Probabilistic Ranking over Relations

Lijun Chang , Jeffrey Xu Yu, Lu Qin
The Chinese University of Hong Kong
Hong Kong, China
{ljchang,yu,lqin}@se.cuhk.edu.hk

Xuemin Lin
The University of New South Wales
Sydney, Australia
lxue@cse.unsw.edu.au

ABSTRACT

Probabilistic top-k ranking queries have been extensively studied due to the fact that data obtained can be uncertain in many real applications. A probabilistic top-k ranking query ranks objects by the interplay of score and probability, with an implicit assumption that both scores based on which objects are ranked and probabilities of the existence of the objects are stored in the same relation. We observe that in general scores and probabilities are highly possible to be stored in different relations, for example, in column-oriented DBMSs and in data warehouses. In this paper we study probabilistic top-k ranking queries when scores and probabilities are stored in different relations. We focus on reducing the join cost in probabilistic top-k ranking. We investigate two probabilistic score functions, discuss the upper/lower bounds in random access and sequential access, and provide insights on the advantages and disadvantages of random/sequential access in terms of upper/lower bounds. We also propose random, sequential, and hybrid algorithms to conduct probabilistic top-k ranking. We conducted extensive performance studies using real and synthetic datasets, and report our findings in this paper.

1. INTRODUCTION

Uncertain data management is an important issue in sensor network, data cleaning, data integration, and market decision making, due to the fact that a large amount of information obtained is either incomplete or uncertain. Several uncertain data models are proposed [2, 12, 4], and probabilistic ranking queries are studied [34, 16, 11, 7, 23, 24] which are based on the interplay of score to be ranked and probability to be observed.

Probabilistic ranking queries are first studied by Soliman et al. under the possible worlds semantics [34], and the efficiency of probabilistic ranking queries are further studied in [37, 16] by utilizing independent and mutually exclusive relationships among tuples under an x-Relation model. Cormode et al. also propose to rank uncertain data based on their expected rank values [11], called expected rank semantic. Li et al. study ranking distributed uncertain data based on the expected rank semantic [23]. The existing approaches assume that both scores based on which objects are ranked and probabilities of the existence of the objects are stored in the same relation.

However, we observe that, in general, scores and probabilities are highly possible to be stored in different relations, e.g. in column-store database [1, 20, 8, 17, 21, 4], data integration [29], and data warehouse [9]. In a column-store database, unlike the tuple-based approach taken in conventional relational DBMSs, information is stored in column relations. For example, one column relation stores object identifier and object scores, and another column relation stores object identifier and object existence probability. It is reported that column-oriented DBMSs can perform much better than conventional relational DBMSs in many real applications such as business intelligence applications [1]. Column storage has been successfully used for many years in OLAP (Online Analytical Processing) [36], and is also adapted to perform OLTP (Online Transactional Processing) recently [28]. As another example, in a data warehouse, data are stored in a fact table and a collection of dimension tables using a star schema. The object identifiers and the probability of the existence of the objects may be stored in the fact table or a dimensional table, whereas the scores based on which users want to rank objects may be stored in another dimension table. In such an environment, it needs to join different relations into one relation to have both score and probability together, and to apply one of the existing approaches to probabilistically rank the objects, which can be costly.

Consider a data warehouse that stores textual information, e.g. reviews, shape, price, weight, about products extracted from online shops and forums. The fact table stores the probability of each fact (e.g. review) to be true, and the shape, price and weight information will be stored in other dimensional tables. In order to analyze such kinds of uncertain information, users may want to rank the facts based on a user-specified score function by combining shape, price, weight, and text information. These scoring attributes and the probability attribute are stored in different tables. Also the users may want to specify selection constraints on the facts that should be ranked, e.g. specify the areas where the products are sold, the countries where the products are made, the time interval the facts are extracted. Users are interested in different portions of the whole data, and also in different ranking criteria. In such cases, it is difficult to materialize data for all possible queries.

In this paper, we study top-k probabilistic ranking queries with joins when scores and probabilities are stored in different relations. To the best of our knowledge, this is the first work in probabilistic ranking under possible worlds semantic that take join issues into consideration. Our work is different from the existing work on rank joins which deal with deterministic data [19, 32]. The main difference is that, for deterministic data, the score of an object can be computed by itself, whereas, for probabilistic data, the probabilistic score of an object cannot be computed by itself, and needs to be computed based on scores and probabilities of other objects.

The main contributions of this work are summarized below. We study top-k probabilistic ranking queries when scores and probabilities are stored in different relations. We investigate two probabilistic score functions, namely, expected rank value [11] and probability of highest ranking [7]. We give upper/lower bounds of such probabilistic score functions in random access and sequential access, and discuss the advantages/disadvantages of random and sequential accesses. We propose new I/O efficient algorithms to find top-k objects with probabilistic ranking functions, using random access, sequential access, and the combination of random and sequential access by taking the advantages from both random/sequential access. We conduct extensive performance studies, and confirm the effectiveness of our approaches.

