The VLDB Journal (2011) 20:129–153
DOI 10.1007/s00778-010-0196-4

REGULAR PAPER

Ranking queries on uncertain data

Ming Hua · Jian Pei · Xuemin Lin

Received: 5 July 2009 / Revised: 1 June 2010 / Accepted: 2 June 2010 / Published online: 6 July 2010

Abstract Uncertain data is inherent in a few important applications. It is far from trivial to extend ranking queries (also known as top-k queries), a popular type of queries on certain data, to uncertain data. In this paper, we cast ranking queries on uncertain data using three parameters: rank threshold k, probability threshold p, and answer set size threshold l. Systematically, we identify four types of ranking queries on uncertain data. First, a probability threshold top-k query computes the uncertain records taking a probability of at least p to be in the top-k list. Second, a top-(k, l) query returns the top-l uncertain records whose probabilities of being ranked among top-k are the largest. Third, the p-rank of an uncertain record is the smallest number k such that the record takes a probability of at least p to be ranked in the top-k list. A rank threshold top-k query retrieves the records whose p-ranks are at most k. Last, a top-(p, l) query returns the top-l uncertain records with the smallest p-ranks. To answer such ranking queries, we present an efficient exact algorithm, a fast sampling algorithm, and a Poisson approximation-based algorithm. To answer top-(k, l) queries and top-(p, l) queries, we propose *PRist+*, a compact index. An efficient index construction algorithm and efficacious query answering methods are developed for *PRist+*. An empirical study using real and synthetic data sets verifies the effectiveness of the probabilistic ranking queries and the efficiency of our methods.

Keywords Uncertain data · Probabilistic ranking 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 in data transfer. In those applications, ranking queries (also known as top-k 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

The authors are grateful to the anonymous reviewers and Dr. Michael Böhlen, the associate editor, for their constructive and insightful advice on the paper. This research was supported by an NSERC Discovery Grant and an NSERC Discovery Accelerator Supplement Grant. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

M. Hua (✉)
Facebook Inc., Cambridge, MA, USA
e-mail: arceehua@facebook.com

J. Pei
Simon Fraser University, Burnaby, British Columbia, Canada

X. Lin
The University of New South Wales & NICTA, Sydney, New South Wales, Australia

Table 1 Panda presence records

RID	Loc.	Time	Sensor-id	Duration	Conf.
(a) Panda presence records					
$R1$	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

Possible world	Probability	Top-2 on duration
(b) The possible worlds of Table 1a		
$W1 = \{R1, R2, R4, R5\}$	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	$R4, R6$

Generation rules: $R2 \oplus R3$, $R5 \oplus R6$

RID	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$
(c) The top-2 probability values of records in Table 1a						
Probability	0.3	0.4	0.38	0.202	0.704	0.014

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 [1,13,23,36]. 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 record 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 1b shows all possible worlds and their existence probability values.

Ranking 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 1b lists the top-2 records in all possible worlds according to the duration attribute.

It is interesting to examine the probability that a record is in the top-2 lists of all possible worlds (defined as "top-k probability" in Sect. 2.2). It can be calculated by summing up the probabilities of all possible worlds where the record is ranked among top-2. Table 1c shows the top-2 probability of each record in Table 1. More detailed discussion on how to calculate top-k probabilities will be provided in Sect. 2.

To answer the top-2 query on the uncertain data, it is helpful to find the records whose probability values in the top-2 lists are at least p, where p is a user-specified probability threshold. This leads to a probabilistic threshold top-k query [21,22]. In this example, if $p = 0.35$, then $\{R2, R3, R5\}$ should be returned.

Alternatively, we can specify the number of records we want in the answer set. For example, a top-$(2, l)$ query [37] retrieves the top-l records of the largest top-2 probabilities. If $l = 2$, then $\{R5, R2\}$ is returned.

In addition, we can compute the top-k probability for $k = 1, 2, 3, 4$, as shown in Table 2. Since there are at most 4 records in a possible world, the largest possible rank for a record is 4. Interestingly, for a record R_i, we can plot the pair (k, the top-k probability of R_i) on a two-dimensional graph. Figure 1 shows the top-k probabilities of R_2, R_3 and R_5 (we omit other records to make the figure readable). Various queries can be asked on Fig. 1. For example, a probabilistic

Table 2 Top-k probabilities of the records in Table 1a

TID	Top-k probabilities			
	$k=1$	$k=2$	$k=3$	$k=4$
$R1$	0.3	0.3	0.3	0.3
$R2$	0.28	0.4	0.4	0.4
$R3$	0.35	0.5	0.5	0.5
$R4$	0.014	0.202	0.784	1
$R5$	0.056	0.584	0.8	0.8
$R6$	0.0	0.014	0.146	0.2

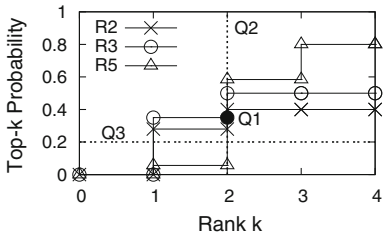


Fig. 1 The top-k probability distribution of R_3 and R_4 in Table 1a

threshold top-2 query with probability threshold 0.35 can be represented by a point $Q1 : (2, 0.35)$. As the answers to the query, the top-k probability curves of R_2, R_3 and R_5 lie northeast to $Q1$ or pass $Q1$. A top-$(2, l)$ query can be represented as a vertical line $Q2 : k = 2$ in Fig. 1. The answer set includes the 2 curves which have the highest intersection points with $Q2$.

Moreover, we can draw a horizontal line $Q3 : p = 0.2$. $Q3$ intersects with the curve of $R5$ at rank 2. This means, only when $k \geq 2$, $R5$ has a probability of at least 0.2 to be ranked among-k. This value k is called the p-rank of $R2$ for $p = 0.2$. If a scientist needs to find the tuples ranked high with a probability of at least 0.5 but does not have any requirement on the actual rank, then the scientist may issue a top-$(p, 3)$ query ($p = 0.5$) to retrieve the top-3 records of the smallest 0.5-ranks. In this example, {$R5$, $R3$, $R4$} should be returned. Top-(p, l) queries are conjugal queries of top-(k, l) queries.

Example 1 demonstrates the ideas of ranking queries on uncertain data. Different from the situation on certain data, ranking queries on uncertain data poses several interesting challenges that will be addressed in this study.

Challenge 1: What does a probabilistic ranking query mean? Various ranking queries may be formulated on uncertain data to address different application scenarios, such as U-Top$k$ queries and U-KRanks queries [39]. We will review the existing ranking queries on uncertain data in Sect. 9.

We address an application scenario other than the recent proposals. Given a probability threshold p, a *probabilistic threshold top-k query* 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. In addition, we discuss top-(k, l) queries and top-(p, l) queries to address the application needs when users do not have specific requirements on confidence level or ranks, respectively. The three kinds of proposed queries provide the flexibility to retrieve top-k results with different requirements.

Challenge 2: How can ranking queries on uncertain data be answered efficiently? A naïve query evaluation method can examine the top-k list in every possible world and derive the final answer. However, the naïve method can be very costly on a large uncertain data set where the number of possible worlds can be huge.

We develop efficient methods to tackle the problem. First, we give an exact algorithm which avoids unfolding all possible worlds. It computes 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. The prefix sharing technique reuses the computation for different tuples. Several pruning techniques are proposed to further improve the efficiency. Second, we devise an effective sampling method to estimates the top-k probabilities. Third, we develop a Poisson approximation based method to answer probabilistic threshold top-k queries in linear time. The proposed query evaluation methods can be extended to answer top-(k, l) queries and top-(p, l) queries.

Challenge 3: Can we develop an efficient method to achieve online query answering? To achieve the best analysis results, users may be interested in applying different queries on uncertain data with varying query parameter values. Such real-time interaction requires online processing of probabilistic ranking queries.

We develop *PRist+*, an index structure which can be used to answer probabilistic threshold top-k queries, top-(k, l) queries and top-(p, l) queries efficiently. We develop an efficient construction method for *PRist+*. Query answering methods for probabilistic threshold top-k queries, top-(k, l) queries and top-(p, l) queries using *PRist+* are discussed.

The rest of the paper is organized as follows. In Sect. 2, we formulate the three types of probabilistic ranking queries. How to compute the exact top-k probabilities of tuples is discussed in Sect. 3. We develop an exact query answering algorithm in Sect. 4. We devise a sampling method in Sect. 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 Sect. 6. Section 7 discusses the *PRist+* index and the query evaluation methods based on *PRist+*. We review the related work in Sect. 9. A systematic empirical evaluation is reported in Sect. 8. The paper is concluded in Sect. 10.

2 Probabilistic databases and ranking queries

In this section, we review the basic notions in uncertain and probabilistic data processing, and define the queries to be addressed in the paper.

2.1 Probabilistic databases and possible worlds

We consider uncertain data in the *possible worlds* semantics model [1,13,23,36], which has been extensively adopted by the recent studies on uncertain data processing, such as [5,34,39].

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 constraint on a set of tuples in the form of $R : t_{r_1}, \ldots, t_{r_m}$ where $t_{r_i} \in T$ $(1 \leq i \leq m)$. We consider the following two kinds of constraints discussed in [28,36].

A *mutual exclusion constraint* (denoted by $\oplus$) specifies the mutual exclusiveness among the tuples in the same rule. *exclusive rule* $R_\oplus = t_{r_1} \oplus \cdots \oplus t_{r_m}$ $(\sum_{i=1}^{m} Pr(t_{r_i}) \leq 1)$ constraints 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. The *probability* of an exclusive rule is the sum of the membership probability values of all tuples involved in the rule, denoted by $Pr(R_\oplus) = \sum_{t \in R_\oplus} Pr(t)$. The set of exclusive rules in T is $\mathcal{R}_\oplus$.

A *mutual inclusion constraint* (denoted by $\equiv$) specifies the coexistence of the tuples involved in the same rule. For example, herd animals such as zebras often live in groups. Similar to the scenario in Example 1, sensors may be deployed to monitor the presence of zebras. Each record returned by sensors can be considered as an uncertain tuple. If external evidence shows that a small group of zebras stayed together during the time of monitoring, then this external knowledge can be described as a mutual inclusion constraint. An *inclusive rule* $R_\equiv : t_{r_1} \equiv \cdots \equiv t_{r_m}$ restricts that, among all tuples $t_{r_1}, \ldots, t_{r_m}$ involved in the same rule, either no tuple appears or all tuples appear in a possible world. All tuples in $R_\equiv$ have the same membership probability value, which is also the probability of rule $R_\equiv$, denoted by $Pr(R_\equiv) = Pr(t_{r_1}) = \cdots = Pr(t_{r_m})$. The set of inclusive rules in T is $\mathcal{R}_\equiv$.

Following [5,39], 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 $R_t : t$ whose probability $Pr(R_t) = Pr(t)$. The set of trivial rules in T is $\mathcal{R}_t$. Therefore, conceptually, an uncertain table T comes with a set of generation rules $\mathcal{R} = \mathcal{R}_\oplus \cup \mathcal{R}_\equiv \cup \mathcal{R}_t$ such that each tuple is involved in one and only one generation rule in $\mathcal{R}$. We write $t \in R$ if tuple t is involved in rule R.

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 $|R| > 1$. In the rest of the paper, we also call a multi-tuple rule a generation rule, for the sake of simplicity. 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 satisfying $R \cap W \in Domain(R)$ for each generation rule R, where $Domain(R)$ is the set of all valid tuples in a possible world defined as follows.

1. If R is an exclusive rule in $\mathcal{R}_\oplus$, then

$$Domain(R) = \begin{cases} \{\{t\} | t \in R\}, & \text{if } Pr(R) = 1; \\ \{\{t\} | t \in R\} \cup \{\emptyset\}, & \text{if } Pr(R) < 1. \end{cases}$$

2. If R is an inclusive rule in $\mathcal{R}_\equiv$, then

$$Domain(R) = \begin{cases} \{R\}, & \text{if } Pr(R) = 1; \\ \{\emptyset, R\}, & \text{if } Pr(R) < 1. \end{cases}$$

We denote by $\mathcal{W}$ the set of all possible worlds. Clearly, for an uncertain table T with a set of generation rules $\mathcal{R}$, the number of all possible worlds is $|\mathcal{W}| = \prod_{R \in \mathcal{R}} |Domain(R)|$.

