to efficiently compute the answer set for a PT-k query on an uncertain table.

A naïve method to answer a PT-k query is to enumerate all possible worlds and apply the query to each possible world. Then, we can compute the top-k probability of each tuple and select the tuples passing the probability threshold. Unfortunately, the naïve method is inefficient since, as discussed before, there can be a huge number of possible worlds on an uncertain table. In [10], Dalvi and Suciu showed that even the problem of counting all possible worlds with different top-k lists is $\#P$-Complete. Therefore, enumerating all possible worlds is too costly on large uncertain data sets. That motivates our development of efficient algorithms which avoid searching all possible worlds.

3. RELATED WORK

In this section, we review the highly related work briefly from three aspects.

3.1 Top-k Queries on Uncertain Data

In [26], Soliman *et al.* considered ranking queries on uncertain data as we do here. The answer to a U-Top$k$ query is always a top-k tuple list in some valid possible worlds, and the exact positions of the tuples in the list are preserved. A U-kRanks query finds the tuple of the highest probability at each ranking position. The tuples in the results of a U-kRanks query may not form a valid top-k tuple list in a possible world, though a U-kRanks query always returns k tuples. A tuple may appear more than once in the answer set if it has the highest probability values to appear in multiple ranking positions, respectively. Lian and Chen developed the spatial and probabilistic pruning techniques for U-kRanks queries [19].

However, as discussed in Section 1, our study and [26] assume different semantics of ranking queries. In Section 7.1, we will further compare PT-k queries, U-Top$k$ queries and U-kRanks queries using a real data set.

Simultaneously with our study, Yi *et al.* [29] proposed efficient algorithms to answer U-Top$k$ queries and U-kRanks queries. Their algorithm for U-kRanks also uses the Poisson binomial recurrence [17] which is used in our exact algorithm (see Theorem 2 in Section 4.2). However, the two studies address different kinds of top-k queries. Moreover, our unique prefix sharing technique and three pruning techniques can greatly improve the efficiency in answering probabilistic threshold queries. It is worth noting that our algorithm can be used to answer U-kRanks query straight-

forwardly, while their algorithm may not be used to handle PT-k query directly.

In [22], Ré *et al.* considered arbitrary SQL queries and the ranking is on the probability that a tuple satisfies the query instead of using a ranking function. [22] and our study address essentially different queries and applications.

Silberstein *et al.* [24] model each sensor in a sensor network as an uncertain object whose values follow some unknown distribution. Then, a top-k query in the sensor network returns the top-k sensors such that the probability of each sensor whose values are ranked top-k in any timestep is the greatest. A sampling-based method collects all values in the network as a sample at randomly chosen timesteps, and the answer to a top-k query is estimated using the samples.

The probabilistic threshold top-k queries proposed here find all tuples whose probability values of being ranked top-k across all possible worlds pass a probability threshold. The ordering among the top-k tuples does not matter. The number of tuples in the answer set can be smaller than, equal to, or greater than k, depending on the threshold.

Probabilistic threshold top-k queries are substantially different from the existing work on both query type and semantics. Most recently, the concept of probabilistic threshold top-k queries is touched in [15]. However, [15] gives only a simple and preliminary exact algorithm and may not be efficient on large uncertain databases. Simultaneously, Zhang and Chomicki developed the global top-k semantics on uncertain data which returns k tuples having the largest probability in the top-k list, and gave a dynamic programming algorithm [30].

3.2 Probabilistic Data and Query Answering

Modeling and querying uncertain data has been a fast growing research direction [18, 11, 23, 3].

The working model for uncertain data proposed in [23] describes the existence probability of a tuple in an uncertain data set and the constraints (i.e., exclusiveness) on the uncertain tuples, which is essentially the uncertain model adopted in this study.

Cheng *et al.* [6] provided a general classification of probabilistic queries and evaluation algorithms over uncertain data sets. Different from the query answering in traditional data sets, a probabilistic quality estimate was proposed to evaluate the quality of results in probabilistic query answering. Dalvi and Suciu [9] proposed an efficient algorithm to evaluate arbitrary SQL queries on probabilistic databases and rank the results by their probability. Later, they showed in [10] that the complexity of evaluating conjunctive queries on a probabilistic database is either PTIME or #P-complete.

Cheng *et al.* [7] modeled uncertain data as a set of ranges and the associated probability density functions. Probabilistic threshold queries are proposed on the above model, which return the results whose confidence is higher than a user defined threshold. An R-tree based index structure is proposed to help answer such queries. However, only simple queries are discussed in [7], which do not include top-k queries. Tao *et al.* [27] proposed a U-tree index to facilitate probabilistic range queries on uncertain objects represented by multi-dimensional probability density functions. Singh *et al.* [25] extended the inverted index and signature tree to index uncertain categorical data, where the attribute value of each tuple was represented by a set of alternatives.

3.3 Poisson Approximation

The problem of answering probabilistic threshold top-k queries is also related to Bernoulli and Poisson trials and the Poisson binomial distribution in probability and statistics.

A Bernoulli trial is an experiment whose outcome is randomly selected from two possible outcomes, "success" or "failure". Let $X_1, \ldots, X_n$ be n independent Bernoulli random trials. For each trial X_i, the success probability is p_i $(1 \le i \le n)$. The experiment is called Bernoulli trials if the trials are identical and thus the success probability of all the trials are the same. Otherwise, the experiment is called Poisson trials [28].

The sum $X = \sum_{i=1}^{n} X_i$ is the total number of successes in n independent Bernoulli trials. X follows a binomial distribution for identical Bernoulli trials, and a Poisson-binomial distribution for Poisson trials [17]. The exact distribution of $Pr(X = i)$ can be calculated recursively using the Poisson binomial recurrence [17].

Given an uncertain table, a generation rule can be viewed as a Bernoulli trial, if we consider the appearance of a tuple as a "success". The probability of a rule is the success probability of the corresponding trial. The probability of a tuple t to be ranked top-k is the probability that t appears and there are fewer than k successes appear before t.

Some results of Poisson trials can be used in answering probabilistic threshold top-k queries. However, the study of Poisson trials in probability theory does not address the concerns on efficient query answering for large databases. Moreover, multi-tuple generation rules pose new challenges.

In our study, we develop several techniques to process generation rules efficiently. Pruning rules are also proposed to achieve early stop without scanning the whole table, which significantly improves the efficiency in query answering.

4. AN EXACT ALGORITHM

In this section, we first observe a useful property of top-k probability. Then, we propose an exact algorithm to compute top-k probability efficiently.

Hereafter, by default we consider a top-k query $Q_{P,f}^k$ on an uncertain table T. $P(T) = \{t | t \in T \land P(t) = true\}$ 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.

$P(T)$ contains all tuples satisfying the query, as well as the membership probabilities and the generation rules. Removing tuples not in $P(T)$ does not affect the answer to a top-k query. Therefore, $Answer(Q, p, T) = Answer(Q, p, P(T))$. We only need to consider $P(T)$ in answering a top-k query.

4.1 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, 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., $S_t = \{t' | t' \in P(T) \land t' \prec_f t\}$.

THEOREM 1 (DOMINANT SET). *For a tuple* $t \in T$, $Pr_{Q,T}^k(t) = Pr_{Q,S_t}^k(t)$.