The remainder of the paper is organized as follows. In Section 2, we review the x-Relation model of uncertain data and probabilistic ranking queries over x-Relation. Section 3 gives out our problem statement. In Section 4, we discuss bounding schema for two top-k probabilistic ranking queries. In Section 5, we propose algorithms to find the top-k answers with respect to the probabilistic ranking function, with random and/or sequential accesses. Experimental studies are reported in Section 6 followed by discussions on related work in Section 7. Finally, Section 8 concludes the paper.

2. PROBABILISTIC RANKING QUERIES

We adopt the x-Relation model [2] to model the uncertain data. An x-Relation $\mathcal{X}$ contains a set of independent x-tuples, i.e. $\mathcal{X} = \{\tau_1, \tau_2, \cdots\}$. By independent, it means that the existence of an x-tuple is independent from the other x-tuples. An x-tuple, τ_j, consists of a set of mutually exclusive tuples (also called alternatives in [16]), and represents a discrete probability distribution of the possible tuples (alternatives) it may take in a randomly instantiated data. In brief, an alternative of an x-tuple, denoted as o_i, has a score, denoted as $score(o_i)$, and a probability, denoted as $p(o_i)$. The probability represents the tuple existence probability over possible instances. By mutually exclusive, it implies that an x-tuple can take at most one alternative in a possible instance.

The x-Relation, $\mathcal{X}$, represents a probability distribution over a set of possible instances $\{I_1, I_2, \cdots\}$. A possible instance, I_i, maintains zero or one alternative for every x-tuple. The probability of an instance, I_i, is the probability that the x-tuples take none/one alternative in that instance,

$$\Pr(I_i) = \prod_{o \in I_i} p(o) \times \prod_{\tau \notin I_i} (1 - \Pr(\tau)) \tag{1}$$

where $\tau \notin I_i$ means x-tuple τ takes no alternative in I_i and $\Pr(\tau) = \sum_{o \in \tau} p(o)$. The set of possible instances with positive probability is called possible worlds of the x-Relation $\mathcal{X}$, denoted as $pwd(\mathcal{X})$.

Below, we discuss ranking based on tuples instead of x-tuples because an x-tuple can take at most one alternative tuple. Assume there is an x-tuple, τ, that has three alternative tuples $\tau = \{o_1, o_3, o_4\}$. Let $p(o_1) = 0.3$, $p(o_3) = 0.4$, and $p(o_4) = 0.1$. It suggests that the x-tuple τ may take either o_1 with probability 0.3, or o_3 with probability 0.4, or o_4 with probability 0.1, or none with probability $1 - 0.3 - 0.4 - 0.1 = 0.2$.

Definition 2.1: (**Rank in a possible instance** I [11]) In a possible instance I, the rank of a tuple $o_i \in I$, $rank_I(o_i)$, is the number of tuples whose score is larger than o_i.

$$rank_I(o_i) = \begin{cases} |\{o_j \in I \mid score(o_j) > score(o_i)\}|, & \text{if } o_i \in I; \\ |I|, & \text{otherwise.} \end{cases}$$

OID	*score*	*prob*
o_1	100	0.3
o_2	95	0.15
o_3	90	0.4
o_4	85	0.1
o_5	80	0.45
o_6	75	0.2
o_7	70	0.2

Table 1: An Example Relation (SP)

rank	tuple	R_E
1	o_3	1.02
2	o_1	1.05
3	o_5	1.17
4	o_2	1.4475
5	o_6	1.56
6	o_7	1.6
7	o_4	1.615

(a) Rank with R_E

rank	tuple	P_{HR}
1	o_1	0.3
2	o_3	0.238
3	o_5	0.144585
4	o_2	0.105
5	o_4	0.0375
6	o_6	0.035343
7	o_7	0.0282744

(b) Rank with P_{HR}

Table 2: Two Rankings

Note that the top tuple has rank 0. □

Definition 2.2: (**Expected rank value** [11]) The expected rank value of a tuple o_i is defined as follows.

$$R_E(o_i) = \sum_{I \in pwd(\mathcal{X})} \Pr(I) \cdot rank_I(o_i)$$

It considers the rank of a tuple as a random variable, and ranks a tuple based on the expected value of the random variable. □

Definition 2.3: (**Probability of highest ranking** [7]) The probability for a tuple o_i to be ranked at the first place is defined as follows.

$$P_{HR}(o_i) = \sum_{\substack{I \in pwd(\mathcal{X}) \\ o_i \in I \\ rank_I(o_i) = 0}} \Pr(I)$$

It ranks tuple, o_i, based on the summation of the probability of the possible instances where o_i appears and is ranked at the first place (rank 0). □

As can be seen from the above, for ranking tuples o_i, the score ($score(o_i)$) and probability ($p(o_i)$) play different roles in the probabilistic ranking. The $score(o_i)$ is used to define the relative rank of tuples in a possible instance, whereas $p(o_i)$ is used to measure the probability in all possible instances. In the following, we discuss probabilistic ranking based on a probabilistic score function that combines both $score(o_i)$ and $p(o_i)$, denoted as $\mathsf{pscore}(o_i)$. Both R_E and P_{HR} are such probabilistic score functions.