Each possible world is associated with an *existence probability* $Pr(W)$ that the possible world happens. Following from the basic probability principles, we have $Pr(W) = \prod_{R \in \mathcal{R}, |R \cap W| \geq 1} Pr(R \cap W) \prod_{R \in \mathcal{R}, R \cap W = \emptyset} (1 - Pr(R))$.

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

2.2 Probabilistic ranking queries

Top-k queries (also known as ranking queries) [15,17,32,33] are a category of important queries in data analysis. Given a set of tuples and a *ranking function* f, all tuples can be ranked according to the ranking function. 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 f. The *ranking order* $\preceq_f$ is a total order on all tuples. Ties can be broken arbitrarily.

Given a set of certain tuples, a *top-k query* Q_f^k returns the top-k tuples ranked higher than the other tuples in the ranking order $\preceq_f$. We often denote a top-k query by Q^k or Q when the predicate and the ranking function are unimportant in our discussion.

How can we apply top-k queries to probabilistic tables? Consider a top-k query Q_f^k on an uncertain table T. All tuples can be ranked according to f. In a possible world W, let $W_f(j)$ be the tuple ranked at the j-th position in W according to f.

For a tuples $t \in T$, the *position probability* $Pr(t, j)$ is the probability that t is ranked at the j-th position in possible worlds according to f. That is, $Pr(t, j) = \sum_{W \in \mathcal{W} \text{ s.t. } t=W_f(j)} Pr(W)$.

Moreover, for a tuple t, the *top-k probability* $Pr^k(t)$ is the probability that t is ranked among-k in possible worlds according to f, that is, $Pr^k(t) = \sum_{j=1}^{k} Pr(t, j)$.

Given a *rank parameter* $k > 0$ and a probability threshold $p \in (0, 1]$, a *probabilistic threshold top-k query* (*PT-k query* for short) [21,22] finds the tuples whose top-k probabilities are at least p.

Alternatively, a user can use an *answer set size constraint* $l > 0$ to replace the probability threshold p and issue a *probabilistic top-(k, l) query* [37,44], which finds the top-l tuples with the highest top-k probabilities.

Now, let us consider the conjugal queries of PT-k queries and top-(k, l) queries.

For a tuple $t \in T$, given a probability threshold $p \in (0, 1]$, the *p-rank* of t is the minimum k such that $Pr^k(t) \geq p$. That is, $MR_p(t) = \min\{k | Pr^k(t) \geq p\}$.

Given a *probability threshold* $p \in (0, 1]$ and a *rank threshold* $k > 0$, a *rank threshold top-k query* (*RT-k query* for short) retrieves the tuples whose p-ranks are at most k. RT-k queries are conjugal queries of PT-k queries.

Alternatively, a user can replace the rank threshold k by an *answer set size constraint* $l > 0$ and issue a *top-(p, l) query*, which returns the top-l tuples with the smallest p-ranks. Clearly, top-(p, l) queries are conjugal queries of top-(k, l) queries.

The answers to a PT-k query and a RT-k query with the same parameter values are identical. However, for top-(k, l) queries and top-(p, l) queries, even they share the same value on l, the answers generally may not be the same.

3 Exact position probability computation

In this section, we first introduce how to compute the exact position probability values. Top-k probabilities and p-ranks can be directly derived from position probabilities.

For a tuple $t \in T$ and a possible world W such that $t \in W$, whether $t \in W_f(k)$ depends only on how many other tuples in T ranked higher than t appear in W. Technically, for a tuple $t \in T$, the *dominant set* of t is the subset of tuples in T that are ranked higher than t, i.e., $S_t = \{t' | t' \in T \wedge t' \prec_f t\}$.

Theorem 1 (Dominant set) *For a tuple $t \in T$, $Pr^k_{Q,T}(t) = Pr^k_{Q,S_t}(t)$.*

Using the dominant set property, we scan the tuples in T in the ranking order and derive the position probabilities for each tuple $t \in T$ based on the tuples preceding t. Generation rules involving multiple tuples are handled by the rule-tuple compression technique developed later in this section.

Table 3 Top-k probabilities of a set of tuples

TID	Rank	Prob.	Top-k probabilities			
			$k = 1$	$k = 2$	$k = 3$	$k = 4$
t_1	1	0.5	0.5	0.5	0.5	0.5
t_2	2	0.3	0.15	0.3	0.3	0.3
t_3	3	0.7	0.245	0.595	0.7	0.7
t_4	4	0.9	0.0945	0.45	0.8055	0.9

3.1 The basic case

We start with the basic case, where we assume that all tuples are independent. Let $L = t_1 \ldots t_n$ be the list of all tuples in table 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 *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, $Pr(t_i, j) = Pr(t_i) Pr(S_{t_i}, j - 1)$. Apparently, the top-$k$ probability of t_i is given by $Pr^k(t_i) = \sum_{j=1}^{k} Pr(t_i, j) = Pr(t_i) \sum_{j=1}^{k} Pr(S_{t_i}, j-1)$. Particularly, when $i \leq k$, we have $Pr^k(t_i) = Pr(t_i)$.

The following theorem can be used to compute the top-k probability values efficiently.

Theorem 2 (Poisson binomial recurrence [24]) *In the basic case, for $1 \leq i, j \leq |T|$,*

1. $Pr(S_{t_i}, 0) = Pr(S_{t_{i-1}}, 0)(1 - Pr(t_i)) = \prod_{j=1}^{i}(1 - Pr(t_i))$;
2. $Pr(S_{t_i}, j) = Pr(S_{t_{i-1}}, j - 1) Pr(t_i) + Pr(S_{t_{i-1}}, j)(1 - Pr(t_i))$.

Proof 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 [24]. □

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 the uncertain tuples in Table 3 and a top-k query Q. Suppose $t_1, \ldots, t_4$ are in the ranking order according to f and 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, $Pr^3(t_1) = 0.5$, $Pr^3(t_2) = 0.3$, and $Pr^3(t_3) = 0.7$.

To compute $Pr^3(t_4)$, we first compute $Pr(S_{t_4}, 0) = 0.105$, $Pr(S_{t_4}, 1) = 0.395$, and $Pr(S_{t_4}, 2) = 0.395$ using

Theorem 2. Then, we have $Pr^3(t_4) = Pr(t_4)(Pr(S_{t_4}, 0) + Pr(S_{t_4}, 1) + Pr(S_{t_4}, 2)) = 0.8055$.

3.2 Handling generation rules

In general, a probabilistic table may contain some multi-tuple generation rules (generation rules or rules for short). For a tuple $t \in T$, two situations due to the presence of generation rules complicate the computation.

First, there may be a rule R such that some tuples involved in R are ranked higher than t. Second, t itself may be involved in a generation rule R. In both cases, some tuples in S_t are dependent and thus Theorem 2 cannot be applied directly. Can dependent tuples in S_t be transformed to independent ones so that Theorem 2 can still be used?

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

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.

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: $R_{left} = \{t_{r_1}, \dots, t_{r_{m_0}}\}$ and $R_{right} = \{t_{r_{m_0}+1}, \dots, t_{r_m}\}$. $Pr^k(t_i)$ is affected by tuples in R_{left} only and not by those in R_{right}. Two subcases may arise, according to whether t belongs in R or not: in subcase 1, $t_i \notin R$; in subcase 2, $t_i \in R$, i.e., $t_i = t_{r_{m_0}+1}$.

Since in Case 1, generation rule R can be ignored, in the rest of this section, we mainly discuss how to handle exclusive rules and inclusive rules in Case 2 and Case 3.

3.2.1 Handling exclusive rules

We first consider computing $Pr^k(t_i)$ when an exclusive rule $R : t_{r_1} \oplus \cdots \oplus t_{r_m}$ $(1 \leq r_1 < \cdots < r_m \leq n)$ is involved.

In Case 2, t_i is ranked lower than all tuples in R. At most one tuple in R can appear in a possible world. According to Theorem 1, we can combine all tuples in R into an *exclusive rule-tuple* t_R with membership probability $Pr(R)$.

Corollary 1 (Exclusive rule-tuple compression) *For a tuple $t \in T$ and a multi-tuple exclusive 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 $T(R)$.*

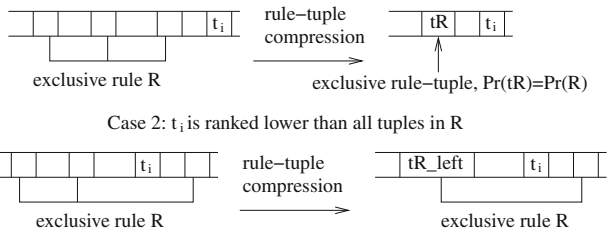


Fig. 2 Computing $Pr^k(t_i)$ for one tuple t_i

In Case 3, t_i is ranked between the tuples in R, which can be further divided into two subcases. First, if $t_i \notin R$, similar to Case 2, we can compress all tuples in R_{left} into an exclusive rule-tuple $t_{r_1,\dots,r_{m_0}}$ where membership probability $Pr(t_{r_1,\dots,r_{m_0}}) = \sum_{j=1}^{m_0} Pr(t_{r_j})$, and compute $Pr^k(t_i)$ using Corollary 1.

Second, if $t_i \in R$, 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 exclusive rule) *For a tuple $t \in R$ such that $|R| > 1$, $Pr^k_{Q,T}(t) = Pr^k_{Q,T'}(t)$ 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 exclusive rules one by one. Each multi-tuple exclusive rule can be handled by one of the above two cases as illustrated in Fig. 2, and the dependent tuples in S_t can be either compressed into some exclusive rule-tuples or removed due to the involvement in the same exclusive rule as t. After the exclusive rule-tuple compression, the resulting set is called the *compressed dominant set* of t, denoted by $T(t)$. Based on the aforementioned discussion, for a tuple $t \in T$, 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.

Example 3 (Exclusive rule-tuple compression) Consider a list of tuples $t_1, \dots, t_{11}$ in the ranking order. Suppose we have two multi-tuple exclusive 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 an exclusive rule-tuple $t_{2,4}$ with membership probability $Pr(t_{2,4}) = Pr(t_2) + Pr(t_4)$. 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, $T(t_6) = \{t_1, t_{2,4}, t_3, t_5\}$. Since the tuples in $T(t_6) \cup \{t_6\}$ are independent, we can apply Theorem 2 to compute $Pr^3(t_6)$ using $T(t_6)$.

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 can sort all tuples in 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.

3.2.2 Handling inclusive rules

Then, let us consider how to handle an inclusive rule $R : t_{r_1} \equiv \cdots \equiv t_{r_m}$ ($t_{r_j} \in T(t_i)$ for $1 \leq j \leq m$) in T. Again, we only need to consider Case 2 and Case 3.

In Case 2, t_i is ranked lower than all tuples in R. Either 0 or $|R|$ tuples in R appear in a possible world. We combine all tuples in R into an *inclusive rule-tuple* t_R with membership probability $Pr(R)$. Different from exclusive rule-tuples discussed in Sect. 3.2, an inclusive rule-tuple takes $|R|$ positions in $T(t_i)$. Therefore, the subset probability computation for $T(t_i)$ containing an inclusive rule-tuple is different from Theorem 2.

Corollary 3 (Inclusive rule-tuples) *Given a tuple $t \in T$ and a multi-tuple inclusive rule R, if $\forall t' \in R, t' \prec_f t$, let $T(R) = S_t - \{t' | t' \in R\} + \{t_R\}$ where $Pr(t_R) = Pr(R)$ and $t_R \prec_f t$, then for $1 \leq j \leq |T|$, $Pr(S_t, j) = Pr(T(R), j)$. Moreover,*

1. $Pr(S_t, 0) = Pr(T(R) - \{t_R\}, 0)(1 - Pr(t_R))$;
2. $Pr(S_t, j) = Pr(T(R) - \{t_R\}, j)(1 - Pr(t_R))$ *for* $0 < j < |R|$;
3. $Pr(S_t, j) = Pr(T(R) - \{t_R\}, j - |R|)Pr(t_R) + Pr(T(R) - \{t_R\}, j)(1 - Pr(t_R))$ *for* $j \geq |R|$.

Proof From the definition of inclusive rules, we have $Pr(t_{r_1}, \ldots, t_{r_m}) = Pr(R)$. The results follow with basic probability principles. □

In Case 3, t_i is ranked between tuples in R. We again consider two subcases. First, if $t_i \notin R$, we compress all tuples in R_{left} into an inclusive rule-tuple $t_{R_{left}}$, which takes $|R_{left}|$ positions and has membership probability $Pr(t_{R_{left}}) = Pr(R)$. Corollary 3 can be used to compute $Pr^k(t_i)$.