TID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
Prob.	0.7	0.2	1	0.3	0.5	0.8	0.1	0.8	0.1

Table 4: The ranked list of uncertain tuples.

Proof sketch. The theorem follows with Equations 1 and 2, and the assumption that each tuple is involved in only one generation rule. ∎

Using the dominant set property, the PT-k query answering algorithm developed in the rest of this section 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 involving multiple tuples are handled by the rule-tuple compression technique. The probability threshold is used to prune tuples whose top-k probability values fail the threshold.

4.2 The Basic Case

In this subsection, we 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,

$$Pr(t_i, j) = Pr(t_i)Pr(S_{t_i}, j - 1) \qquad (3)$$

Apparently, the top-k probability of t_i is given by

$$Pr^k(t_i) = \sum_{j=1}^{k} Pr(t_i, j) = Pr(t_i) \sum_{j=1}^{k} Pr(S_{t_i}, j - 1) \qquad (4)$$

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

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

Proof sketch. In the basic case, all tuples are independent. The theorem follows with the basic probability principles. It is also called the Poisson binomial recurrence in [17]. ∎

Theorem 2 can be used to compute the top-k probability values efficiently, as illustrated in the following example.

EXAMPLE 2 (THE BASIC CASE). *Consider a top-k query $Q^3(true, f)$. Suppose f ranks the tuples in an uncertain table T into a list L as in Table 4. Apparently, $S_{t_i} = L[1..i - 1]$. In this example, we assume all tuples are independent.*

To compute the top-3 probability of each tuple, we first initialize $Pr(\emptyset, 0) = 1$, $Pr(\emptyset, 1) = 0$ and $Pr(\emptyset, 2) = 0$. Then, we scan the ranked list.

For t_j ($1 \leq j \leq 3$), $Pr^3(t_j) = Pr(t_j)$. Thus, $Pr^3(t_1) = 0.7$, $Pr^3(t_2) = 0.2$, and $Pr^3(t_3) = 1$.

To compute $Pr^3(t_4)$, we first compute $Pr(S_{t_3}, 0) = 0$, $Pr(S_{t_3}, 1) = 0.24$, and $Pr(S_{t_3}, 2) = 0.62$ using Theorem 2.

Then, according to Equation 4, we have

$$Pr^3(t_4) = Pr(t_4)(Pr(S_{t_3}, 0) + Pr(S_{t_3}, 1) + Pr(S_{t_3}, 2))$$
$$= 0.258.$$
∎

4.3 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.

Suppose tuples $t_1, \ldots, t_n \in P(T)$ are in the ranking order. For a tuple t_i, two situations due to the presence of multi-tuple generation rules complicate the computation.

First, there may be a rule R such that some tuples involved in R are ranked higher than t_i. Those tuples cannot appear together in a possible world. To compute subset probability $Pr(S_{t_i}, j)$ correctly, we need to make sure that rule R is respected.

Second, t_i itself may be involved in a generation rule R, and cannot appear together in a possible world with any other tuples involved in the same rule.

In both cases, some tuples in S_{t_i} are dependent and thus Theorem 2 cannot be applied directly. Can dependent tuples in S_{t_i} be transformed to independent ones so that Theorem 2 can still be used?

4.3.1 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.
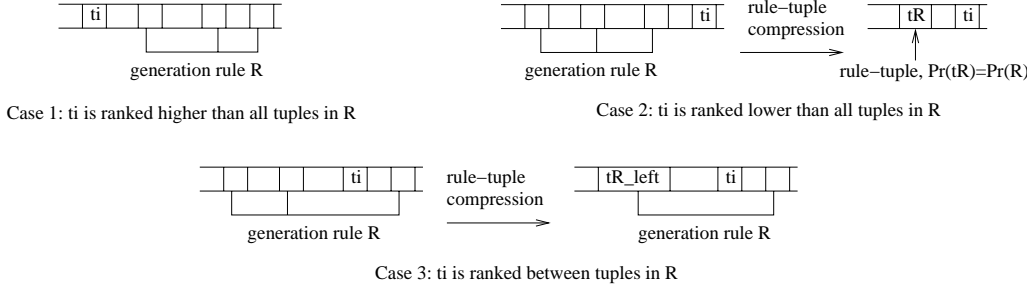
Case 1: ti is ranked higher than all tuples in R

Case 2: ti is ranked lower than all tuples in R

rule–tuple compression

rule–tuple, Pr(tR)=Pr(R)

Case 3: ti is ranked between tuples in R

rule–tuple compression

generation rule R

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

Second, if $t_i \in R$, 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 $|R| > 1$, $Pr_{Q,T}^k(t) = Pr_{Q,T'}^k(t)$ 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 the 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 resulting 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, $Pr_{Q,T}^k(t) = Pr_{Q,T(t)\cup\{t\}}^k(t)$. We can apply Theorem 2 to calculate $Pr^k(t)$ by scanning $T(t)$ once.

EXAMPLE 3 (RULE-TUPLE COMPRESSION). *Consider the uncertain table and the top-k query in Example 2 again. Now, suppose we have two multi-tuple generation rules: $R_1 = t_2 \oplus t_4 \oplus t_9$ and $R_2 = t_5 \oplus t_7$. Let us consider how to compute $Pr^3(t_6)$ and $Pr^3(t_7)$.*

Tuple t_6 is ranked between tuples in R_1, but $t_6 \notin R_1$. The first subcase of Case 3 should be applied. Thus, we compress $R_{1left} = \{t_2, t_4\}$ into a rule-tuple $t_{2,4}$ with membership probability $Pr(t_{2,4}) = Pr(t_2) + Pr(t_4) = 0.5$. Similarly, t_6 is also ranked between tuples in R_2 and $t_6 \notin R_2$, but $R_{2left} = \{t_5\}$. The compression does not remove any tuple. After the compression, we compute $Pr^3(t_6)$ using $T(t_6) = \{t_1, t_{2,4}, t_3, t_5\}$. The tuples in $T(t_6) \cup \{t_6\}$ are independent. We apply Theorem 2 as illustrated in Example 2 to obtain $Pr^3(t_6) = 0.32$.

Since $t_7 \in R_2$, the tuples in R_2 except for t_7 itself should be removed. Thus, we have $T(t_7) = \{t_1, t_{2,4}, t_3, t_6\}$. We get $Pr^3(t_6) = 0.025$ on $T(t_7) \cup \{t_7\}$. ∎

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 $O(kn^2)$ time where n is the number of tuples in the uncertain table.

4.3.2 Scan Reduction by Prefix Sharing

EXAMPLE 4 (EXAMPLE 3 REVISITED). *In Example 3, t_6 and t_7 are neighbors in the sorted list L. However, since t_5 is in $T(t_6)$ but not in $T(t_7)$, the subset probabilities $Pr(T(t_6), j)$ $(0 \leq j \leq 2)$ cannot be used in calculating $Pr^k(t_7)$.*

However, $T(t_6)$ and $T(t_7)$ share the subset $\{t_1, t_{2,4}, t_3\}$. The subset probability values $Pr(S_{t_3}, j)$ computed using $T(t_6)$ can be reused for $T(t_7)$. In other words, by sharing the prefix between the two compressed dominant sets, we can reduce the cost of computing subset probability, which is the major cost in top-k probability computation. ∎

Equation 4 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.

How can we achieve good sharing? 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 4.3.1), 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 4.3.1).