Based on the probabilistic score function R_E, a Top-k Expected Rank Value (Top-kERV) query returns top-k tuples with lowest R_E values. Based on the probabilistic score function P_{HR}, a Top-k Probable Highest Ranking (Top-kPHR) query returns top-k tuples with highest P_{HR} values.

Table 1 shows a relation with 7 tuples $\{o_1, o_2, \cdots, o_7\}$. A tuple o_i is associated with a score ($score(o_i)$) and a probability ($p(o_i)$). For example, tuple o_3 has a score value, $score(o_3) = 90$, and a probability $p(o_3) = 0.4$. Assume an x-Relation $\mathcal{X}$ has 7 x-tuples $\{\tau_1, \tau_2, \cdots, \tau_7\}$, and each x-tuple τ_i has only one alternative tuple o_i in Table 1. The ranking based on R_E and P_{HR} are shown in Table 2.

OID	*score*
o_1	100
o_2	95
o_3	90
o_4	85
o_5	80
o_6	75
o_7	70

(a) Relation S

OID	*prob*
o_5	0.45
o_3	0.4
o_1	0.3
o_6	0.2
o_7	0.2
o_2	0.15
o_4	0.1

(b) Relation P

Table 3: Two Relations (S and P)

3. PROBLEM STATEMENT

All the existing work assume that there is an x-Relation $\mathcal{X}$ which contains both score and probability. However, in real applications, the scores and the probabilities may be stored in different relations. As an example, the same information stored in relation SP (Table 1) may be stored in two separated relations, S and P, as shown in Table 3 where $SP = S \bowtie P$.

A naive approach to compute a probabilistic ranking query, when score and probability are not stored in the same relation, is to join the relations followed by applying an existing approach to compute the probabilistic ranking query in the relation. But this naive approach will incur both high computational cost and I/O cost, because it needs to join the whole relations.

Problem Statement: In this paper, we study how to compute a probabilistic ranking query (Top-kERV or Top-kPHR) by reducing the total I/O cost, when score and probability are not stored in the same relation. We aim at computing the top-k tuples by accessing tuples as least as possible.

In the following, we discuss our approaches using the two relations, S and P, as shown in Table 3. We consider two access methods, namely, random access and sequential access.

- For the random access, it assumes that the relation S is sorted in descending order based on the scores. It sequentially accesses the tuples in relation S one-by-one. When it accesses a tuple o_i in relation S in an iteration, it obtains the score value ($score(o_i)$) from the same tuple in relation S, and obtains the probability of the tuple o_i, $p(o_i)$, in relation P using a SQL selection with the same `OID` o_i, which results in a random access of relation P.

- For the sequential access, it assumes that both relation S and relation P are sorted in descending order based on the scores and probabilities, respectively. It sequentially accesses the tuples in relation S and relation P following the descending order of score and probability, respectively. In every iteration, it accesses an additional tuple from relation S and an additional tuple from relation P, respectively. For example, as shown in Table 3, in the first iteration, it accesses the tuple identified by o_1 from S and the tuple identified by o_5 from P; in the second iteration, it accesses the tuple identified by o_2 from S and the tuple identified by o_3 from P.

In summary, we consider that the relation S is accessed sequentially in descending score order, which is the access method used in all the existing algorithms when both scores and probabilities are stored in the same relation [34, 37, 16, 11]. The two access methods, namely random/sequential accesses, are about how to access relation P.

The key issue is how many tuples it needs to access in order to compute Top-kERV/Top-kPHR using random access and sequential access. We show that we do not need to compute the exact pscore values (R_E values and P_{HR} values) with the assistance of upper bounds and lower bounds for pscore values. We can compute Top-kERV/Top-kPHR by its relative orders.

4. BOUNDING RANKING FUNCTIONS

A probabilistic ranking query ranks a tuple, o_i, with a score function $\mathsf{pscore}(o_i)$. It is worth noting that the probabilistic score function $\mathsf{pscore}(o_i)$ is completely different from those score functions that can be evaluated by the tuple in question and is independent from other tuples. In other words, the probabilistic score function $\mathsf{pscore}(o_i)$ needs to be evaluated depending on the $score(o_i)$ and $p(o_i)$ for the tuple o_i itself as well as $score(o_j)$ and $p(o_j)$ for the other tuples o_j. It becomes very important to identify characteristics of the probabilistic score function pscore, especially the monotonicity, upper bounds, and lower bounds.

In this section, first, for simplicity, we focus on independent case such that in an x-Relation $\mathcal{X}$, every x-tuple has only one tuple (alternative). Then we will discuss mutually exclusive case. We assume that all the scores are of different values, it is straightforward to extend to tie scores. We use the notation in a way that the tuples, $o_1, \cdots, o_n$, are in descending score order such that $score(o_i) > score(o_j)$ if $i < j$.