Second, if $t_i \in R$, then whenever t_i appears in a possible world, all tuples in R also appear in the same possible world. Moreover, t_i is always ranked lower than the tuples in R_{left} (i.e., the tuples in R that are ranked higher than t_i) when it appears. The top-k probability of t_i in T equals to the top-$(k - |R_{left}|)$ probability of t_i in $T - R_{left}$.

Corollary 4 (Tuple in inclusive rule) *For a tuple $t \in T$ and a multi-tuple inclusive rule R such that $t \in R$, $Pr^k_{Q,T}(t) = Pr^{k-|R_{left}|}_{Q,T'}(t)$ where uncertain table $T' = (S_{t_i} - \{t' | t' \in R\}) \cup \{t\}$.*

Example 4 (Inclusive rule-tuple compression) Consider a list of tuples $t_1, \ldots, t_{11}$ in the ranking order. Now, suppose we have two multi-tuple inclusive rules: $R_1 : t_6 \equiv t_8$ and $R_2 : t_7 \equiv t_9$, whose probabilities are 0.8 and 0.1, respectively. Let us consider how to compute $Pr^4(t_9)$.

The dominant set S_{t_9} is $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$. First, since t_7 is in the same inclusive rule with t_9, the second subcase of Case 3 applies. We remove t_7 from the dominant set of t_9, and compute the top-3 probability of t_9 using $S'_{t_9} = \{t_1, t_2, t_3, t_4, t_5, t_6, t_8\}$, denoted by $Pr^3(t_9)'$.

Second, all tuples in R_1 are ranked higher than t_9, so Case 2 applies. We combine t_6 and t_8 to an inclusive rule-tuple $t_{6,8}$ whose membership probability $Pr(t_{6,8}) = Pr(t_6)$. Now the dominant set of t_9 becomes $S''_{t_9} = \{t_1, t_2, t_3, t_4, t_5, t_{6,8}\}$ where $t_{6,8}$ takes two positions in S''_{t_9}. Let $S = S''_{t_9} - t_{6,8}$. We first calculate $Pr(S, j)$ for $0 \leq j \leq 2$ using Theorem 2 since no generation rules are involved in S. Then, we can use Corollary 3 to calculate $Pr(S''_{t_9}, j)$ for $0 \leq j \leq 2$.

For any tuple t_i, we can scan its dominant set S_{t_i} once and derive the compressed dominant set $T(t_i)$ where both exclusive rules and inclusive rules are properly processed so that all tuples in the compressed dominant set are independent. Then, $Pr^k(t_i)$ can be computed using either Theorem 2 or Corollary 3, depending on whether inclusive rules are involved or not.

4 Exact query answering methods

Straightforwardly, to answer a PT-k query with probability threshold p, we simply scan all tuples in T in the ranking order and compute the top-k probability of each tuple. The tuples whose top-k probabilities passing the threshold p are returned as the answers. Top-(k, l) queries and top-(p, l) queries can be answered similarly.

Can we further improve the efficiency of the query answering methods? In Sect. 4.1, we discuss how to reuse subset probability calculation during computing the top-k probability values for all tuples. In Sect. 4.2, we develop several effective pruning techniques.

4.1 Scan reduction by prefix sharing

We scan the dominant set S_{t_i} of each tuple $t_i \in T$ once and compute the subset probabilities $Pr(S_{t_i}, j)$. Can we reduce the number of scans of the sorted tuples to improve the efficiency of query evaluation?

To compute $Pr^k(t_i)$ using subset probability $Pr(S_{t_i}, j)$, the order of tuples in S_{t_i} does not matter. This gives us the flexibility to order 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. In this section, we introduce two reordering methods to achieve good sharing.

4.1.1 Aggressive reordering

Consider the list $L = t_1 \ldots t_n$ of all tuples in T and a tuple t_i in L. Two observations 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 Sect. 3.2), 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 Sect. 3.2).

Second, there can be multiple rules open with respect to t_i. Each such 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 in descending order.

An *aggressive reordering 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.

We scan all tuples in T in the ranking order. Two buffer lists, L_{complete} and L_{open}, are used to help aggressive reordering. L_{complete} contains all independent tuples or completed rule-tuples, while L_{open} contains all open rule-tuples during the scan. Both L_{complete} and L_{open} are initialized to $\emptyset$ before the scan.

When scanning tuple t_i, we compute the compressed dominant set of t_i, and update L_{complete} and L_{open} according to the following two cases.

Case 1: If t_i is an independent tuple, then the compressed dominant set of t_i contains all tuples in L_{complete} and L_{open}. Moreover, we put t_i into L_{complete}, meaning that t_i will appear in the compressed dominant set of all tuples ranked lower than t_i.

Case 2: If t_i is involved in a multi-tuple generation rule $R : t_{r_1}, \ldots, t_{r_m}$, then the compressed dominant set of t_i contains all tuples in L_{complete} and L_{open}, except for the rule-tuple $t_{R_{left}}$ in L_{open}, where $t_{R_{left}}$ is the rule-tuple compressed from all tuples in R that are ranked higher than t_i.

In order to update L_{complete} and L_{open}, the following two subcases arise. First, if t_i is not the last tuple in R (i.e., $t_i = t_{r_{m_0}}$ where $1 \leq m_0 < m$), then we update rule-tuple $t_{R_{left}}$ by compressing t_i into $t_{R_{left}}$, using the methods discussed in Sect. 3.2. If $t_{R_{left}}$ is not in L_{open}, then we add $t_{R_{left}}$ into L_{open}. Moreover, we sort the rule-tuples in L_{open} in their next tuple indices descending order. Second, if t_i is the last tuple in R, which means that the rule-tuple t_R will never be updated later. Therefore, we remove $t_{R_{left}}$ from L_{open}, and add t_R into L_{complete}.

Table 4 Results of reordering techniques

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	

The subset probabilities of the tuples in L_{complete} only need to be calculated once and can be reused by all tuples ranked lower than them. In contrast, the rule-tuples in L_{open} may be updated when other tuples in the same rule are scanned. Therefore, only part of the subset probabilities can be reused.

For two consecutive tuples t_i and t_{i+1} in the sorted list L of all tuples in 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 the aggressive reordering 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 $Cost = \sum_{i=1}^{n-1}(|L(t_{i+1})| - |Prefix(L(t_i), L(t_{i+1}))|)$.

Example 5 (Aggressive reordering) Consider a list of ranked tuples $t_1, \ldots, t_{11}$ with two multi-tuple rules $R_1 : t_1 \oplus t_2 \oplus t_8 \oplus t_{11}$ and $R_2 : t_4 \equiv t_5 \equiv t_{10}$. The compressed dominant sets of tuples in the orders made by the aggressive reordering method is listed in Table 4.

For example, before scanning t_6, L_{complete} contains independent tuple t_3 and L_{open} contains rule-tuples $t_{4,5}$ and $t_{1,2}$. $t_{4,5}$ is ranked before $t_{1,2}$, since the next tuple in R_2, t_{10}, is ranked lower than R_1's next tuple t_8. Since t_6 is independent, the compressed dominant set of t_6 includes all 3 tuples in L_{complete} and L_{open}. $T(t_6)$ and $T(t_5)$ only share the common prefix t_3, therefore, the cost of calculating the subset probabilities for $T(t_6)$ is $3 - 1 = 2$. After scanning t_6, t_6 is added into L_{complete}.

The total cost by using the aggressive reordering method is $Cost_{\text{aggressive}} = 15$. As a comparison, without reordering, the total number of subset probability values needed to be calculated is the sum of lengths of all compressed dominant sets, which is 31.

4.1.2 Lazy reordering

On the other hand, a *lazy method* always reuses the longest common prefix in the compressed dominant set of the last tuple scanned, and reorders only the tuples not in the common prefix using the two observations discussed in Sect. 4.1.1.

We scan the tuples in T in the ranking order. During the scan, we maintain the compressed dominant set of the last scanned tuple. When processing tuple t_i, one of the following two cases may apply.

Case 1: If t_i is an independent tuple, or t_i is the first tuple scanned in a multi-tuple generation rule R, then the compressed dominant set of t_i can be computed by one of the following two subcases.

First, if t_{i-1} is independent, then $T(t_i)$ can be obtained by adding t_{i-1} to the rear of $T(t_{i-1})$.

Second, if t_{i-1} is involved in a multi-tuple generation rule R', then $T(t_i)$ is computed by adding $t_{R'_{left}}$ to the rear of $T(t_{i-1})$.

Case 2: If t_i is involved in a multi-tuple generation rule R but not the first tuple scanned in R, then there are three subcases.

First, if t_{i-1} is involved in the same rule with t_i, then $T(t_i) = T(t_{i-1})$.

Second, if t_{i-1} is an independent tuple, then $T(t_{i-1})$ must contain a rule-tuple $t_{R_{left}}$ corresponding to rule R, which should not be included in $T(t_i)$. Moreover, t_{i-1} should be added at the rear of $T(t_{i-1})$. In that case, the longest common prefix of $T(t_{i-1})$ and $T(t_i)$ includes the tuples ranked before $t_{R_{left}}$ in $T(t_{i-1})$. The subset probabilities for the tuples in the longest common prefix can be reused. For those tuples or rule-tuples not in the longest common prefix, we reorder them so that the independent tuples are always sorted before the rule-tuples and the rule-tuples are sorted in their next tuple indices descending order.

Third, if t_{i-1} is involved in another rule $R' \neq R$, then there are two differences between $T(t_{i-1})$ and $T(t_i)$: (1) $T(t_{i-1})$ contains $t_{R_{left}}$ but $T(t_i)$ does not; (2) $T(t_i)$ includes $t_{R'_{left}}$ but $T(t_{i-1})$ does not. Therefore, we first add $t_{R'_{left}}$ to the rear of $t_{R'_{left}}$. Then, as discussed in the second subcase, we can reuse the longest common prefix of $T(t_{i-1})$ and $T(t_i)$, and reorder the tuples not in the longest common prefix.

Example 6 (Lazy reordering) Consider a list of ranked tuples $t_1, \ldots, t_{11}$ with multi-tuple rules $R_1 : t_1 \oplus t_2 \oplus t_8 \oplus t_{11}$ and $R_2 : t_4 \equiv t_5 \equiv t_{10}$ again. The compressed dominant sets of tuples in the orders made by the lazy reordering method is listed in Table 4.

The lazy reordering method orders the compressed dominant sets in the same way as the aggressive reordering method 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. The total cost of the lazy reordering method is 12.

We can show that the lazy method is never worse than the aggressive method.

Theorem 3 (Effectiveness of lazy reordering) *Given a ranked list of tuples in T, let $Cost(agg)$ and $Cost(lazy)$ be the total number of subset probability values needed to be calculated by the aggressive reordering method and the lazy reordering method, respectively. Then, $Cost(agg) \geq Cost(lazy)$.*

Proof For two consecutive tuples t_i and t_{i+1} in T ($1 \leq i \leq |T| - 1$), we consider the following three cases.

First, if t_i and t_{i+1} are involved in the same generation rule, then the cost of computing $T(t_{i+1})$ is 0 using either aggressive reordering or lazy reordering, since $T(t_{i+1})$ contains the same set of tuples in $T(t_i)$.

Second, if t_{i+1} is an independent tuple or the first tuple in a generation rule R, then the cost of computing $T(t_{i+1})$ using lazy reordering is 1, since a tuple t_i (if t_i is independent) or rule-tuple $t_{R'_{\text{left}}}$ (if t_i is involved in R') should be added into $T(t_i)$ to form $T(t_{i+1})$. The cost of computing $T(t_{i+1})$ using aggressive reordering is at least 1.