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 contrary, 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.

EXAMPLE 5 (REORDERING). *Figure 2 shows a list of tuples in an uncertain table T in the ranking order with respect to a top-k query. There are two multi-tuple rules $R_1 : t_1 \oplus t_2 \oplus t_8 \oplus t_{11}$ and $R_2 : t_4 \oplus t_5 \oplus t_{10}$. The compressed dominant sets of tuples in the orders made by the aggressive method and the lazy method are also listed in the figure.*
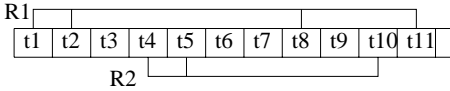
A sorted list of tuples in P(T)

tuple	Aggressive reordering		Lazy reordering	
	Prefix	Cost	Prefix	Cost
t_1	$\emptyset$	0	$\emptyset$	0
t_2	$\emptyset$	0	$\emptyset$	0
t_3	$t_{1,2}$	1	$t_{1,2}$	1
t_4	$t_3 t_{1,2}$	2	$t_{1,2} t_3$	1
t_5	$t_3 t_{1,2}$	0	$t_{1,2} t_3$	0
t_6	$t_3 t_{4,5} t_{1,2}$	2	$t_{1,2} t_3 t_{4,5}$	1
t_7	$t_3 t_6 t_{4,5} t_{1,2}$	3	$t_{1,2} t_3 t_{4,5} t_6$	1
t_8	$t_3 t_6 t_7 t_{4,5}$	2	$t_3 t_6 t_7 t_{4,5}$	4
t_9	$t_3 t_6 t_7 t_{1,2,8} t_{4,5}$	2	$t_3 t_6 t_7 t_{4,5} t_{1,2,8}$	1
t_{10}	$t_3 t_6 t_7 t_9 t_{1,2,8}$	2	$t_3 t_6 t_7 t_9 t_{1,2,8}$	2
t_{11}	$t_3 t_6 t_7 t_9 t_{4,5,10}$	1	$t_3 t_6 t_7 t_9 t_{4,5,10}$	1
	Total cost: 15		Total cost: 12	

Figure 2: Comparison of two reordering methods.

The two methods order the compressed dominant sets in the same way for t_1, t_2 and t_3.

For t_4, the aggressive method orders t_3, an independent tuple, before $t_{1,2}$, the rule-tuple for rule R_1 which is open with respect to t_4. The subset probability values computed in $T(t_3)$ cannot be reused. The lazy method reuses the prefix $t_{1,2}$ from $T(t_3)$, and appends t_3 after $t_{1,2}$. All subset probability values computed in $T(t_3)$ can be reused.

Contrarily, the lazy method reorders only tuples not in the maximum common prefix. For example, for t_8, $t_{1,2}$ is not in $T(t_8)$ and thus no subset probability values computed in $T(t_7)$ by the lazy method can be reused. Then, the lazy method reorders the tuples in $T(t_8)$ by putting the independent tuples before the rule-tuples of open rules. ∎

For two consecutive tuples t_i and t_{i+1} in the sorted list L of all tuples in $P(T)$, let $L(t_i)$ and $L(t_{i+1})$ be the sorted lists of the tuples in $T(t_i)$ and $T(t_{i+1})$, respectively, given by a reordering method such as the aggressive method and the lazy method. Let $Prefix(L(t_i), L(t_{i+1}))$ be the longest common prefix between $L(t_i)$ and $L(t_{i+1})$. The total number of subset probability values needed to be calculated is

$$Cost = \sum_{i=1}^{n-1}(|L(t_{i+1})| - |Prefix(L(t_i), L(t_{i+1}))|) \quad (5)$$

In Example 5, $Cost_{aggressive} = 15$ and $Cost_{lazy} = 12$. We can show that the lazy method is never worse than the aggressive method. Limited by space, we omit the details here. In our experiments, the lazy method shows a great advantage against the aggressive method.

4.4 Pruning Techniques

So far, we implicitly have a requirement: all tuples satisfying the predicate in the PT-k query, i.e., $P(T)$, should be sorted in the ranking order. However, a PT-k query is interested in only those tuples whose top-k probabilities pass the probability threshold. Can we avoid retrieving or checking all tuples satisfying the query predicates?

Some existing methods such as the well known TA algorithm [12] can retrieve in batch tuples satisfying the pred-

icate 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 with probability threshold p. We give four pruning rules: Theorems 3 and 4 can avoid checking some tuples that cannot satisfy the probability threshold, and Theorems 5 and 6 specify stopping conditions. The tuple retrieval method (e.g., an adaption of the TA algorithm [12]) 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.

A tuple involved in a multi-tuple rule may be pruned using the other tuples in the same rule.

THEOREM 4 (PRUNING BY TUPLES IN THE SAME RULE). For tuples t and t' 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 is based on the observation that the sum of the top-k probability values of all tuples is exactly k. That is $\sum_{t \in T} Pr^k(t) = k$.

THEOREM 5 (PRUNING BY TOTAL TOP-k PROBABILITY). Let A be a set of tuples whose top-k probability values have been computed. If $\sum_{t \in A} Pr^k(t) > k - p$, then for every tuple $t' \notin A$, $Pr^k(t') < p$. ∎

Moreover, we have a tight stopping condition as follows.

THEOREM 6 (A TIGHT STOPPING CONDITION). Let $t_1, \ldots, t_m, \ldots, t_n$ be the tuples in the ranking order. Assume $L = t_1, \ldots, t_m$ are read. Let LR be the set of open rules with respect to t_{m+1}. For any tuple t_i $(i > m)$,

1. if t_i is not in any rule in LR, the top-k probability of t_i $Pr^k(t_i) \leq \sum_{j=0}^{k-1} Pr(L, j)$.

2. if t_i is in a rule in LR, the top-k probability of t_i $Pr^k(t_i) \leq \max_{R \in LR}(1 - Pr(t_{R_{left}})) \sum_{j=0}^{k-1} Pr(L - t_{R_{left}}, j)$.

Input: an uncertain table T, a set of generation rules $\mathcal{R}_T$, a top-k query $Q_{P,f}^k$, 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` $t_i \in P(T)$ `do`
2:      compute $T(t_i)$ by rule-tuple compression;
3:      compute subset probability values and $Pr^k(t_i)$;
4:      `if` $Pr^k(t_i) \geq p$ `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`

Figure 3: The exact algorithm.

Proof. For item (1), consider the compressed dominant set $T(t_i)$ of t_i. $L \subseteq T(t_i)$. Therefore,

$$Pr^k(t_i) = Pr(t_i) \sum_{j=0}^{k-1} Pr(T(t_i), j) \leq \sum_{j=0}^{k-1} Pr(L, j).$$

The equality holds if tuple t_{m+1} is independent with membership probability 1.