4.1 Ranking Function R_E

Assume the tuples $o_1, \cdots, o_n$ are in descending score order, and they are totally independent. The expected size of possible instances is $E[|I|] = \sum p(o_j)$ for all o_j in relation P, and the probabilistic score function $R_E(o_i)$ can be simplified as follows.

$$\begin{aligned}
& R_E(o_i) \\
= & \sum_{I \in pwd(\mathcal{X})} \Pr(I) \cdot rank_I(o_i) \\
= & \sum_{o_i \in I} \Pr(I) \cdot rank_I(o_i) + \sum_{o_i \notin I} \Pr(I) \cdot |I| \\
= & p(o_i) \cdot \sum_{j=1}^{i-1} p(o_j) + (1 - p(o_i)) \cdot \sum_{j \neq i} p(o_j) \qquad (2) \\
= & p(o_i) \cdot \sum_{j=1}^{i-1} p(o_j) + (1 - p(o_i)) \cdot (E[|I|] - p(o_i)) \\
= & E[|I|] - p(o_i) \cdot (E[|I|] - \sum_{j=1}^{i-1} p(o_j) - p(o_i) + 1)
\end{aligned}$$

The details and correctness of Eq. (2) are given in [11]. Eq. (2) suggests that we cannot simply compute $R_E(o_i)$ even if we have already known its $score(o_i)$ and $p(o_i)$, because it requests us to know $p(o_j)$ for those tuples that $score(o_j) > score(o_i)$. In order to find the top-k tuples without accessing all tuples, we need to bound the R_E value for each seen or unseen tuple. Before discussing bounds, we first prove the monotonicity of R_E below.

Lemma 4.1: *The R_E function, on which the Top-kERV query is based, is a monotone function, i.e. for any two tuples o_i and o_j, if $score(o_i) \geq score(o_j)$ and $p(o_i) \geq p(o_j)$, then $R_E(o_i) \leq R_E(o_j)$ and o_i ranks higher than o_j.* □

Proof Sketch: We simplify $p(o_l)$ as p_l in the following proof. Consider $R_E(o_i) - R_E(o_j)$. We have

$$\begin{aligned}
& R_E(o_i) - R_E(o_j) \\
= & p_i \sum_{l=1}^{i-1} p_l + (1 - p_i)(E[|I|] - p_i) - p_j \sum_{l=1}^{j-1} p_l - (1 - p_j)(E[|I|] - p_j) \\
= & (p_i - p_j) \sum_{l=1}^{i-1} p_l - p_j \sum_{l=i}^{j-1} p_l - E[|I|](p_i - p_j) + (p_i - p_j)(p_i + p_j - 1) \\
= & (p_i - p_j)(\sum_{l=1}^{i} p_l - E[|I|] + p_j - 1) - p_j \sum_{l=i}^{j-1} p_l \\
\leq & 0
\end{aligned}$$

Set	S	P	Type	Bounds for $E[\|I\|] - R_E$	Bounds for P_{HR}
$\mathcal{H}_s^+$	✓	✓	*lower*	$p(o_i) \cdot (E[\|I\|] - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} p(o_j) - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^-}} \bar{p} + 1)$	$p(o_i) \cdot \prod_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} (1 - p(o_j)) \cdot \prod_{\substack{j \le i \\ o_j \in \mathcal{H}_s^-}} (1 - \bar{p})$
$\mathcal{H}_s^+$	✓	✓	*upper*	$p(o_i) \cdot (E[\|I\|] - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} p(o_j) + 1)$	$p(o_i) \cdot \prod_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} (1 - p(o_j))$
$\mathcal{H}_s^-$	✓	×	*upper*	$\bar{p} \cdot (E[\|I\|] - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} p(o_j) + 1)$	$\bar{p} \cdot \prod_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} (1 - p(o_j))$
$\mathcal{H}_{\neg s}$	×	✓	*upper*	$p(o_i) \cdot (E[\|I\|] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) - p(o_i) + 1)$	$p(o_i) \cdot \prod_{o_j \in \mathcal{H}_s^+} (1 - p(o_j))$
U	×	×	*upper*	$\bar{p} \cdot (E[\|I\|] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) + 1)$	$\bar{p} \cdot \prod_{o_j \in \mathcal{H}_s^+} (1 - p(o_j))$

Table 4: $E[|I|] - R_E$ **and** P_{HR} **bounds for tuple** o_i **(independent)**

where the last inequality holds because $\sum_{l=1}^{i} p_l + p_j \le E[|I|]$ and $p_i \ge p_j$. Therefore, $R_E(o_i) \le R_E(o_j)$ and o_i ranks higher than o_j. □

We can bound $R_E(o_i)$, under the random access and sequential access of relation P respectively, where relation S is accessed sequentially in descending score order. We denote the upper and lower bounds as R_E^{up} and R_E^{low}. It is reasonable to assume that $E[|I|]$ is available in advance, because $E[|I|] = \sum p(o_j)$ for all o_j in relation P.

Random access on relation P: For a tuple o_i, we obtain its score $score(o_i)$ when accessing relation S in descending score order and obtain its probability $p(o_i)$ using a SQL selection to access relation P randomly at the same time. Because relation S is sorted in descending score order, we know the probability $p(o_j)$ for all tuples whose scores are larger than that of the tuple o_i in question ($score(o_j) > score(o_i)$).