Third, if t_{i+1} is involved in rule R but is not the first tuple in R, then $t_{R_{\text{left}}}$ must be removed from $T(t_i)$. Moreover, t_i or $t_{R'_{\text{left}}}$ should be added into $T(t_i)$, as discussed in the second case. Let L_{reorder} be the set of tuples or rule-tuples in $T(t_i)$ that are ranked lower than $t_{R_{\text{left}}}$, then the cost of computing $T(t_{i+1})$ is $|L_{\text{reorder}}| + 1$. Now let us show that, using aggressive reordering, the same amount of cost is also needed before scanning t_{i+1}. For each tuple $t \in L_{\text{reorder}}$, one of the following two subcases may arise: (1) t is an independent tuple or a completed rule-tuple, then t must be put into L_{complete} using aggressive reordering. The subset probability of t_R need to be recomputed once L_{complete} is updated. Thus, 1 cost is required. (2) t is an open-rule tuple, then it must be put into L_{open} using aggressive reordering. The subset probability of t needs to be recomputed after removing L_R, which requires a cost of 1.

Therefore, in any of the three cases, the cost of lazy reordering is not more than the cost of aggressive reordering. The conclusion holds. □

4.2 Pruning techniques

So far, we implicitly have a requirement: all tuples in T are scanned in the ranking order. However, a probabilistic ranking query or conjugal query is interested in only those tuples passing the query requirement. Can we avoid retrieving or checking all tuples satisfying the query predicates?

Some existing methods such as the well-known TA algorithm [17] can retrieve in batch tuples satisfying the predicate in the ranking order. Using such a method, we can retrieve tuples in T progressively in the ranking order. Now, the problem becomes how we can use the tuples seen so far to prune some tuples ranked lower in the ranking order.

Consider rank parameter k and probability threshold p. We give the following pruning rules: Theorems 4 and 5 can avoid checking some tuples that cannot satisfy the probability threshold, and Theorems 6 and 7 specify stopping conditions. The tuple retrieval method (e.g., an adaption of the TA algorithm [17]) uses the pruning rules in the retrieval. Once it can determine all remaining tuples in 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 4 (Pruning by membership probability) *For a tuple $t \in T$, $Pr^k(t) \leq Pr(t)$. Moreover, if t is an independent tuple and $Pr^k(t) < p$, 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 $t'' \in R$.*

Proof To prove the first item, we only need to show that $S_t \subset S_{t'}$, then $\sum_{0 \leq i \leq k} Pr(S_t, i) \geq \sum_{0 \leq j \leq k} Pr(S_{t'}, j)$. Thus, the conclusion holds. The second item can be proved similarly. □

To use Theorem 4, 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 5 (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 $Pr^k(t') < p$.*

Proof Since t and t' are in the same rule and $t \preceq_f t'$, we have $S_t \subseteq S_{t'}$. The conclusion holds following the similar proof of Theorem 4. □

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 6 (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 $t' \notin A$, $Pr^k(t') < p$.*

Proof For each $1 \leq j \leq k$, $\sum_{t \in A} \Pr(t, j) = 1$. Therefore, $\sum_{t \in A} \Pr^k(t) = k$. □

Each pruning rule has its own edge. Theorem 4 is very effective for pruning independent tuples and the rule-tuples as a whole. Theorem 5 targets at the tuples in the same rule. Both Theorems 4 and 5 are based on the condition that, there is a tuple t failing the query condition and a tuple t' has a smaller membership probability than t. When this condition does not meet, we can consider using the third pruning rule, Theorem 6. By utilizing the three pruning rules together, we can capture more situations where an early stop of the scan is possible.

Moreover, we have a tight stopping condition as follows.

Theorem 7 (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 $Pr^k(t_i) \leq \max_{R \in LR}(1 - Pr(t_{R_{\text{left}}})) \sum_{j=0}^{k-1} Pr(L - t_{R_{\text{left}}}, j)$.*

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.

Algorithm 1 The exact algorithm with reordering and pruning techniques

Input: an uncertain table T, a set of generation rules $\mathcal{R}$, a top-k query Q_f^k, and a probability threshold p
Output: $Answer(Q, p, T)$
Method:
1: retrieve tuples in T in the ranking order one by one
2: **for all** $t_i \in T$ **do**
3: compute $T(t_i)$ by rule-tuple compression and reordering
4: compute subset probability values and $Pr^k(t_i)$
5: **if** $Pr^k(t_i) \geq p$ **then**
6: output t_i
7: **end if**
8: check whether t_i can be used to prune future tuples
9: **if** all remaining tuples in T fail the probability threshold **then**
10: exit
11: **end if**
12: **end for**

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

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

$$\leq (1 - Pr(t_{R_{\text{left}}})) \sum_{j=0}^{k-1} Pr(L - t_{R_{\text{left}}}, j)$$

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

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

□

Theorem 7 provides two 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 Algorithm 1. We analyze the complexity of the algorithm as follows.

For a multi-tuple rule $R : t_{r_1}, \ldots, t_{r_m}$ 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. Each subset probability value can be computed based on the previously computed subset probability values using Theorem 2, which takes $O(1)$ time. Moreover, in the worst case where all tuples in T pass the probability threshold, all tuples in T have to be read at least once. The time complexity of the whole algorithm is $O(kn + k \sum_{R \in \mathcal{R}} span(R))$. It is a polynomial time algorithm.

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

Interestingly, since PT-k query answering methods can be extended to evaluate top-(k, l) queries and top-(p, l) queries, the pruning techniques introduced in this section can be applied to answering top-(k, l) queries and top-(p, l) queries as well.

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 estimating top-k probabilities of tuples.

For a tuple t, let X_t be a random variable as an indicator to the event that t is ranked among-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 $\mathcal{R}$, 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 exclusive 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 exclusive rule. For a multi-tuple inclusive rule $R : t_{r_1} \equiv \cdots \equiv t_{r_m}$, s takes a probability of $Pr(R)$ to include all tuples involved in R.

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.

Theorem 8 (Sample size) *For any $\delta \in (0, 1)$, $\epsilon > 0$, and a sample S of possible worlds, if $|S| \geq \frac{3 \ln \frac{2}{\delta}}{\epsilon^2}$, then for any tuple t,* $\Pr\{|E_S[X_t] - E[X_t]| > \epsilon E[X_t]\} \leq \delta$.

Proof The theorem follows with the well known Chernoff-Hoeffding bound [3]. □

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

First, we can sort all tuples in 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 $|T|$ when $k \ll |T|$.

Second, in practice, the actual approximation quality may converge well before the sample size reaches the bound given in Theorem 8. 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.

To answer a PT-k query with probability threshold p, we first run the above sampling algorithm. Then, we scan the tuples in L and output the tuples whose estimated top-k probabilities are at least p.

After obtaining estimated top-k probabilities of tuples using the above sampling method, top-(k, l) queries and top-(p, l) queries can be answered similarly.

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 [19]. Since the PT-k query answering methods can be extended to evaluate top-(k, l) queries and top-(p, l) queries, the Poisson approximation-based method can be used to answer top-(k, l) queries and top-(p, l) queries too. We omit the details to avoid redundance.

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 $t \in T$, 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 exclusive rule-tuple in $T(t)$ with probability 1, we can remove the tuple from $T(t)$, and compute the top-$(k-1)$ probability of t. For an inclusive rule-tuple R with probability 1, we remove it from $T(t)$ and compute the top-$(k - |R|)$ 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 exclusive rule $R_\oplus$ $(R_\oplus \cap T(t) \neq \emptyset)$, we combine the tuples in $R_\oplus \cap T(t)$ into a rule-tuple $t_{R_\oplus}$ such that $Pr(t_{R_\oplus}) = \sum_{t' \in R_\oplus \cap T(t)} Pr(t')$, and construct a random trial $X_{t_{R_\oplus}}$ whose success probability $Pr(X_{t_{R_\oplus}} = 1) = Pr(t_{R_\oplus})$. For each multi-tuple inclusive $R_\equiv$ $(R_\equiv \cap T(t) \neq \emptyset)$, we construct two sets of trials: one contains all tuples in $R_\equiv$ and we compute the top-$(k - |R_\equiv|)$ probability of t; the other sets of trials do not contain any tuple in $R_\equiv$ and we compute the top-k probability of t. Therefore, hereafter, we only focus on the trials without multi-tuple inclusive rules.

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 $\mu = E[X]$ [20]. Therefore, when the query parameter k varies from 1 to $|T(t)| + 1$, $Pr(t, k)$ follows the similar trend.

Corollary 5 (Distribution of position probability) *For a tuple $t \in T$,*

$$1.\quad Pr(t,k) \begin{cases} = 0, & if\ k > |T(t)| + 1; \\ < Pr(t, k+1), & if\ k \leq \mu - 1; \\ > Pr(t, k+1), & if\ k \geq \mu. \end{cases}$$

2. $\arg\max_{j=1}^{|T(t)|+1} Pr(t, j) = \mu + 1$, *where* $\mu = \sum_{t' \in T(t)} Pr(t')$.

6.2 A general stopping condition

Corollary 5 shows that, given a tuple t and its compressed dominant set $T(t)$, the most likely 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 9 (A General Stopping Condition) *Given a top-k query $Q^k(f)$ and probability threshold p, for a tuple $t \in T$, let $\mu = \sum_{t' \in T(t)} Pr(t')$. Then, $Pr^k(t) < p$ if $\mu \geq k + \ln \frac{1}{p} + \sqrt{\ln^2 \frac{1}{p} + 2k \ln \frac{1}{p}}$.*

Proof To prove Theorem 9, we need Theorem 4.2 in [31]. □

Lemma 1 (Chernoff Bound of Poisson Trials [31])*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*

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

As discussed in Sect. 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

$$k \leq (1-\epsilon)\mu \tag{1}$$

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

$$e^{-\frac{\mu\epsilon^2}{2}} \leq p \tag{2}$$

Combining inequality 1 and 2, we get $2 \ln \frac{1}{p} \leq \mu(1-\frac{k}{\mu})^2$. The inequality in Theorem 9 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 9 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 9.

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 $\mu \geq 117$.

In the experiments, we show in Fig. 6 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 [19].

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 [8] showed that the quality of the approximation has the upper bound $\sup_{0 \leq l \leq n} \left| \sum_{k=0}^l Pr(X = k) - \sum_{k=0}^l f(k, \mu) \right| \leq 9 \max_i \{p_i\}$.

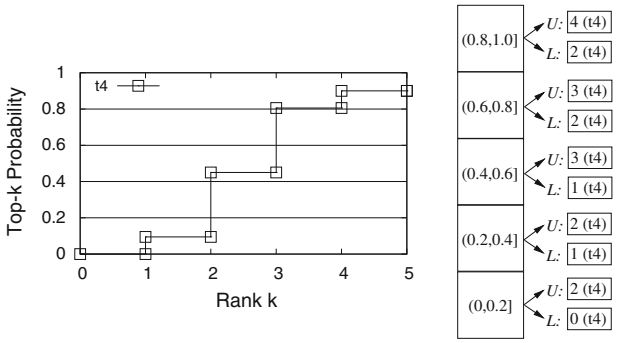
The above upper bound depends on only the maximum success probability in the Poisson trials. In the worst case where $\max_i \{p_i\} = 1$, the error bound is very loose. However, our experimental results (Fig. 8) show that the Poisson approximation method achieves very good approximation quality in practice.

To use Poisson approximation to evaluate a top-k query $Q^k(f)$, we scan the tuples in 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 9 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 $O(|\mathcal{R}| + 1)$, where $\mathcal{R}$ 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.



(a) Top-k probabilities of tuple t_4. **(b)** Index entries.

Fig. 3 The index entries in *PRist* for tuple t_4

7 Online query answering

Since probabilistic ranking queries involve several parameters, a user may be interested in how query results change as parameters vary. To support the interactive analysis, online query answering is highly desirable. In this section, we develop *PRist+* (for probabilistic ranking lists), an index for online answering probabilistic ranking queries on uncertain data, which is compact in space and efficient in construction.

7.1 The *PRist* index

To answer probabilistic ranking queries, for a tuple $t \in T$, a rank parameter k and a probability threshold p, we often need to conduct the following two types of checking operations.

- *Top-k probability checking*: is the top-k probability of t at least p?
- *p-rank checking*: is the p-rank of t at most k?

To support online query answering, we need to index the top-k probabilities and the p-ranks of tuples so that the checking operations can be conducted efficiently. One critical observation is that, for a tuple, the top-k probabilities and the p-ranks can be derived from each other. We propose *PRist*, a list of probability intervals, to store the rank information for tuples.

Example 7 (Indexing top-k probabilities) Consider the uncertain tuples in Table 3. Suppose all tuples are independent. Figure 3a shows the top-k probabilities of tuple t_4 with respect to different values of k. Interestingly, it can also be used to retrieve the p-rank of t_4: for a given probability p, we can draw a horizontal line for top-k probability p, and then check where the horizontal line cuts the curve in Fig. 3a. The point right below the horizontal line gives the answer k.