For item (2), suppose t_i is involved in an open rule $R \in LR$. $Pr(t_i) \leq 1 - Pr(t_{R_{left}})$. Moreover, for the compressed dominant set $T(t_i)$ of t_i, $(L - t_{R_{left}}) \subseteq T(t_i)$. Therefore,

$$
\begin{aligned}
Pr^k(t_i) =\ & Pr(t_i) \sum_{j=0}^{k-1} Pr(T(t_i), j) \\
\leq\ & (1 - Pr(t_{R_{left}})) \sum_{j=0}^{k-1} Pr(L - t_{R_{left}}, j)
\end{aligned}
$$

The equality holds when tuple t_{m+1} is involved in rule R' with membership probability $1 - Pr(t_{R'_{left}})$, where

$$R' = \arg \max_{R \in LR} (1 - Pr(t_{R_{left}})) \sum_{j=0}^{k-1} Pr(L - t_{R_{left}}, j).$$

∎

Theorem 6 provides the upper bounds for tuples that have not been seen yet. If the upper bounds are both lower than the probability threshold p, then the unseen tuples do not need to be checked.

In summary, the exact algorithm for PT-k query answering is shown in Figure 3. We analyze the complexity of the algorithm 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 four 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.

5. A SAMPLING METHOD

One may trade off the accuracy of answers against the efficiency. In this section, we present a simple yet effective sampling method.

For a tuple t, let X_t be a random variable as an indicator to the event that t is ranked top-k in possible worlds. $X_t = 1$ if t is ranked in the top-k list, and $X_t = 0$ otherwise. Apparently, the top-k probability of t is the expectation of X_t, i.e., $Pr^k(t) = E[X_t]$. Our objective is to draw a set of samples S of possible worlds, and compute the mean of X_t in S, namely $E_S[X_t]$, as the approximation of $E[X_t]$.

We use uniform sampling with replacement. For table $T = \{t_1, \ldots, t_n\}$ and the set of generation rules $\mathcal{R}_T$, a sample unit (i.e., an observation) is a possible world. We generate the sample units under the distribution of T: to pick a sample unit s, we scan T once. An independent tuple t_i is included in s with probability $Pr(t_i)$. For a multi-tuple generation rule $R : t_{r_1} \oplus \cdots \oplus t_{r_m}$, s takes a probability of $Pr(R)$ to include one tuple involved in R. If s takes a tuple in R, then tuple t_{r_l} $(1 \leq l \leq m)$ is chosen with probability $\frac{Pr(t_{r_l})}{Pr(R)}$. s can contain at most 1 tuple from any generation rule.

Once a sample unit s is generated, we compute the top-k tuples in s. For each tuple t in the top-k list, $X_t = 1$. The indicators for other tuples are set to 0.

The above sample generation process can be repeated so that a sample S is obtained. Then, $E_S[X_t]$ can be used to approximate $E[X_t]$. When the sample size is large enough, the approximation quality can be guaranteed following from the well known Chernoff-Hoeffding bound [2].

THEOREM 7 (SAMPLE SIZE). *For any δ ($0 < \delta < 1$), ϵ ($\epsilon > 0$), and a sample S of possible worlds, if*

$$|S| \geq \frac{3 \ln \frac{2}{\delta}}{\epsilon^2}$$

then for any tuple t,

$$Pr\{|E_S[X_t] - E[X_t]| > \epsilon E[X_t]\} \leq \delta.$$

∎

We can implement the sampling method efficiently using the following two techniques, as verified by our experiments.

First, we can sort all tuples in $P(T)$ in the ranking order into a sorted list L. The first k tuples in a sample unit are the top-k answers in the unit. Thus, when generating a sample unit, instead of scanning the whole table T, we only need to scan L from the beginning and generate the tuples in the sample as described before. However, once the sample unit has k tuples, the generation of this unit can stop. In this way, we reduce the cost of generating sample units without losing the quality of the sample. For example, when all tuples are independent, if the average membership probability is μ, the expected number of tuples we need to scan to generate a sample unit is $\lceil \frac{k}{\mu} \rceil$, which can be much smaller than $|P(T)|$ when $k \ll |P(T)|$.

Second, in practice, the actual approximation quality may converge well before the sample size reaches the bound given in Theorem 7. Thus, *progressive sampling* can be adopted: we generate sample units one by one and compute the estimated top-k probability of tuples after each unit is drawn. For given parameters $d > 0$ and $\phi > 0$, the sampling process stops if in the last d sample units the change of the estimated X_t for any tuple t is smaller than ϕ.

6. A POISSON APPROXIMATION BASED METHOD

In this section, we further analyze the properties of top-k probability from the statistics aspect, and derive a general stopping condition for query answering algorithms which depends on parameter k and threshold p only and is independent from data set size. We also devise an approximation method based on the Poisson approximation.

6.1 Distribution of Top-k Probability

Let $X_1, \ldots, X_n$ be a set of independent random variables, such that $Pr(X_i = 1) = p_i$ and $Pr(X_i = 0) = 1 - p_i$ $(1 \leq i \leq n)$. Let $X = \sum_{i=1}^{n} X_i$. Then, $E[X] = \sum_{i=1}^{n} p_i$. If all p_i's are identical, $X_1, \ldots, X_n$ are called *Bernoulli trials*, and X follows a *binomial distribution*; otherwise, $X_1, \ldots, X_n$ are called *Poisson trials*, and X follows a *Poisson binomial distribution*.

For a tuple $t \in P(T)$, according to Equation 4, the top-k probability of t is

$$Pr^k(t) = Pr(t) \sum_{j=1}^{k} Pr(T(t), j-1),$$

where $Pr(t)$ is the membership probability of t, $T(t)$ is the compressed dominant set of t. Moreover, the probability that fewer than k tuples appear in $T(t)$ is

$$\sum_{j=1}^{k} Pr(T(t), j-1).$$

If there is any tuple or rule-tuple with probability 1, we can remove the tuple from $T(t)$, and compute the top-$(k-1)$ probability of t. Thus, we can assume that the membership probability of any tuple or rule-tuple in $T(t)$ is smaller than 1.

To compute $Pr^k(t)$, we construct a set of Poisson trials corresponding to $T(t)$ as follows. For each independent tuple $t' \in T(t)$, we construct a random trial $X_{t'}$ whose success probability $Pr(X_{t'} = 1) = Pr(t')$. For each multi-tuple rule R $(R \cap T(t) \neq \emptyset)$, we combine the tuples in $R \cap T(t)$ into a rule-tuple t_R such that $Pr(t_R) = \sum_{t' \in R \cap T(t)} Pr(t')$, and construct a random trial X_{t_R} whose success probability $Pr(X_{t_R} = 1) = Pr(t_R)$.

Let $X_1, \ldots, X_n$ be the resulting trials. Since the independent tuples and rule-tuples in $T(t)$ are independent and their membership probabilities vary in general, $X_1, \ldots, X_n$ are independent and have unequal success probability values. They are Poisson trials. Let $X = \sum_{i=1}^{n} X_i$, then $Pr(T(t), j) = Pr(X = j)$ $(0 \leq j \leq n)$ where $Pr(X = j)$ is the probability of j successes. Thus, the probability that t is ranked the k-th is $Pr(t, k) = Pr(t)Pr(X = k-1)$. Moreover, the top-$k$ probability of t is given by

$$Pr^k(t) = Pr(t)Pr(X < k)$$

X follows the Poisson binomial distribution. Therefore, $Pr(t, k)$ also follows the Poisson binomial distribution, and $Pr^k(t)$ follows the cumulative distribution function of $Pr(t, k)$.

In a Poisson binomial distribution X, The probability density of X is unimodal (i.e., first increasing then decreasing), and attains its maximum at $\mu = E[X]$ [14]. Therefore, when the query parameter k varies from 1 to $|T(t)| + 1$, $Pr(t, k)$ follows the similar trend.