For each seen tuple o_j, we can compute its exact $R_E(o_j)$ value by Eq. (2). Assume that o_i is the last seen tuple after retrieving the tuples $o_1, \cdots, o_{i-1}$. For the unseen tuples o, we can lower bound $R_E(o)$ by the following equation.

$$\begin{aligned} R_E(o) &= p(o) \cdot \sum_{score(o_j) > score(o)} p(o_j) + (1 - p(o)) \cdot \sum_{o_j \neq o} p(o_j) \\ &\ge p(o) \cdot \sum_{j=1}^{i} p(o_j) + (1 - p(o)) \cdot \sum_{j=1}^{i} p(o_j) \\ &= \sum_{j=1}^{i} p(o_j) \end{aligned} \quad (3)$$

Intuitively, it is lower bounded by the expected size of the possible worlds generated by the tuples $\{o_1, \cdots, o_i\}$. Note that for the Top-kERV query the smaller R_E value the better.

Example 4.1: Consider relation S and relation P in Table 3. Assume the first three tuples in relation S have been retrieved. We get all the scores for o_1, o_2, and o_3, and also get the probability for the three type by random accesses on relation P. The set of seen tuples is $\{o_1(100, 0.3), o_2(95, 0.15), o_3(90, 0.4)\}$, where each entry represents $o_i(score(o_i), p(o_i))$. $E[|I|] = 1.8$. Based on Eq. (2), we have $R_E(o_1) = 1.05$, $R_E(o_2) = 1.4475$, and $R_E(o_3) = 1.02$. The lower bound for any unseen tuple o is $R_E^{low}(o) = 0.3 + 0.15 + 0.4 = 0.85$. □

It is difficult to bound $R_E(o)$ tight, if we do not know all the tuples o_j whose scores are larger ($score(o_j) > score(o)$). In other words, all the unseen tuples may have larger scores or none of them have larger score.

Sequential access on relation P: In this scenario, each time we retrieve one entry from S and P in descending score and probability order respectively. For each seen tuple, we may know its score and/or probability. In other words, we may not know both score and probability for every seen tuple. However, in the sequential access, unlike the random access, we have one additional piece of information, the upper bound of all the unknown probabilities, denoted as $\bar{p}$. It is the last retrieved probability from relation P.

Let $\mathcal{H}_s$ denote the set of seen tuples that we know their scores, and let $\mathcal{H}_{\neg s}$ denote the set of seen tuples that we know their probabilities but do not know their scores. In other words, we know $score(o_i)$ for those tuples $o_i \in \mathcal{H}_s$ and $p(o_i)$ only for those tuples $o_i \in \mathcal{H}_{\neg s}$. Furthermore, the tuples o_i that we know both $score(o_i)$ and $p(o_i)$ are also kept in $\mathcal{H}_s$. In particular, we have $\mathcal{H}_s = \mathcal{H}_s^+ \bigcup \mathcal{H}_s^-$, where $\mathcal{H}_s^+$ contains the tuples that we know both score and probability, and $\mathcal{H}_s^-$ contains the tuples we only know their score. In summary, we need to bound R_E for the tuples in $\mathcal{H}_s^+$, $\mathcal{H}_s^-$, $\mathcal{H}_{\neg s}$, and those tuples we have not seen. For the tuples in $\mathcal{H}_s^+$, we need to get both the lower bound and the upper bound, to find the top-k tuples earlier. For the other tuples, we only need to get its lower bound, because its upper bound can be very loose, and we can not determine any of the other tuples to be in top-k results at this step. If the upper bound R_E of any tuples in $\mathcal{H}_s^+$ is no larger than the lower bound of all the other tuples, then this tuple can be determined in the top-k results. In order to bound R_E, we need the following information.

For the tuples in $\mathcal{H}_s^+$ with known score and probability, we need both lower bound and upper bound of R_E. Consider Eq. (2). When $p(o_i)$ is known, the formula can be simplified to the form of $R_E(o_i) = c \cdot \sum_{j=1}^{i-1} p(o_j) + c'$, where $c > 0$ and c' are constants. The lower bound and upper bound are obtained by replacing those unknown $p(o_j)$'s with 0 and $\bar{p}$ respectively.

For those tuples in $\mathcal{H}_s^-$ with known score only, we need to compute its lower bound. It is lower bounded by $E[|I|] - \bar{p} \cdot (E[|I|] - \sum_{j<i, o_j \in \mathcal{H}_s^+} p(o_j) + 1)$.

For those tuples $o_i \in \mathcal{H}_{\neg s}$, we need its lower bound, which can be obtained in a similar way as discussed above. It is lower bounded by $E[|I|] - p(o_i) \cdot (E[|I|] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) - p(o_i) + 1)$. Similarly, we can lower bound $R_E(o_i)$ for the unseen tuples by $E[|I|] - \bar{p} \cdot (E[|I|] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) + 1)$.