Storing the top-k probabilities for all possible k can be costly. To save space, we divide the domain of top-k probabilities (0, 1] into h *prob-intervals*.

In Fig. 3a, we partition the probability range (0,1] to 5 prob-intervals: (0,0.2], (0.2,0.4], (0.4,0.6], (0.6,0.8], and (0.8,1.0]. For each interval, we record for each tuple a lower bound and an upper bound of the ranks whose corresponding top-k probabilities lie in the prob-interval. The lower bounds and upper bounds are stored in an L-list and a U-list, respectively. Figure 4a shows the constructed index for the tuples in Table 3.

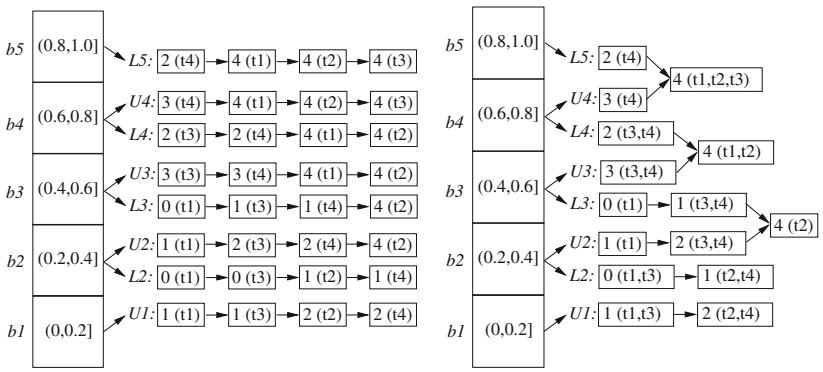
Formally, given a set of uncertain tuples T and a *granularity parameter* $h > 0$, a *PRist* index for T contains a set of *prob-intervals* $\{b_1, \ldots, b_h\}$, where $b_i = (\frac{i-1}{h}, \frac{i}{h}]$ $(1 \leq i \leq h)$.

Each prob-interval b_i $(2 \leq i \leq h-1)$ is associated with two lists: a U-list and an L-list.

An entry in the U-list of b_i corresponds to a tuple t and consists of two items: the tuple id t and an *upper rank* of t in b_i, denoted by $t.U_i$, such that one of the following holds: (1) $Pr^{t.U_i}(t) > \frac{i}{h}$ and $Pr^{t.U_i-1} \leq \frac{i}{h}$ when $Pr^m(t) > \frac{i}{h}$; or (2) $t.U_i = m$ when $Pr^m(t) \leq \frac{i}{h}$. Each tuple $t \in T$ has an entry in the U-list. All entries in the U-list are sorted in ascending order of the upper ranks.

An entry in the L-list of b_i corresponds to a tuple t and consists of two items: the tuple id t and a *lower rank* of t in b_i, denoted by $t.L_i$, such that one of the following holds: (1) $Pr^{t.L_i}(t) \leq \frac{i-1}{h}$ and $Pr^{t.L_i+1}(t) > \frac{i-1}{h}$ when $Pr^m(t) > \frac{i-1}{h}$; or (2) $t.L_i = m$ when $Pr^m(t) \leq \frac{i-1}{h}$. Each tuple $t \in T$ has an entry in the L-list. All entries in the L-list are sorted in ascending order of the lower ranks.

The space cost of a *PRist* is $O(2hn)$, where n is the number of tuples and h is the number of prob-intervals. To reduce the space cost of *PRist*, if an entry $(o, rank)$ appears in the U-list of prob-interval b_i and the L-list of prob-interval b_{i+1} $(1 \leq i < h)$, we can let the two lists share the entry. Moreover, if multiple entries in a list have the same rank, we can compress those entries into one which carries one rank and multiple tuple ids. A *compressed PRist* index is shown in Fig. 4b.



(a) The *PRist* index. **(b)** The compressed *PRist* index.

Fig. 4 A *PRist* index for the uncertain tuples in Table 3

To construct a *PRist* index, we compute the top-k probabilities for all tuples ($1 \leq k \leq m$) in T, where m is the number of rules in T. This takes $O(m^2 n)$ time. Then, the U-lists and L-lists for prob-intervals can be constructed by scanning the tuples and their top-k probabilities. The overall time complexity of the basic construction algorithm is $O(m^2 n + mn + 2hn \log n) = O(m^2 n + 2hn \log n)$.

7.2 *PRist+* and a fast construction algorithm

To reduce the index construction time of *PRist*, we bound top-k probabilities using the binomial distribution. Without loss of generality, we assume that the membership probability of any tuple or rule-tuple in T is smaller than 1.

Theorem 10 (Bounding the probability) *For a tuple $t \in T$, let $T(t)$ be the compressed dominant set of t. Then,*

$$F(k, N, p_{\max}) \leq \sum_{0 \leq j \leq k} Pr(T(t), j) \leq F(k, N, p_{\min}) \quad (3)$$

where $p_{\max}$ and $p_{\min}$ are the greatest and the smallest probabilities of the tuples/rule-tuples in $T(t)$ ($0 < p_{\min} \leq p_{\max} < 1$), N is the number of tuples/rule-tuples in $T(t)$, and F is the cumulative distribution function of the binomial distribution.

Proof We first prove the left side of inequality 3. For a tuple set S, let $Pr(S, \leq k)$ denote $\sum_{0 \leq j \leq k} Pr(S, j)$. For any tuple $t' \in T(t)$, $Pr(t') \leq p_{\max}$.

Consider tuple set $S = T(t) - \{t'\}$ and $T'(t) = S + t_{\max}$ where $Pr(t_{\max}) = p_{\max}$. From Theorem 2

$$\begin{aligned} Pr(T(t), \leq k) &= Pr(t')Pr(S, \leq k-1) \\ &\quad + (1 - Pr(t'))Pr(S, \leq k); \text{ and} \\ Pr(T'(t), \leq k) &= Pr(t_{\max})Pr(S, \leq k-1) \\ &\quad + (1 - Pr(t_{\max}))Pr(S, \leq k). \end{aligned}$$

Then,

$$\begin{aligned} &Pr(T(t), \leq k) - Pr(T'(t), \leq k) \\ &= [Pr(t') - Pr(t_{\max})] \times [Pr(S, \leq k-1) - Pr(S, \leq k)] \end{aligned}$$

Since $Pr(t') \leq Pr(t_{\max})$ and $Pr(S, \leq k-1) \leq Pr(S, \leq k)$, we have $Pr(T(t), \leq k) \geq Pr(T'(t), \leq k)$.

By replacing each tuple/rule-tuple in $T(t)$ with $t_{\max}$, we obtain a set of tuples with the same probability $p_{\max}$, whose subset probabilities follows the binomial distribution $F(k, N, p_{\max})$. Thus, the left side of Inequality 3 is proved.

The right side of Inequality 3 can be proved similarly. □

Moreover, Hoeffding [20] gave the following bound.

Theorem 11 (Extrema [20]) *For a tuple $t \in T$ and its compressed dominant set $T(t)$, let $\mu = \sum_{t' \in T(t)} Pr(t' \prec_f t)$. Then,*

1. $\sum_{j=0}^{k} Pr(T(t), j) \leq F(k, N, \frac{\mu}{N})$ *when* $0 \leq k \leq \mu - 1$; *and*
2. $\sum_{j=0}^{k} Pr(T(t), j) \geq F(k, N, \frac{\mu}{N})$ *when* $\mu \leq k \leq N$,

where N is the number of tuples and rule-tuples in $T(t)$, and F is the cumulative distribution function of the binomial distribution.

Based on Theorems 10 and 11, we derive the following bound for the top-k probability of tuple $t \in T$.

Theorem 12 (Bounds of top-k probabilities) *For a tuple $t \in T$, the top-k probability of t satisfies*

1. $Pr(t)F(k-1, N, p_{\max}) \leq Pr^k(t) \leq Pr(t)F(k-1, N, \frac{\mu}{N})$ *for* $1 \leq k \leq \mu$;
2. $Pr(t)F(k-1, N, \frac{\mu}{N}) \leq Pr^k(t) \leq Pr(t)F(k-1, N, p_{\min})$ *for* $\mu + 1 \leq k \leq N + 1$.

Proof The conclusion follows from Theorems 10 and 11 directly. □

Since the cumulative probability distribution of the binomial distribution is easier to calculate than top-k probabilities, we propose *PRist+*, a variant of *PRist* using the binomial distribution bounding technique.

The only difference between *PRist+* and *PRist* is the upper and lower ranks in the U-lists and L-lists. In *PRist+*, we compute the upper and lower bounds of the top-k probabilities of tuples using the binomial distributions. Then, the upper and lower ranks are derived from the upper and lower bounds of the top-k probabilities.

Take the U-list in prob-interval b_i as an example. In *PRist*, an entry in the U-list consists of the tuple id t and the upper rank $t.U_i$, such that $Pr^{t.U_i}(t) > \frac{i}{h}$ (if $Pr^m(t) > \frac{i}{h}$). Once the top-i probabilities of all ranks $1 \leq i \leq m$ for t are computed, $t.U_i$ can be obtained by one scan.

In *PRist+*, we store the upper rank $t.U_i$ as the smallest rank x such that the lower bound of $Pr^x(t)$ is greater than $\frac{i}{h}$. Following with Theorem 12, the upper rank can be calculated by $x = F^{-1}(\frac{i}{h \cdot Pr(t)}, N, \frac{\mu}{N}) + 1$ or $x = F^{-1}(\frac{i}{h \cdot Pr(t)}, N, p_{\max}) + 1$, where F^{-1} is the binomial inverse cumulative distribution function. The lower rank of t can be obtained similarly using Theorem 12.

Computing the upper and lower ranks for t in b_i requires $O(1)$ time. Thus, the overall complexity of computing the upper and lower ranks of all tuples in all prob-intervals is $O(2hn)$, where n is the total number of tuples. The complexity of sorting the bound lists is $O(2hn \log n)$. The overall time complexity of constructing a *PRist+* index is $O(2hn + 2hn \log n) = O(hn \log n)$.

Clearly, the construction time of *PRist+* is much lower than *PRist*. The tradeoff is that the bounds of ranks in *PRist+* are slightly looser than the rank bounds in *PRist*. The looser rank bounds in *PRist+* do not affect the accuracy of the answers. They only make a very minor difference in query answering time in our experiments (as shown in Fig. 13). Therefore, in real applications, *PRist+* is often more preferable than *PRist*.

7.3 Query evaluation based on *PRist*

In this section, we first discuss the evaluation of PT-k queries based on *PRist*. Then, we briefly discuss how other queries can be answered.

7.3.1 Answering PT-k queries

Example 8 (Answering PT-k queries) Consider the uncertain tuples indexed in Fig. 4 again, and a PT-k query with $k = 3$ and $p = 0.45$.

To find the tuples satisfying the query, we only need to look at the prob-interval containing $p = 0.45$, which is $b_3 = (0.4, 0.6]$. In the U-list of b_3, we find that $t_3.U_3 = 3$ and $t_4.U_3 = 3$, which means that $Pr^3(t_3) > 0.6$ and $Pr^3(t_4) > 0.6$. Therefore, t_3 and t_4 can be added into the answer set without calculating their exact top-k probabilities. In the L-list of b_3, we find that $t_2.L_3 = 4$, which means $Pr^4(t_2) \leq 0.4$. Therefore, t_2 can be pruned.

Thus, only the top-3 probability of t_1 needs to be calculated in order to further verify if t_1 is an answer to the query. Since $Pr^3(t_1) = 0.5$, it can be added into the answer set. The final answer is $\{t_1, t_3, t_4\}$.

Generally, a PT-k query can be evaluated following three steps.

First, we use Corollary 6 to determine whether the top-k probability of t lies in b_i.