COROLLARY 3 (DISTRIBUTION OF POSITION PROBABILITY). *For a tuple $t \in P(T)$,*

1. $Pr(t, k) = 0$ *for* $k > |T(t)| + 1$;

2. $Pr(t, k) < Pr(t, k+1)$ *for* $k \leq \mu - 1$, *and* $Pr(t, k) > Pr(t, k+1)$ *for* $k \geq \mu$; *and*

3.

$$\arg \max_{j=1}^{|T(t)|+1} Pr(t, j) = \mu + 1,$$

where $\mu = \sum_{t' \in T(t)} Pr(t')$. ∎

6.2 A General Stopping Condition

Corollary 3 shows that, given a tuple t and its compressed dominant set $T(t)$, the most possible ranks of t are around $\mu + 1$. In other words, if $k \ll \mu + 1$, then the top-k probability of t is small. Now, let us use this property to derive a general stopping condition for query answering algorithms progressively reading tuples in the ranking order. That is, once the stopping condition holds, all unread tuples cannot satisfy the query and can be pruned. The stopping condition is independent from the number of tuples in the data set, and dependent on only the query parameter k and the probability threshold p.

THEOREM 8 (A GENERAL STOPPING CONDITION). *Given a top-k query $Q^k(P, f)$ and probability threshold p, for a tuple $t \in P(T)$, let $\mu = \sum_{t' \in T(t)} Pr(t')$. Then, $Pr^k(t) < p$ if*

$$\mu \geq k + \ln \frac{1}{p} + \sqrt{\ln^2 \frac{1}{p} + 2k \ln \frac{1}{p}}.$$

To prove Theorem 8, we need Theorem 4.2 in [20].

LEMMA 1 (CHERNOFF BOUND OF POISSON TRIALS [20]). *Let $X_1, \ldots, X_n$ be independent Poisson trials such that, for $1 \leq i \leq n$, $Pr[X_i = 1] = p_i$, where $0 < p_i < 1$. Then, for $X = \sum_{i=1}^{n} X_i$, $\mu = E[X] = \sum_{i=1}^{n} p_i$, and $0 < \epsilon \leq 1$, we have*

$$Pr[X < (1 - \epsilon)\mu] < e^{-\frac{\mu\epsilon^2}{2}}.$$

Proof of Theorem 8. As discussed in Section 6.1, we can construct a set of Poisson trials corresponding to the tuples in $T(t)$ such that, for each tuple or rule-tuple $t' \in T(t)$, there is a corresponding trial whose success probability is the same as $Pr(t')$. Moreover,

$$\sum_{j=0}^{k-1} Pr(T(t), j) = Pr[X < k].$$

For $0 < \epsilon \leq 1$, inequality $Pr[X < k] \leq Pr[X < (1 - \epsilon)\mu]$ holds when

$$k \leq (1 - \epsilon)\mu \qquad (6)$$

Using Lemma 1, we have

$$Pr[X < k] \leq Pr[X < (1 - \epsilon)\mu] < e^{-\frac{\mu\epsilon^2}{2}}$$

$Pr[X < k] < p$ holds if

$$e^{-\frac{\mu\epsilon^2}{2}} \leq p \qquad (7)$$

Combining Inequality 6 and 7, we get

$$2\ln\frac{1}{p} \le \mu(1 - \frac{k}{\mu})^2$$

The inequality in Theorem 8 is the solution to the above inequality. ∎

Since $\mu = \sum_{t' \in T(t)} Pr(t')$, the μ value is monotonically increasing if tuples are sorted in the ranking order. Using Theorem 8 an algorithm can stop and avoid retrieving further tuples in the rear of the sorted list if the μ value of the current tuple satisfies the condition in Theorem 8.

The value of parameter k is typically set to much smaller than the number of tuples in the whole data set. Moreover, since a user is interested in the tuples with a high probability to be ranked in top-k, the probability threshold p is often not too small. Consequently, μ is often a small value. For example, if $k = 100$, $p = 0.3$, then the stopping condition is $\mu \ge 117$.

In the experiments, we show in Figure 4 that the exact algorithm and the sampling algorithm stop close to the general stopping condition. The results verify the tightness of the stopping condition.

6.3 A Poisson Approximation Based Method

When the success probability is small and the number of Poisson trials is large, Poisson binomial distribution can be approximated well by Poisson distribution [13].

For a set of Poisson trials $X_1, \ldots, X_n$ such that $Pr(X_i = 1) = p_i$, let $X = \sum_{i=1}^{n} X_i$. X follows a Poisson binomial distribution. Let $\mu = E[X] = \sum_{i=0}^{n} p_i$. The probability of $X = k$ can be approximated by $Pr(X = k) \approx f(k; \mu) = \frac{\mu^k}{k!}e^{-\mu}$, where $f(k; \mu)$ is the Poisson probability mass function. Thus, the probability of $X < k$ can be approximated by

$$Pr(X < k) \approx F(k; \mu) = \frac{\Gamma(\lfloor k+1 \rfloor, \mu)}{\lfloor k \rfloor!},$$

where $F(k; \mu)$ is the cumulative distribution function corresponding to $f(k; \mu)$, and $\Gamma(x, y) = \int_y^\infty t^{x-1}e^{-t}dt$ is the upper incomplete gamma function. Theoretically, Le Cam [5] showed that the quality of the approximation has the following upper bound.

$$\sup_{0 \le l \le n} \left| \sum_{k=0}^{l} Pr(X = k) - \sum_{k=0}^{l} f(k; \mu) \right| \le 9 \max_i \{p_i\}$$

The above upper bound depends on only the maximum success probability in the Poisson trials. In the worst case where $\max_i\{p_i\} = 1$, the error bound is very loose. However, our experimental results (Figure 7) show that the Poisson approximation method achieves very good approximation quality in practice.

To use Poisson approximation to evaluate a top-k query $Q^k(P, f)$, we scan the tuples in $P(T)$ in the ranking order. The sum of membership probabilities of the scanned tuples is maintained in μ. Moreover, for each generation rule R, let R_{left} be the set of tuples in R that are already scanned. Correspondingly, let μ_R be the sum of membership probabilities of the tuples in R_{left}.

When a tuple t is scanned, if t is an independent tuple, then the top-k probability of t can be estimated using

$$Pr(t)F(k-1; \mu) = Pr(t)\frac{\Gamma(k, \mu)}{(k-1)!}.$$

If t belongs to a generation rule R, then the top-k probability of t can be estimated by

$$Pr(t)F(k-1; \mu') = Pr(t)\frac{\Gamma(k, \mu')}{(k-1)!},$$

where $\mu' = \mu - \mu_R$. t is output if the estimated probability $Pr^k(t)$ passes the probability threshold p. The scan stops when the general stopping condition in Theorem 8 is satisfied.

In the Poisson approximation based method, we need to maintain the running μ and μ_R for each open rule R. Thus, the space requirement of the Poisson approximation based method is $O(|\mathcal{R}_T| + 1)$, where $\mathcal{R}_T$ is the set of generation rules. The time complexity is $O(n')$, where n' is the number of tuples read before the general stopping condition is satisfied, which depends on parameter k, probability threshold p and the probability distribution of the tuples and is independent from the size of the uncertain table.