In a summary, the bounds for tuples in different sets are listed in Table 4. Where the `Set` column is the name of the set that the tuple belongs to, and U denotes the set of unseen tuples. The S and P columns means whether we know the score and probability for the tuple respectively. We show bounds for $E[|I|] - R_E(o_i)$ in Table 4. All the lower bounds for $E[|I|] - R_E(o_i)$ become upper bounds for $R_E(o_i)$, and upper bounds for $E[|I|] - R_E(o_i)$ become lower bounds for $R_E(o_i)$.

Example 4.2: Assume that we have retrieved 3 tuples from both relation S and relation P in Table 3, respectively. Then, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4)\}$ and $\mathcal{H}_{\neg s} = \{o_5(-, 0.45)\}$. $\mathcal{H}_s$ can be further partitioned into $\mathcal{H}_s^+ = \{o_1(100, 0.3), o_3(90, 0.4)\}$ and $\mathcal{H}_s^- = \{o_2(95, -)\}$. "-" means the value of that field is unknown. Here, $\bar{p} = p(o_1) = 0.3$, which is the last probability we have seen. $E[|I|] = 1.8$. For tuple o_1, no tuple has a larger score. $p(o_1) = 0.3$, and then $R_E(o_1) = 1.8 - 0.3 \times (1.8 - 0.3 + 1) = 1.05$. For tuple o_2, tuple o_1 is the only tuple with a larger score, and we do not know the probability of o_2. We compute its lower bound, which is $R_E^{low}(o_2) = 1.8 - 0.3 \times (1.8 - 0.3 + 1) = 1.05$. For tuple o_3, tuple o_1 and o_2 are the tuples with a larger score, and $p(o_3) = 0.4$.

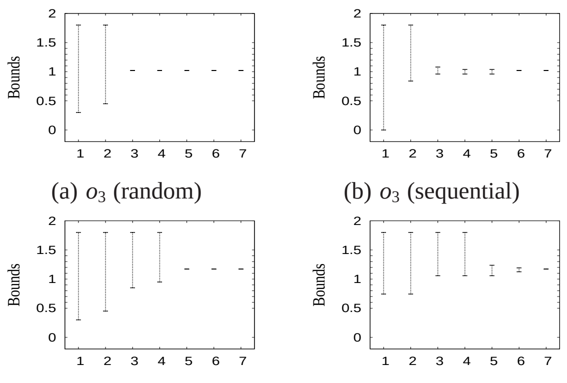
Tuple	Randomly Access P		Sequentially Access P	
	$R_E^{low}(o_i)$	$R_E^{up}(o_i)$	$R_E^{low}(o_i)$	$R_E^{up}(o_i)$
o_1	1.05	1.05	1.05	1.05
o_2	1.4475	1.4475	1.05	-
o_3	1.02	1.02	0.96	1.08
o_5	-	-	1.0575	-
unseen	0.85	-	1.17	-

Table 5: Bounds of $R_E(o_i)$ in ran/seq access

$R_E(o_3) = 1.8 - 0.4\times(1.8 - 0.3 - p(o_2) - 0.4 + 1) = 0.96 + 0.4\times p(o_2)$, then $R_E^{up}(o_3) = 0.96{+}0.4{\times}0.3 = 1.08$ and $R_E^{low}(o_3) = 0.96$. For tuple o_5, we do not know its exact score, and only know that the tuples in $\mathcal{H}_s$ are with a larger score. Hence, $R_E^{low}(o_5) = E[|I|]{-}p(o_5){\times}(E[|I|]{-}p(o_1) - p(o_2) - p(o_3) - p(o_5) + 1) \geq 1.8 - 0.45\times(1.8 - 0.3 - 0 - 0.4 - 0.45 + 1) = 1.0575$, with a lower bound 1.0575. For all the unseen tuples o_i, $R_E^{low}(o_i) = E[|I|] - \bar{p}\times(E[|I|] - p(o_1) - p(o_2) - p(o_3) + 1) \geq 1.17$, with a lower bound 1.17. □

Table 5 summarizes the bounds for seen and unseen tuples, after three iterations of random/sequential accesses on relation P respectively. The $R_E(o_i)$ for o_1, o_2, and o_3, are exact values in random access, which is tighter compared to the bounds in sequential access. But the lower bound for unseen tuples in random access is looser, i.e. $R_E^{low}(o) = 0.85$, whereas $R_E^{low}(o) = 1.17$ in sequential access, which is tighter.

Fig. 1 shows the [lower bound, upper bound] interval for tuple o_3 and o_5 in every iteration from 1 to 7. As we get more information, the lower bound goes non-decrease, and the upper bound goes non-increase, eventually we get the exact R_E values.



(a) o_3 (random) (b) o_3 (sequential)

(c) o_5 (random) (d) o_5 (sequential)

Figure 1: R_E bound changes for o_3 and o_5

Lemma 4.2: *Our bounding scheme is correct.* □

The bounding scheme is correct based on Lemma 4.1.