Corollary 6 (Bounding top-k probabilities) *Let T be a set of uncertain tuples indexed by PRist+ with granularity parameter h. For a tuple $t \in T$ and a positive integer k, if b_i $(1 \leq i \leq h)$ is the prob-interval such that $t.L_i < k < t.U_i$, then $\frac{i-1}{h} < Pr^k(t) \leq \frac{i}{h}$.*

Proof According to the definition of $PRist+$, we have $Pr^{t.L_i}(t) \leq \frac{i-1}{h}$ and $Pr^{t.L_i+1}(t) > \frac{i-1}{h}$. Since $k > t.L_i$, $Pr^k(t) \geq Pr^{t.L_i+1}(t) > \frac{i-1}{h}$. On the other hand, $Pr^{t.U_i}(t) > \frac{i}{h}$ and $Pr^{t.U_i-1}(t) \leq \frac{i}{h}$. Since $k < t.U_i$, we have $Pr^k(t) \leq Pr^{t.U_i-1}(t) \leq \frac{i}{h}$. □

Second, a tuple may be pruned or validated by checking its lower rank L_i or upper rank U_i in the prob-interval containing the probability threshold, as stated in Theorem 13.

Theorem 13 (Answering PT-k queries) *Let T be a set of uncertain tuples indexed by PRist+ with granularity parameter h. For a tuple $t \in T$ and a PT-k query Q_f^k with probability threshold $p \in b_i = (\frac{i-1}{h}, \frac{i}{h}]$:*

1. *Pruning: if $t.L_i > k$, then $Pr^k(t) < p$;*
2. *Validating: if $t.U_i \leq k$, then $Pr^k(t) > p$.*

Proof The top-k probability distribution of a tuple t increases monotonically with respect to k. Therefore, the conclusions hold. □

Last, we only need to compute the exact top-k probabilities for those tuples that cannot be validated or pruned by Theorem 13.

In the U-list of the prob-interval containing the probability threshold, finding the tuples whose upper ranks are less than or equal to k requires $O(\log n)$ time, where n is the number of tuples in T. Similarly, in the L-list, finding the tuples whose lower ranks are larger than k also takes $O(\log n)$ time. Let d be the number of tuples that cannot be pruned or validated. Computing the top-k probabilities of those tuples requires $O(kmd)$ time, where m is the number of rules in T.

7.3.2 Answering Top-(k, l) and Top-(p, l) queries

To answer a top-(k, l) query Q, we want to scan the tuples in the descending order of their top-k probabilities. However,

PRist+ does not store any exact top-k probabilities. We scan the prob-intervals in the top-down manner instead. For each prob-interval, we retrieve the tuples whose top-k probabilities lie in the prob-interval. Obviously, for two prob-intervals b_i and b_j $(i > j)$, the top-k probabilities falling in b_i is always greater than the top-k probabilities in b_j.

To answer a top-(p, l) query, we only need to check the prob-intervals containing p. Let b_i be the prob-interval containing p. We use the l-th rank k in list $b_i.U$ as a pruning condition. Any tuple t' whose lower rank in b_i is at least k can be pruned. Moreover, for any tuple t, if there are fewer than l tuples whose lower ranks in b_i is smaller than the upper rank of t in b_i, then t can be validated.

8 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.

8.1 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, 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 and number of days drifted. Among them, the number of days drifted 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 $R : t_{r_1} \oplus \cdots \oplus t_{r_m}$, $Pr(R)$ 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 $Pr(t_{r_l}) = \frac{conf(t_{r_l})}{\sum_{1 \le i \le m} conf(t_{r_i})} Pr(R)$ $(1 \le l \le m)$, 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 (Table 5).

8.1.1 Comparing PT-k queries, U-Topk queries and U-KRanks Queries

We applied a *PT-k query*, a *U-TopK query* and a *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 records $\{R1, R2, R3, R4, R5, R6, R9, R10, R11, R14\}$. 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 shown in Table 6.

The *PT-k query* captures some important tuples missed by the *U-TopK query* and the *U-KRanks query*. For example, $R4$, $R10$ and $R14$ have high probabilities to be ranked among-10, but they are not captured by the *U-TopK query* or the *U-KRanks query*.

To compare the difference between the ranking results, we compute the *normalized Kendall distance* [18], as suggested in [27]. Given two ranked list $\mathcal{R}_1$ and $\mathcal{R}_2$, let $\mathcal{K}_1$ and $\mathcal{K}_2$ be the corresponding top-k query results, the normalized Kendall distance is defined as:

$$dis(\mathcal{K}_1, \mathcal{K}_2) = \frac{1}{k^2} \sum_{(i_1,i_2)\in P(\mathcal{K}_1,\mathcal{K}_2)} K(i, j)$$

where $P(\mathcal{K}_1, \mathcal{K}_2)$ is the set of all unordered pairs in $\mathcal{K}_1$ and $\mathcal{K}_2$. $K(i, j) = 1$ if i and j are in opposite order in $\mathcal{R}_1$ and $\mathcal{R}_2$; $K(i, j) = 0$ otherwise. A larger $dis(\mathcal{K}_1, \mathcal{K}_2)$ indicates more disagreement between the two answers. Let $\mathcal{K}_{PT-k}$, $\mathcal{K}_{U-TopK}$ and $\mathcal{K}_{U-KRanks}$ be the results returned by the *PT-k query*, the *U-TopK query* and the *U-KRanks query*, respectively. Then, $dis(\mathcal{K}_{PT-k}, \mathcal{K}_{U-TopK}) = 0.19$, $dis(\mathcal{K}_{PT-k}, \mathcal{K}_{U-KRanks}) = 0.31$, and $dis(\mathcal{K}_{U-TopK}, \mathcal{K}_{U-KRanks}) = 0.31$.

8.1.2 Answering top-(k, l) queries and top-(p, l) queries

Moreover, We conducted top-(k, l) queries, PT-k queries and top-(p, l) queries on the database.

Table 5 Some tuples in the IIP Iceberg Sightings Database 2006

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 The answers to the U-K Ranks query

Rank	1	2	3	4	5	6	7	8	9	10
Tuple	$R1$	$R2$	$R3$	$R5$	$R6$	$R9$	$R9$	$R11$	$R11$	$R18$
$Pr(t, j)$	0.8	0.64	0.512	0.348	0.328	0.258	0.224	0.234	0.158	0.163

Table 7 Results of top-(k, l) queries on the IIP iceberg sighting database ($l = 10$)

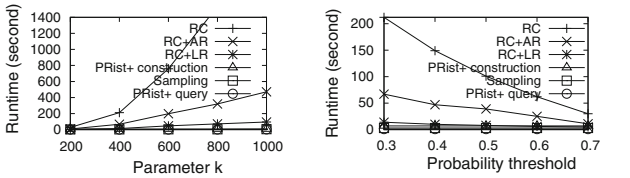
$k = 5$			$k = 20$		
RID	Top-5 prob.	# of days drifted	RID	Top-20 prob.	# of days drifted
$R1$	0.8	435.8	$R1$	0.8	435.8
$R2$	0.8	341.7	$R2$	0.8	341.7
$R3$	0.8	335.7	$R3$	0.8	335.7
$R5$	0.8	284.7	$R5$	0.8	284.7
$R6$	0.61	266.8	$R6$	0.8	266.8
$R4$	0.6	323.9	$R9$	0.8	233.6
$R9$	0.22	233.6	$R11$	0.8	232.6
$R7$	0.17	259.5	$R18$	0.8	229.3
$R10$	0.09	233.3	$R23$	0.79	227.2
$R8$	0.05	240.4	$R33$	0.75	222.2

Table 8 Results of top-(p, l) queries on the IIP iceberg sighting database ($l = 10$)

$p = 0.5$			$p = 0.7$		
RID	0.5-rank	# of days drifted	RID	0.7-rank	# of days drifted
$R1$	2	435.8	$R1$	2	435.8
$R2$	2	341.7	$R2$	2	341.7
$R3$	3	335.7	$R3$	3	335.7
$R4$	4	323.9	$R5$	5	284.7
$R5$	4	284.7	$R6$	6	266.8
$R6$	5	266.8	$R9$	7	233.6
$R9$	7	233.6	$R11$	9	232.6
$R10$	8	233.3	$R10$	10	233.3
$R11$	8	232.6	$R18$	13	229.3
$R14$	10	231.1	$R23$	15	227.2

Table 7 shows the results of a top-(5, 10) query and a top-(20, 10) query. Some records returned by the top-(5, 10) are not in the results of the top-(20, 10) query, such as $R4$, $R7$, $R10$ and $R8$. Moreover, $R11$, $R18$, $R23$ and $R33$ are returned by the top-(20, 10) query but are not in the results of the top-(5, 10) query. Comparing two rank parameters k_1 and k_2 ($k_1 < k_2$), the records only appearing in the answers to the top-(k_1, l) query may have higher scores and lower probabilities than the records only appearing in the answers to the top-(k_2, l) query.

The results to a top-(0.5,10) query and a top-(0.7,10) query are listed in Table 8. By varying p in a top-(p, l) query, we can see the tradeoff between the confidence and the highest ranks a tuple can get with the confidence.



(a) Runtime vs. k. **(b)** Runtime vs. probability threshold.

Fig. 5 Efficiency on real data sets

8.1.3 Efficiency on real data sets

We evaluate the efficiency of the query evaluation algorithms on the IIP Iceberg Database. Limited by space, only the experimental results on PT-k queries are shown in this section. A more thorough performance evaluation using synthetic data sets will be presented in Sect. 8.2. For the exact algorithm, we compare three versions: *RC* (using rule-tuple compression and pruning techniques only), *RC+AR* (using aggressive reordering), and *RC+LR* (using lazy reordering). The sampling method uses the two improvements described in Sect. 5. The query evaluation method based on *PRist+* index is denoted by *PRist+ query*.

We first fix the probability threshold to 0.3 and test the efficiency of the query evaluation algorithms on different rank parameter values. Then, we fix the rank parameter to 400 and increase the probability threshold values from 0.3 to 0.7. The results are shown in Fig. 5a and b, respectively. Clearly, the exact algorithm using lazy reordering achieves the best efficiency among the exact algorithms without indexing. The query evaluation based on *PRist+* is the most efficient among all exact algorithms. The runtime increase of the sampling method is very mild when k becomes larger. The runtime of the Poisson approximation-based method is always less than one second, so we omit the curves for the sake of the readability of the figures.

8.2 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: 1,500 exclusive rules and 500 inclusive 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 probability threshold top-k queries with $k = 200$ and $p = 0.3$.

Since ranking queries are extensively supported by modern database management systems, we treat the generation of a ranked list of tuples as a black box, and test our algorithms on top of the ranked list.

First, to evaluate the efficient top-k probability computation techniques, we compare the exact algorithm, the sampling method, and the Poisson approximation-based method for evaluating PT-k queries. The experimental results for top-(k, l) queries and top-(p, l) queries are similar to the results for PT-k queries. Therefore, we omit the details.

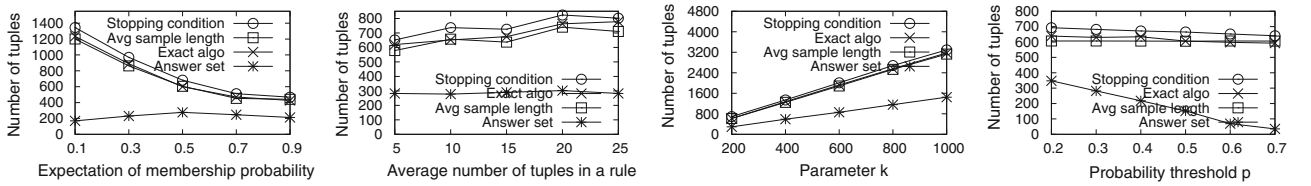
8.2.1 Scan depth

We test the number of tuples scanned by the methods (Fig. 6). 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 Sect. 6.

In Fig. 6a, 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.

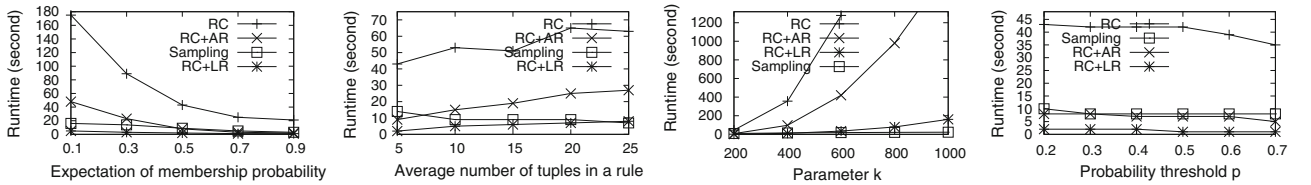
In Fig. 6b, when the rule complexity increases, more tuples are involved in a rule. The average membership probability of those tuples decreases, and thus more tuples need to be scanned to answer the query. In Fig. 6c, both the scan depth and the answer set size increase linearly when k increases, which is intuitive. In Fig. 6d, 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 Sect. 4.2, a tuple t failing the probability threshold still has to be retrieved if some tuples ranked lower than t may satisfy the threshold.