7. EXPERIMENTAL RESULTS

We conducted a systematic empirical study using a real data set and some synthetic data sets on a PC computer with a 3.0 GHz Pentium 4 CPU, 1.0 GB main memory, and a 160 GB hard disk, running the Microsoft Windows XP Professional Edition operating system. Our algorithms were implemented in Microsoft Visual C++ V6.0. The executable code of our algorithms and the source code of the data generator are available at `http://www.cs.sfu.ca/~jpei/software.htm`.

7.1 Results on IIP Iceberg Database

We use the International Ice Patrol (IIP) Iceberg Sightings Database (`http://nsidc.org/data/g00807.html`) to examine the effectiveness of top-k queries on uncertain data in real applications. The International Ice Patrol (IIP) Iceberg Sightings Database collects information on iceberg activities in the North Atlantic. The mission is to monitor iceberg danger near the Grand Banks of Newfoundland by sighting icebergs (primarily through airborne Coast Guard reconnaissance missions and information from radar and satellites), plotting and predicting iceberg drift, and broadcasting all known ice to prevent icebergs threatening.

In the database, each sighting record contains the sighting date, sighting location (latitude and longitude), number of days drifted, etc. Among them, the number of days drifted is derived from the computational model of the IIP, which is crucial in determining the status of icebergs. It is interesting to find the icebergs drifting for a long period.

However, each sighting record in the database is associated with a confidence level according to the source of sighting, including: R/V (radar and visual), VIS (visual only), RAD(radar only), SAT-L(low earth orbit satellite), SAT-M (medium earth orbit satellite) and SAT-H (high earth orbit satellite). In order to quantify the confidence, we assign confidence values 0.8, 0.7, 0.6, 0.5, 0.4 and 0.3 to the above six confidence levels, respectively.

Moreover, generation rules are defined in the following way. For the sightings with the same time stamp, if the sighting locations are very close – differences in latitude and longitude are both smaller than 0.01 (i.e.,0.02 miles), they are considered referring to the same iceberg, and only one of the sightings is correct. All tuples involved in such a sighting form a multi-tuple rule. For a rule $R : t_{r_1} \oplus \cdots \oplus$

Rank	1	2	3	4	5	6	7	8	9	10
Tuple	$R1$	$R2$	$R3$	$R5$	$R6$	$R9$	$R9$	$R11$	$R11$	$R18$
Probability at this rank	0.8	0.64	0.512	0.348	0.328	0.258	0.224	0.234	0.158	0.163

Table 5: The answers to the *U-kRanks* query.

Tuple	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$	$R7$	$R8$	$R9$	$R10$	$R11$	$R14$	$R18$
Drifted days	435.8	341.7	335.7	323.9	284.7	266.8	259.5	240.4	233.6	233.3	232.6	230.9	229.3
Membership prob.	0.8	0.8	0.8	0.6	0.8	0.8	0.4	0.15	0.8	0.7	0.8	0.6	0.8
Top-10 prob.	0.8	0.8	0.8	0.6	0.8	0.8	0.4	0.15	0.8	0.7	0.79	0.52	0.359

Table 6: Some tuples in the IIP Iceberg Sightings Database 2006.

t_{r_m}, $Pr(R)$ is set to the maximum confidence among the membership probability values of tuples in the rule. Then, the membership probability of a tuple is adjusted to

$$Pr(t_{r_l}) = \frac{conf(t_{r_l})}{\sum_{1 \leq i \leq m} conf(t_{r_i})} Pr(R) \;\; (1 \leq l \leq m)$$

, where $conf(t_{r_l})$ is the confidence of t_{r_l}. After the above preprocessing, the database contains $4,231$ tuples and 825 multi-tuple rules. The number of tuples involved in a rule varies from 2 to 10. We name the tuples in the number of drifted days descending order. For example, tuple $R1$ has the largest value and $R2$ has the second largest value on the attribute.

We applied *PT-k query*, *U-TopK query* and *U-KRanks query* on the database by setting $k = 10$ and $p = 0.5$. The ranking order is the number of drifted days descending order. The PT-k query returns a set of 10 objects $\{R1, R2, R3, R4, R5, R6, R9, R10, R11, R14\}$. The U-Top$k$ query returns a vector $\langle R1, R2, R3, R4, R5, R6, R7, R9, R10, R11 \rangle$ with probability 0.0299. The U-kRanks query returns 10 tuples as shown in Table 5. The probability values of the tuples at the corresponding ranks are also shown in the table. To understand the answers, in Table 6 we also list the membership probability values and the top-10 probability values of some tuples including the ones returned by the PT-k, U-Top$k$, and U-kRanks queries.

All tuples with top-10 probability at least 0.5 are returned by the PT-k query. The top-10 probability of $R14$ is higher than $R7$, but $R7$ is included in the answer of the U-Top$k$ query and $R14$ is missing. Moreover, the presence probability of the top-10 list returned by the U-Top$k$ query is quite low. Although it is the most probable top-10 tuple list, the low presence probability limits its usefulness and interestingness.

$R10$ and $R14$, whose top-10 probability values are high, are missing in the results of the U-kRanks query, since none of them is the most probable at any rank. Nevertheless, $R18$ is returned by the U-kRanks query at the 10-th position, though its top-10 probability is much lower than $R10$ and $R14$. Moreover, $R9$ and $R11$ each occupies two positions in the answer of the U-kRanks query.

The results clearly show that the *PT-k query* captures some important tuples missed by the *U-TopK query* and the *U-KRanks query*. This example elaborates the differences among the three types of top-k queries on uncertain data.

7.2 Results on Synthetic Data Sets

To evaluate the query answering quality and the scalability of our algorithms, we generate various synthetic data sets. The membership probability values of independent tuples and multi-tuple generation rules follow the normal distribution $N(\mu_{P_t}, \sigma_{P_t})$ and $N(\mu_{P_R}, \sigma_{P_R})$, respectively. The rule complexity, i.e., the number of tuples involved in a rule, follows the normal distribution $N(\mu_{|R|}, \sigma_{|R|})$.

By default, a synthetic data set contains $20,000$ tuples and $2,000$ multi-tuple generation rules. The number of tuples involved in each multi-tuple generation rule follows the normal distribution $N(5, 2)$. The probability values of independent tuples and multi-tuple generation rules follow the normal distribution $N(0.5, 0.2)$ and $N(0.7, 0.2)$, respectively. We test the probabilistic threshold top-k queries with $k = 200$ and $p = 0.3$.

We assume that all tuples in the synthetic data sets satisfy the predicates in the top-k queries. Since ranking queries are extensively supported by modern database management systems, we treat the generation of a ranked list of uncertain tuples as a black box, and test our algorithms on top of the ranked list.

We compare the exact algorithm, the sampling method, and the Poisson approximation based method. For the exact algorithm, we compare three versions: RC (rule-tuple compression only), RC+AR (RC with aggressive reordering), and RC+LR (RC with lazy reordering). The sampling method uses the two improvements described in Section 5.

We test the number of tuples scanned by the methods (Figure 4). We count the number of distinct tuples read by the exact algorithm and the *sample length* as the average number of tuples read by the sampling algorithm to generate a sample unit. For reference, we also plot the number of tuples in the answer set, i.e., the tuples satisfying the probabilistic threshold top-k queries, and the number of tuples computed by the general stopping condition discussed in Section 6.