4.2 Ranking Function P_{HR}

If the tuples $o_1, \cdots, o_n$ are in descending score order and are totally independent, the $P_{HR}(o_i)$ function can be simplified as below.

$$P_{HR}(o_i) = p(o_i) \times \prod_{j=1}^{i-1}(1 - p(o_j)) \tag{4}$$

It is the product of the nonexistence probability of tuples o_j that have a larger score than the score of o_i and the probability of tuple o_i itself. We prove that P_{HR} is a monotone function in the following.

Lemma 4.3: *The P_{HR} function, on which the Top-kPHR query is based, is a monotone function. For any two tuples, o_i and o_j, if $score(o_i) \geq score(o_j)$ and $p(o_i) \geq p(o_j)$, then $P_{HR}(o_i) \geq P_{HR}(o_j)$.* □

Tuple	Randomly Access P		Sequentially Access P	
	$P_{HR}^{up}(o_i)$	$P_{HR}^{low}(o_i)$	$P_{HR}^{up}(o_i)$	$P_{HR}^{low}(o_i)$
o_1	0.3	0.3	0.3	0.3
o_2	0.105	0.105	0.21	-
o_3	0.238	0.238	0.28	0.196
o_5	-	-	0.189	-
unseen	0.357	-	0.126	-

Table 6: Bounds of $P_{HR}(o_i)$ in ran/seq access

Proof Sketch: As all the tuples are totally independent, the following equation holds.

$$\frac{P_{HR}(o_j)}{P_{HR}(o_i)} = \frac{p(o_j)}{p(o_i)} \times \prod_{l=i}^{j-1}(1 - p(o_l)) \tag{5}$$

Here, the first part $p(o_j)/p(o_i) \leq 1$ because $p(o_i) \geq p(o_j)$, and the second part is no larger than one too. Therefore the probabilistic score function P_{HR} is monotone, $P_{HR}(o_i) \geq P_{HR}(o_j)$, if $score(o_i) \geq score(o_j)$ and $p(o_i) \geq p(o_j)$. □

Let the upper and lower bounds of $P_{HR}(o_i)$ be $P_{HR}^{up}(o_i)$ and $P_{HR}^{low}(o_i)$, respectively. We consider random and sequential accesses on relation P respectively.

Random access on relation P: In this scenario, we get the probability for each seen tuple by a random access on relation P. For any seen tuple we know its exact P_{HR} value. For all the unseen tuples we can upper bound it by $\prod_{j=1}^{i}(1 - p(o_j))$, where o_i is the last accessed tuple from relation S. Note that this upper bound is tight, because an unseen tuple o_{i+1} may have probability 1.

Example 4.3: Consider relation S and relation P in Table 3. Assume the first three tuples in relation S have been retrieved. The set of seen tuples is $\{o_1(100, 0.3), o_2(95, 0.15), o_3(90, 0.4)\}$. Based on Eq. (4), we have $P_{HR}(o_1) = 0.3$, $P_{HR}(o_2) = 0.105$, and $P_{HR}(o_3) = 0.238$. The upper bound for any unseen tuple, o, is $P_{HR}^{up}(o) = (1-0.3)\cdot(1-0.15)\cdot(1-0.4) = 0.357$. This upper bound is achieved by giving the unseen tuple with highest score a probability 1. That is, o_4 is estimated to have a probability 1. □

Sequential access on relation P: In this scenario, in every iteration, we retrieve a tuple from relation S and a tuple from relation P in descending score/probability order respectively. We know the score and/or probability for the retrieved tuples from both relations. Let $\bar{p}$ denote the last retrieved probability from P.

For a tuple $o_i \in \mathcal{H}_s$, all the tuples o_j such that $score(o_j) > score(o_i)$ are in $\mathcal{H}_s$ already. We compute $P_{HR}(o_i)$ using Eq. (4). However, because the probability for some tuples may be unknown, we need to upper bound and lower bound $P_{HR}(o_i)$. If $p(o_i)$ is unknown, then we upper bound it by $\bar{p}$, and lower bound it by 0. For each tuple o_j involved in Eq. (4) to compute $P_{HR}(o_i)$, the upper/lower bounds of $(1 - p(o_j))$ are 1 and $(1 - \bar{p})$ respectively.

For a tuple $o_i \in \mathcal{H}_{\neg s}$, we upper/lower bound $P_{HR}(o_i)$ value. Note that its lower bound is 0. All the tuples in $\mathcal{H}_s$ have a score larger than $score(o_i)$. We upper bound $P_{HR}(o_i)$ by multiplying $p(o_i)$ and $(1 - p(o_j))$ for all the tuples in $\mathcal{H}_s$ as discussed above. Note that there may exist some tuple o_j that has a larger score than o_i but has not been retrieved from relation S yet. Similarly, we upper bound $P_{HR}(o)$ for the unseen tuples by multiplying $\bar{p}$ and $(1 - p(o_j))$ for all the tuples in $\mathcal{H}_s$ as discussed above.

In a summary, the bounds of P_{HR} for tuples in different sets are listed in Table 4.