Figure 6 verifies the effectiveness of the pruning techniques discussed in Sect. 4.2. 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



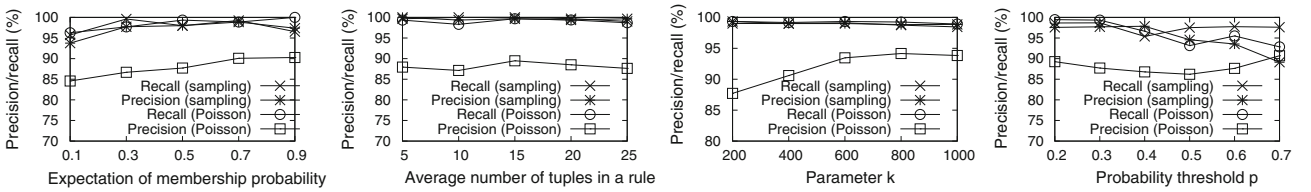
(a) Scan depth vs. probability. **(b)** Scan depth vs. rule complexity. **(c)** Scan depth vs. k. **(d)** Scan depth vs. probability threshold.

Fig. 6 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.

Fig. 7 Efficiency (same settings as in Fig. 6)



(a) Quality vs. probability. **(b)** Quality vs. rule complexity. **(c)** Quality vs. k. **(d)** Quality vs. probability threshold.

Fig. 8 The approximation quality of the sampling method and the Poisson approximation-based method

stopping condition is close to the real stopping point, which shows the effectiveness of the stopping condition.

8.2.2 Efficiency and approximation quality

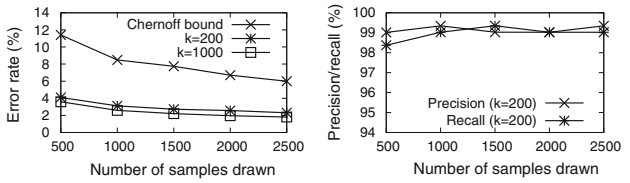
Figure 7 compares the runtime of the three versions of the exact algorithm and the sampling algorithm with respect to the four aspects tested in Fig. 6. The runtime of the Poisson approximation-based method is always less than one second, so we omit it in Fig. 7 for the sake of the readability of the 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 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 8 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.

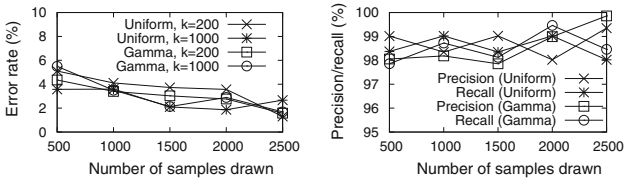
The recall of the Poisson approximation-based method increases significantly when the query parameter k increases. As indicated in [19], the Poisson distribution approximates the Poisson binomial distribution well when the number of Poisson trials is large. When the parameter k increases, more tuples are read before the stopping condition is satisfied. Thus, the Poisson approximation-based method provides better approximation for the top-k probability values.

Figure 9a 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



(a) Error rate vs. sample size. **(b)** Precision/recall vs. sample size.

Fig. 9 The approximation quality of the sampling method



(a) Error rate vs. sample size. **(b)** Precision/recall vs. sample size.

Fig. 10 The approximation quality of the sampling method versus data distributions

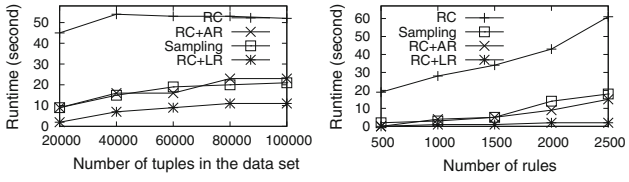
probability estimated by the sampling method is $\widehat{Pr}^k(t)$, the average error rate is defined as

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

For reference, we also plot the error bound calculated from the Chernoff-Hoeffding bound [3] 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, Fig. 9b 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 10 shows the approximation quality of the sampling method with respect to different probability distributions of tuples. One data set contains the tuples whose membership probability follows the uniform distribution between [0.01, 0.99], whose mean is 0.5. The other data set contains the tuples whose membership probability follows the Gamma distribution $\Gamma\left(\frac{\mu}{\theta}, \theta\right)$, where $\mu = 0.5$ and $\theta = 0.2$. The approximation quality is very similar to that on the data set with the Normal distribution, since the sampling method does not make any assumption on the tuples membership probability distribution.



(a) Runtime vs. cardinality. **(b)** Runtime vs. number of rules.

Fig. 11 Scalability

8.2.3 *Scalability*

Last, Fig. 11 shows the scalability of the exact algorithm and the sampling algorithm. In Fig. 11a, we vary the number of tuples from 20,000 to 1,00,000, 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.

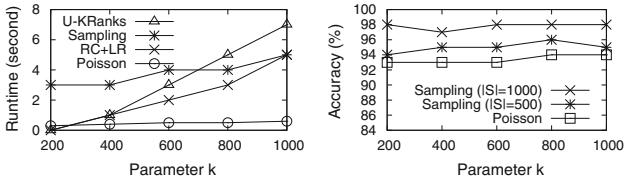
In Fig. 11b, we fix the number of tuples to 20,000, and vary the number of rules from 500 to 2,500. The runtime of the algorithms increases since more rules lead to smaller tuple probabilities and more scans tuples back and forth in the span of rules. However, the reordering techniques can handle the 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.

8.3 Answering U-KRanks queries

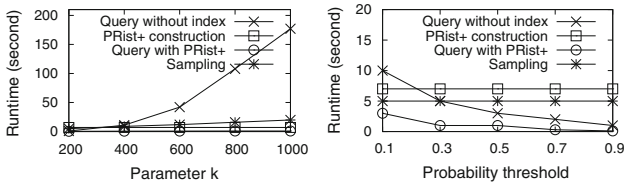
In [39], Soliman et al. propose a U-KRanks query that finds the tuple of the highest probability at each ranking position. Simultaneously with our study, Yi et al. [42,43] proposed efficient algorithms to answer U-KRanks queries (denoted by U-KRanks Algo in this section). The U-KRanks Algo also uses the Poisson binomial recurrence [24] which is used in our exact algorithm (see Theorem 2 in Sect. 3.1). Moreover, the U-KRanks Algo handles generation rules using the similar techniques as our rule-tuple compression discussed in Sect. 3.2.1.

Interestingly, the query evaluation techniques developed in this paper can be used to answer U-KRanks queries with minor changes. We scan the ranked list of tuples in the ranking order and compute the position probability of each tuple. For each rank i ($1 \leq i \leq k$), the tuple t with the largest position probability $Pr(t, i)$ is stored and updated during the scan. The prefix-sharing technique can be used to reduce the computational cost. The algorithm stops when for



(a) Runtime vs. k. **(b)** Accuracy vs. k.

Fig. 12 Performance on U-KRanks queries



(a) Runtime vs. k. **(b)** Runtime vs. p.

Fig. 14 Runtime of PT-k queries

any remaining tuple t and each rank i, the subset probability $Pr(S_t, i-1)$ is already smaller than the largest position probability at rank i ($1 \leq i \leq k$). Similarly, the sampling algorithm and the Poisson approximation based method can be adopted to estimate the position probability of each tuple, and thus find the answers to U-KRanks queries.

We compare our exact algorithm ($RC+LR$), the sampling method, and the Poisson approximation-based method with the U-KRanks Algo. The source code of U-KRanks Algo and the data sets are downloaded from http://www.cs.fsu.edu/~lifeifei/utopk/. As shown in Fig. 12a, our exact algorithm achieves better efficiency than the U-KRanks Algo. This is because our exact algorithm adopts the prefix sharing technique discussed in Sect. 4.1, which reuses the subset probability computation for different tuples and thus reduces the computational cost.

Figure 12b shows the approximation quality of the sampling algorithm and the Poisson approximation-based algorithm. The approximation quality is measured by

$$\frac{\sum_{1 \leq i \leq k} Correct(i)}{k} \times 100\%$$

where $Correct(i) = 1$ if the answers at rank i returned by the exact algorithm and that returned by the approximation algorithm are identical. Both algorithms have high approximation quality.

8.3.1 Online query answering

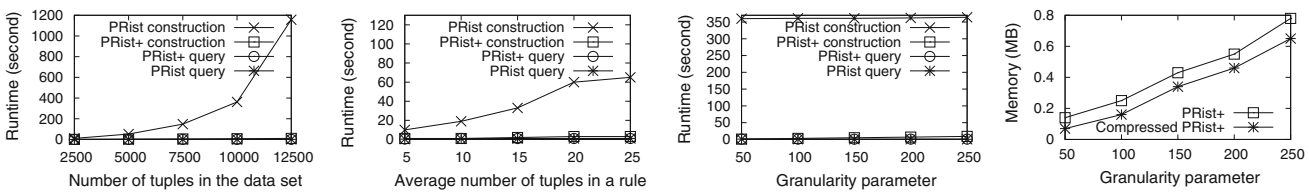
We evaluate the performance of the *PRist* and *PRist+* indices in answering PT-k queries, top-(k, l) queries, and top-(p, l) queries.

Figure 13a–c compare the construction time and average query answering time of *PRist* and *PRist+*. Clearly, *PRist+* can be constructed much more efficiently than *PRist* without sacrificing much efficiency in query answering.

The memory usage of the *PRist+* and compressed *PRist+* is shown in Fig. 13d. *PRist+* uses the compression techniques illustrated in Fig. 4b. The space to fully materialize the top-k probability for each tuple at each rank k is 8.3 MB. It shows that *PRist+* and *PRist* are much more space efficient than the full materialization method. The memory usage of *PRist* is similar to the memory usage of *PRist+*, since they both store the same amount of information and use the same compression techniques. We omit the experimental results for *PRist* for the interest of space.

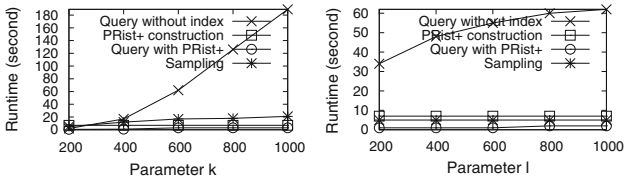
We test the efficiency of the query evaluation methods. Since the query answering time based on *PRist* and *PRist+* is similar, here we only compare the efficiency of the following three methods: the query evaluation methods based on *PRist+*, the query evaluation without index ($RC+LR$), and the sampling method. PT-k queries, top-(k, l) queries and top-(p, l) queries are tested in Figs. 14, 15 and 16, respectively. The construction time of *PRist+* is also plotted in those figures. There are 10,000 tuples, 500 exclusive rules and 500 inclusive rules. The number of tuples in a rule follows the normal distribution $N(5, 2)$. Clearly, the query evaluation methods based on *PRist+* have a dramatic advantage over the query answering methods without the index. Interestingly, even we construct a *PRist+* index on-the-fly to answer a query, in most cases it is still substantially faster than the query evaluation methods without indices.

Last, we test the scalability of the *PRist+* construction method and the query evaluation methods with respect to the number of tuples and the number of tuples in rules, respectively. Figure 17 shows that the methods are scalable and much more efficient than query answering without index.



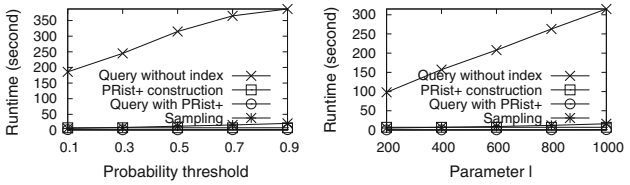
(a) Runtime vs. cardinality. **(b)** Runtime vs. rule complexity. **(c)** Runtime vs. h. **(d)** Memory vs. h.

Fig. 13 The time and memory usage of *PRist* and *PRist+*



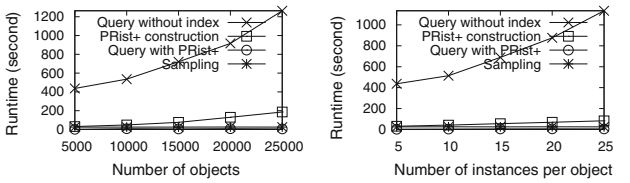
(a) Runtime vs. k. **(b)** Runtime vs. l.