In Figure 4(a), when the expected membership probability is high, the tuples at the beginning of the ranked list likely appear, which reduce the probabilities of the lower ranked tuples to be ranked in the top-k lists in possible worlds. If the membership probability of each tuple is very close to 1, then very likely we can prune all the tuples after the first k tuples are scanned. In contrary, if the expectation of the membership probability is low, then more tuples have a chance to be in the top-k lists of some possible worlds. Consequently, the methods have to check more tuples.

In Figure 4(b), when the rule complexity increases, more tuples are involved in a rule. The average membership probability of those tuples decreases, and thus more tuples need to be scanned to answer the query. In Figure 4(c), both the scan depth and the answer set size increase linearly when k
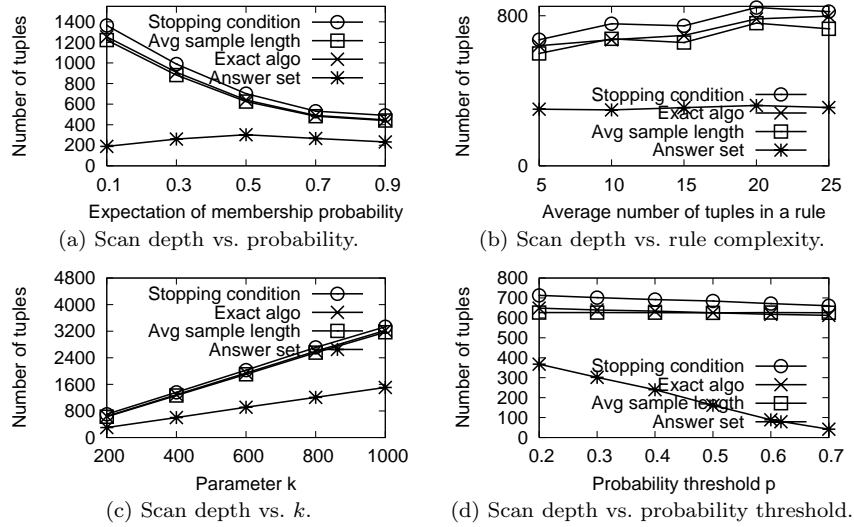
(a) Scan depth vs. probability.

(b) Scan depth vs. rule complexity.

(c) Scan depth vs. k.

(d) Scan depth vs. probability threshold.

Figure 4: Scan depth (each test data set contains $20,000$ **tuples and** $2,000$ **generation rules).**

(a) Runtime vs. probability.

(b) Runtime vs. rule complexity.

(c) Runtime vs. k.

(d) Runtime vs. probability threshold.
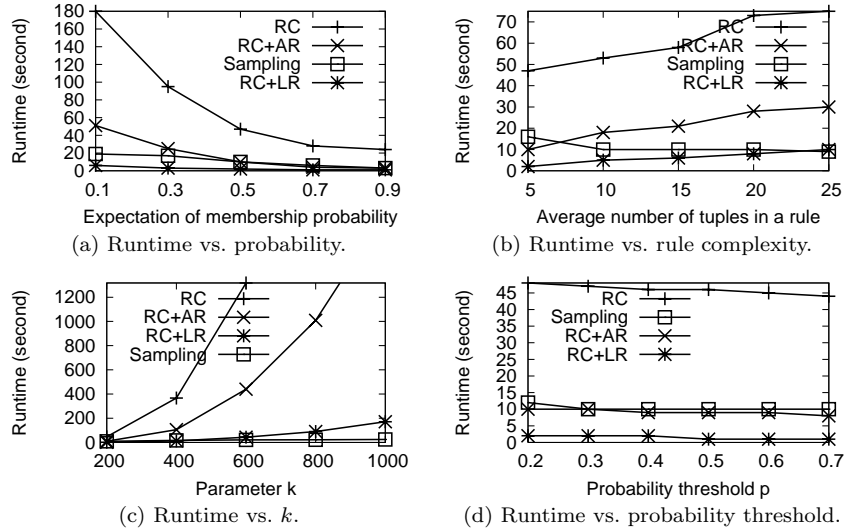
Figure 5: Efficiency (same settings as in Figure 4).

increases, which is intuitive. In Figure 4(d), the size of the answer set decreases linearly as the probability threshold p increases. However, the number of tuples scanned decreases much slower. As discussed in Section 4.4, a tuple t failing the probability threshold still has to be retrieved if some tuples ranked lower than t may satisfy the threshold.

Figure 4 verifies the effectiveness of the pruning techniques discussed in Section 4.4. With the pruning techniques, the exact algorithm only accesses a small portion of the tuples in the data set. Interestingly, the average sample length is close to the number of tuples scanned in the exact algorithm, which verifies the effectiveness of our sampling techniques. Moreover, the exact algorithm and the sampling algorithm access fewer tuples than the number computed by the general stopping condition, while the number computed by the stopping condition is close to the real stopping point, which shows the effectiveness of the stopping condition.

Figure 5 compares the runtime of the three versions of the exact algorithm and the sampling algorithm with respect to the four aspects tested in Figure 4. The runtime of the Poisson approximation based method is always less than one second, so we omit it in Figure 5 for the sake of the readability of the figures. We also count the number of times in the three versions of the exact algorithm that subset probability values are computed. The trends are exactly the same as their runtime. Limited by space, we omit the figures here. The results confirm that the rule-tuple compression technique and the reordering techniques speed up the exact algorithm substantially. Lazy reordering always outperforms aggressive reordering substantially.

Compared to the exact algorithm, the sampling method is generally more stable in runtime. Interestingly, the exact algorithm (RC+LR) and the sampling algorithm each has its edge. For example, when k is small, the exact algorithm
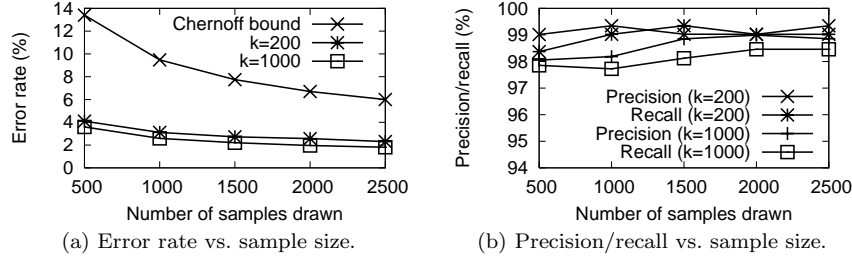
(a) Error rate vs. sample size.

(b) Precision/recall vs. sample size.

Figure 6: The approximation quality of the sampling-based method.

(a) Quality vs. probability.

(b) Quality vs. rule complexity.

(c) Quality vs. k.