Fig. 15 Runtime of top-(k, l) queries



(a) Runtime vs. p. **(b)** Runtime vs. l.

Fig. 16 Runtime of top-(p, l) queries



(a) Runtime vs. cardinality. **(b)** Runtime vs. rule complexity.

Fig. 17 Scalability

9 Related work

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

9.1 Top-k queries on uncertain data

In [39], 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. Lian and Chen developed the spatial and probabilistic pruning techniques for U-KRanks queries [29]. Simultaneously with our study, Yi et al. [42,43] proposed efficient algorithms to answer U-Top$k$ queries and U-KRanks queries. In Sect. 8.3, we present a detailed discussion and empirical evaluation on the difference between our query answering methods and the algorithm proposed in [42,43].

More recently, Cormode et al. [11] proposed to rank probabilistic tuples by expected ranks. The expected rank of a tuple t is the expectation of t's ranks in all possible worlds. For example, consider a probabilistic table containing three tuples t_1, t_2 and t_3, with membership probabilities 0.6, 1 and 1, respectively. Suppose the ranking order on the three tuples based on their scores is $t_1 \prec t_2 \prec t_3$. There are two possible worlds $W_1 = \{t_1, t_2, t_3\}$ and $W_2 = \{t_2, t_3\}$, with probabilities 0.6 and 0.4, respectively. The expected rank of t_1 is $0 \times 0.6 + 2 \times 0.4 = 0.8$. The expected ranks of t_2 and t_3 are 0.6 and 1.6, respectively. A top-1 query based on expected ranks returns t_2 as the result, since t_2 has the smallest expected rank. However, is t_2 the most likely tuple to be ranked among-1? The top-1 probabilities of t_1, t_2 and t_3 are 0.6, 0.4 and 0, respectively. Clearly, t_1 has the highest probability to be ranked among-1. A top-(k, l) query with $k = 1$ and $l = 1$ returns t_1 as the result. Therefore, ranking by expected ranks cannot capture the semantics of the probabilistic ranking queries discussed in this paper.

In [35], 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. [35] and our study address essentially different queries and applications. Silberstein et al. [37] 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 among-k in any timestep is the greatest. Meanwhile, 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 [44]. Li et al. [26] discussed the problem of ranking distributed probabilistic data. The goal is to minimize the communication cost while retrieve the top-k tuples with expected ranks from distributed probabilistic data sets. In [27], ranking in probabilistic databases is modeled as a multi-criteria optimization problem. A general ranking function of a tuple t is defined as the weighted sum of the position probabilities of t. This allows users to explore the possible ranking functions space. Moreover, how to learn the weight parameters of different position probabilities from user preference data was discussed. Li and Deshpande [28] consider finding the "consensus" answer to queries on uncertain data represented by and/xor trees. An and/xor tree specifies two types of correlations, mutual exclusion and coexistence, among uncertain tuples. Given a query, the "consensus" answer is an answer that is closest in expectation to the answers of the possible worlds. It is shown that, using symmetric difference metric to measure the difference between two top-k answers, the "consensus" answer to a top-k query is the set of k tuples with the largest top-k probabilities, which can be computed by the algorithms discussed in this paper.

The probabilistic threshold top-k queries discussed in this paper 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 [21,22]. However, in this paper, we consider more types of generation

rules in addition to exclusive rules studied in [21,22]. Moreover, we propose top-(p, l) queries that are not considered by any previous studies. We also develop a new index structure to support online ranking query answering on uncertain data. Our query evaluation methods are different from the methods discussed above.

9.2 Probabilistic data and query answering

Modeling and querying uncertain data has been a fast growing research direction [4,16,25,36].

The working model for uncertain data proposed in [36] describes the existence probability of a tuple in an uncertain data set and the constraints (i.e., mutual exclusion and inclusion) on the uncertain tuples, which is essentially the uncertain model adopted in this study. Trio[1] is a database management system that adopts the same uncertain data model. Trio is based on an extended relational model called ULDBs and supports a SQL-based query language.

MayBMS[2] [2] is another probabilistic database management system that adopts the possible worlds semantics. It uses the probabilistic world-set algebra, which consists of the relational algebra operations, tuple confidence computation, and an operation that constructs possible worlds.

Cheng et al. [9] 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 [14] proposed an efficient algorithm to evaluate arbitrary SQL queries on probabilistic databases and rank the results by their probability. Later, they showed in [12] that the conjunctive queries on tuple independent probabilistic databases can be partitioned into "easy" queries (PTIME data complexity) and "hard" queries (the data complexity is $\#P$-Complete). The "easy" queries are the ones having safe query plans. A safe query plan computes the probabilities using the extended relational algebra. MystiQ [7] is a system developed by Sucio et al.[3] that adopts efficient query processing techniques for large probabilistic databases.

9.3 Indexing uncertain data

Tao et al. [40,41] proposed a U-tree index to facilitate probabilistic range queries on uncertain objects represented by multi-dimensional probability density functions. The major advantage of U-tree is to reduce the multidimensional probability calculation to one-dimensional probability distribution. U-tree may not help answering probabilistic ranking queries, since the top-k probability distribution is already a single dimensional probability distribution.

Ljosa et al. [30] developed an APLA-tree index for uncertain objects with arbitrary probability distributions using linear approximation. Cheng et al. [10] developed PTI, probability thresholding index, to index uncertain objects with one dimensional uncertain values. Bohm et al. [6] developed an index for uncertain objects whose probability density functions are Gaussian functions. Moreover, Singh et al. [38] extended the inverted index and signature tree to index uncertain categorical data.

Different from the above work, the index structure developed in our study supports probabilistic ranking queries. Therefore, the distributions of ranks are indexed, instead of probability density functions.

Can we treat the set of $(k, Pr^k(t))$ pairs as points in a two-dimensional space and index them using the traditional spatial indices such as KD-trees and MVB-trees? As KD-trees and MVB-trees are designed for indexing a set of independent points, while the top-k probabilities for each tuple follow a discrete probability distribution. Treating those points as independent points may not allow efficient space compression. The space requirement for storing all top-k probabilities is $O(mn)$, where m is the number of rules and n is the number of tuples in the data set.

The *PRist+* index approximates the top-k probabilities based on Theorem 16. The time complexity of constructing a *PRist+* index is $O(hn \log n)$. Moreover, it only stores the top-k probability bounds for h prob-intervals. The space required for *PRist+* is $O(hn)$. Therefore, both the time and the space complexity are greatly reduced. This distinguishes *PRist+* from other existing spatial indices.

10 Conclusions

In this paper, we studied several novel probabilistic ranking 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. Moreover, an index structure is developed to support online evaluation of probabilistic ranking queries.

It is interesting to extend our study to more sophisticated uncertain data models. In addition, different kinds of ranking and preference queries on uncertain data can be considered as future study.

References

1. Abiteboul, S., Kanellakis, P., Grahne, G.: On the representation and querying of sets of possible worlds. In: SIGMOD'87

[1] http://infolab.stanford.edu/trio/.

[2] http://www.cs.cornell.edu/bigreddata/maybms/.

[3] http://www.cs.washington.edu/homes/suciu/project-mystiq.html.

2. Antova, L., Jansen, T., Koch, C., Olteanu, D.: Fast and simple relational processing of uncertain data. In: ICDE'08
3. Angluin, D., Valiant, L.G.: Fast probabilistic algorithms for hamiltonian circuits and matchings. In: STOC'77
4. Antova, L., Koch, C., Olteanu, D.: 10^{10^6} worlds and beyond: efficient representation and processing of incomplete information. In: ICDE'07
5. Benjelloun, O., Sarma, A.D., Halevy, A., Widom, J.: Uldbs: databases with uncertainty and lineage. In: VLDB'06
6. Böhm, C., Pryakhin, A., Schubert, M.: The gauss-tree: efficient object identification in databases of probabilistic feature vectors. In: ICDE'06
7. Boulos, J., Dalvi, N., Mandhani, B., Mathur, S., Re, C., Suciu, D.: MYSTIQ: a system for finding more answers by using probabilities. In: SIGMOD'05
8. Cam, L.L.: An approximation theorem for poisson binomial distribution. Pac. J. Math. **10**, 1181–1197 (1960)
9. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: SIGMOD'03
10. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.S.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: VLDB'04
11. Cormode, G., Li, F., Yi, K.: Semantics of ranking queries for probabilistic data and expected ranks. In: ICDE'09
12. Dalvi, N., Suciu, D.: The dichotomy of conjunctive queries on probabilistic structures. In: PODS'07
13. Dalvi, N., Suciu, D.: Management of probabilistic data: foundations and challenges. In: PODS'07
14. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB'04
15. Das, G., Gunopulos, D., Koudas, N., Tsirogiannis, D.: Answering top-k queries using views. In: VLDB'06
16. Dey, D., Sarkar, S.: A probabilistic relational model and algebra. ACM Trans. Database Syst. **21**(3), 339–369 (1996)
17. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS'01
18. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. In: SODA'03
19. Hodges, J.L., Cam, J.L.L.: The poisson approximation to the poisson binomial distribution. Ann. Math. Stat. **31**(3), 737–740 (1960)
20. Hoeffding, W.: On the distribution of the number of successes in independent trials. Ann. Math. Stat. **27**, 713–721 (1956)
21. Hua, M., Pei, J., Zhang, W., Lin, X.: Efficiently answering probabilistic threshold top-k queries on uncertain data (extended abstract). In: ICDE'08
22. Hua, M., Pei, J., Zhang, W., Lin, X.: Ranking queries on uncertain data: A probabilistic threshold approach. In: SIGMOD'08
23. Imielinski, T., Lipski, W. J.: Incomplete information in relational databases. J. ACM **31**(4), 761–791 (1984)
24. Lange, K.: Numerical analysis for statisticians. Stat. comput. (1999)
25. Lee, S.K.: Imprecise and uncertain information in databases: an evidential approach. In: ICDE'92
26. Li, F., Yi, K., Jestes, J.: Ranking distributed probabilistic data. In: SIGMOD'09
27. Li, J., Saha, B., Deshpande, A.: A unified approach to ranking in probabilistic databases. In: VLDB'09
28. Li, J., Deshpande, A.: Consensus answers for queries over probabilistic databases. In: PODS'09
29. Lian, X., Chen, L.: Probabilistic ranked queries in uncertain databases. In: EDBT'08
30. Ljosa, V., Singh, A.K.: Apla: Indexing arbitrary probability distributions. In: ICDE'07
31. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, United Kingdom (1995)
32. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding windows. In: SIGMOD'06
33. Nepal, S., Ramakrishna, M.: Query processing issues in image(multimedia) databases. In: ICDE'99
34. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: VLDB '07
35. Ré, C., Dalvi, N., Suciu, D.: Efficient top-*k* query evaluation on probabilistic data. In: ICDE'07
36. Sarma, A.D., Benjelloun, O., Halevy, A., Widom, J.: Working models for uncertain data. In: ICDE'06
37. Silberstein, A.S., Braynard, R., Ellis, C., Munagala, K., Yang, J.: A sampling-based approach to optimizing top-k queries in sensor networks. In: ICDE '06
38. Singh, S., Mayfield, C., Prabhakar, S., Shah, R., Hambrusch, S.: Indexing uncertain categorical data. In: ICDE'07
39. Soliman, M.A., Ilyas, I.F., Chang, K.C.C.: Top-*k* query processing in uncertain databases. In: ICDE'07
40. Tao, Y., Cheng, R., Xiao, X., Ngai, W.K., Kao, B., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: VLDB'05
41. Tao, Y., Xiao, X., Cheng, R.: Range search on multidimensional uncertain data. ACM Trans. Database Syst. **32**(3), 15 (2007)
42. Yi, K., Li, F., Srivastava, D., Kollios, G.: Efficient processing of top-k queries in uncertain databases. In: ICDE'08
43. Yi, K., Li, F., Kollios, G., Srivastava, D.: Efficient processing of Top-k queries in uncertain databases with x-Relations. IEEE Trans. Knowl. Data Eng. **20**(12), 1669–1682 (2008)
44. Zhang, X., Chomicki, J.: Semantics and evaluation of Top-k queries in probabilistic databases. Distrib. Parallel Databases (DAPD) J. (Special Issue on Ranking in Databases), pp. 67–126