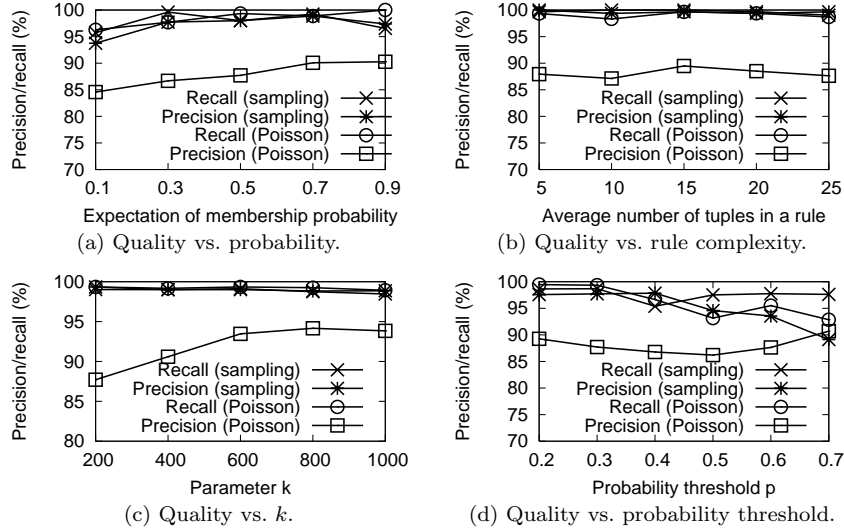(d) Quality vs. probability threshold.

Figure 7: The approximation quality of the sampling method and the Poisson approximation-based method.

is faster. The sampling method is the winner when k is large. As k increases, more tuples need to be scanned in the exact algorithm, and those tuples may be revisited in subset probability computation. But the only overhead in the sampling method is to scan more tuples when generating a sample unit, which is linear in k. This justifies the need for both the exact algorithm and the sampling algorithm.

Figure 6(a) tests the average error rate of the top-k probability approximation using the sampling method. Suppose the top-k probability of tuple t is $Pr^k(t)$, and the top-k probability estimated by the sampling method is $\widehat{Pr}^k(t)$, the average error rate is defined as

$$\frac{\sum_{Pr^k(t)>p} |Pr^k(t) - \widehat{Pr}^k(t)|/Pr^k(t)}{|\{t|Pr^k(t) > p\}|}$$

For reference, we also plot the error bound calculated from the Chernoff-Hoeffding bound [2] given the sample size. We can clearly see that the error rate of the sampling method in practice is much better than the theoretical upper bound.

Moreover, Figure 6(b) shows the precision and recall of the sampling method. The precision is the percentage of tuples returned by the sampling method that are in the actual top-k list returned by the exact algorithm. The recall is the percentage of tuples returned by the exact method that are also returned by the sampling method. The results show that the sampling method only needs to draw a small number of samples to achieve good precision and recall. With

a larger k value, more samples have to be drawn to achieve the same quality

Figure 7 compares the precision and the recall of the sampling method and the Poisson approximation based method. The sampling method achieves better results in general. However, the precision and the recall of the Poisson approximation based method is always higher than 85% with the runtime less than one second. Thus, it is a good choice when the efficiency is a concern.

The recall of the Poisson approximation based method increases significantly when the query parameter k increases. As indicated in [13], the Poisson distribution approximates the Poisson binomial distribution well when the number of Poisson trials is large. When the parameter k increases, more tuples are read before the stopping condition is satisfied. Thus, the Poisson approximation based method provides better approximation for the top-k probability values.

Last, Figure 8 shows the scalability of the exact algorithm and the sampling algorithm. In Figure 8(a), we vary the number of tuples from $20,000$ to $100,000$, and set the number of multi-tuple rules to 10% of the number of tuples. We set $k = 200$ and $p = 0.3$. The runtime increases mildly when the database size increases. Due to the pruning rules and the improvement on extracting sample units, the scan depth (i.e., the number of tuples read) in the exact algorithm and the sampling algorithm mainly depends on k and is insensible to the total number of tuples in the data set.
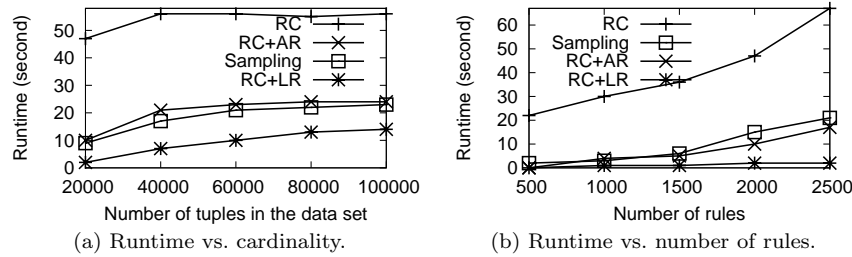
(a) Runtime vs. cardinality.   (b) Runtime vs. number of rules.

Figure 8: Scalability.

In Figure 8(b), we fix the number of tuples to $20,000$, and vary the number of rules from 500 to $2,500$. The runtime of the algorithms increases since more rules lead to smaller tuple probabilities and more scans tuples back and forth in the span of rules. However, the reordering techniques can handle the rule complexity nicely, and make RC+AR and RC+LR scalable.

In all the above situations, the runtime of the Poisson approximation based method is insensitive to those factors, and remains within 1 second.

8. CONCLUSIONS

In this paper, we studied the novel probabilistic threshold top-k queries on uncertain data, which are different in semantics from the recent proposals of top-k queries on uncertain data. An exact algorithm, a sampling method, and a Poisson approximation based method were developed and examined empirically. The results show that they are efficient and each of them has its unique edge.

It is interesting to extend our study to handle different kinds of ranking and preference queries on uncertain data.

9. REFERENCES

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

[2] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. In *STOC'77*.

[3] L. Antova *et al.* 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE'07*.

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

[5] L. Le Cam. An approximation theorem for poisson binomial distribution. *Pacific J. Math.*, 10:1181–1197, 1960.

[6] R. Cheng *et al.* Evaluating probabilistic queries over imprecise data. In *SIGMOD'03*.

[7] R. Cheng *et al.* Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB'04*.

[8] N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS'07*.

[9] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB'04*.

[10] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS'07*.

[11] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, 1996.

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

[13] J. L. Hodges and Jr. L. Le Cam. The poisson approximation to the poisson binomial distribution. *The Annals of Mathematical Statistics*, 31(3):737–740, 1960.

[14] W. Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27:713–721, 1956.

[15] M. Hua *et al.* Efficiently Answering Probabilistic Threshold Top-k Queries on Uncertain Data. In *ICDE'08*.

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

[17] K. Lange. *Numerical analysis for statisticians.* Statistics and computing. 1999.

[18] S. K. Lee. Imprecise and uncertain information in databases: An evidential approach. In *ICDE'92*.

[19] X. Lian and L. Chen. Probabilistic Ranked Queries in Uncertain Databases. In *EDBT'08*.

[20] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, United Kingdom, 1995.

[21] J. Pei *et al.* Probabilistic skylines on uncertain data. In *VLDB'07*, Viena, Austria, September 2007.

[22] C. Ré *et al.* Efficient top-k query evaluation on probabilistic data. In *ICDE'07*.

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

[24] A. Silberstein *et al.* A Sampling-Based Approach to Optimizing Top-k Queries in Sensor Networks. In *ICDE'06*.

[25] S. Singh *et al.* Indexing uncertain categorical data. In *ICDE'07*.

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

[27] Y. Tao *et al.* Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB'05*.

[28] Y. H. Wang. On the number of successes in independent trials. *Statistica Sinica*, 3:295–312, 1993.

[29] K. Yi *et al.* Efficient processing of top-k queries in uncertain databases. In *ICDE'08*.

[30] X. Zhang and J. Chomicki. On the Semantics and Evaluation of Top-k Queries in Probabilistic Databases. In *DBRank'08 Workshop*.