Example 4.4: Assume that we have retrieved 3 tuples from both relation S and relation P in Table 3, respectively. Then, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4)\}$ and $\mathcal{H}_{\neg s} = \{o_5(-, 0.45)\}$. $\mathcal{H}_s$

can be further partitioned into $\mathcal{H}_s^+ = \{o_1(100, 0.3), o_3(90, 0.4)\}$ and $\mathcal{H}_s^- = \{o_2(95, -)\}$. Here, $\bar{p} = 0.3$, which is the last probability we have seen. In the following, we compute upper bounds and lower bounds for all the partial/full seen tuples. For tuple o_1, no tuple has a larger score. $p(o_1) = 0.3$, then $P_{HR}(o_1) = p(o_1) = 0.3$. For tuple o_2, tuple o_1 is the only tuple with a larger score, but we do not know the probability of o_2. $P_{HR}(o_2) = p(o_2) \times (1 - p(o_1))$, then $P_{HR}^{up}(o_2) = \bar{p} \times (1 - p(o_1)) = 0.21$ and $P_{HR}^{low}(o_2) = 0$. For tuple o_3, tuple o_1 and o_2 are the tuples with larger scores, and $p(o_3) = 0.4$. $P_{HR}(o_3) = p(o_3) \times (1 - p(o_1)) \times (1 - p(o_2)) = 0.4 \times (1 - 0.3) \times (1 - p(o_2)) = 0.28 \times (1 - p(o_2))$, then $P_{HR}^{up}(o_3) = 0.28$ and $P_{HR}^{low}(o_3) = 0.28 \times (1 - \bar{p}) = 0.196$. For tuple o_5, we do not know its exact score, but we know that the tuples in $\mathcal{H}_s$ are with a larger score. So $P_{HR}^{up}(o_5) = p(o_5) \times (1 - p(o_1)) \times (1 - p(o_2)) \times (1 - p(o_3)) \leq 0.45 \times (1 - 0.3) \times (1 - 0) \times (1 - 0.4) = 0.189$, with an upper bound 0.189. For all the unseen tuples o_i, $P_{HR}^{up}(o_i) = \bar{p} \times (1 - p(o_1)) \times (1 - p(o_2)) \times (1 - p(o_3)) \leq 0.126$, with an upper bound 0.126. □

Table 6 summarizes the bounds for seen and unseen tuples, after three iterations of random/sequential accesses on relation P respectively. The $P_{HR}(o_i)$ for o_1, o_2, and o_3, are exact values in random access, which is tighter compared to the bounds in sequential access. But the upper bound for unseen tuples is a little looser i.e. $P_{HR}^{up}(o) = 0.357$, whereas $P_{HR}^{up}(o) = 0.126$ in sequential access, which is tighter.

Lemma 4.4: *Our bounding scheme is correct and tight among all possible bounding schema provided that relation P is sorted in descending probability order.* □

Proof Sketch: The correctness of our bounding scheme directly follows from Lemma 4.3. For the tightness, we assume that a tuple may have zero probability. There does not exist any other bounding scheme (without random guess) that is more tight than ours. Based on Eq. (4), the lower bound is achieved, when all the tuples with a larger score has probability $\bar{p}$; and the upper bound is achieved when all the tuples with larger score has probability 0. Note that the upper bound and lower bound are achievable individually. □

4.3 Ranking Function with Exclusive Relationship

In the previous section, we discussed the bounds for R_E and P_{HR} respectively, assuming that all the tuples are independent. In this section, in a general, we discuss bounds for R_E and P_{HR} in an x-Relation with exclusive relationships.

Let $o_i \diamond o_j$ denote that tuple o_i and o_j are mutually exclusive, i.e. they belong to the same x-tuple τ, $o_i \in \tau$ and $o_j \in \tau$, and let $o_i \bar{\diamond} o_j$ denote that tuple o_i and o_j are from different x-tuples (independent). Note that o_i and o_j are different tuples. Let τ_{o_i} denote the x-tuple that o_i belongs to, i.e., $\tau_{o_i} = \{o_j \mid o_j \diamond o_i\} \cup \{o_i\}$.

4.3.1 R_E function

Assume the tuples $o_1, \cdots, o_n$ are in descending score order, the $R_E(o_i)$ with exclusive relationship is as follows.

$$\begin{aligned}
& R_E(o_i) \\
= \; & p(o_i) \cdot \sum_{o_j \bar{\diamond} o_i, j<i} p(o_j) \\
& +(1 - p(o_i)) \cdot \left(\frac{\sum_{o_j \diamond o_i} p(o_j)}{1 - p(o_i)} + \textstyle\sum_{o_j \bar{\diamond} o_i} p(o_j)\right) \\
= \; & p(o_i) \cdot \Big(\sum_{j<i} p(o_j) - \sum_{o_j \diamond o_i, j<i} p(o_j)\Big) + \sum_{o_j \diamond o_i} p(o_j) \\
& +(1 - p(o_i)) \cdot (E[|I|] - p(o_i) - \textstyle\sum_{o_j \diamond o_i} p(o_j)). \\
= \; & E[|I|] - p(o_i) \cdot \Big(E[|I|] - \sum_{j<i} p(o_j) - p(o_i) + 1\Big) \\
& +p(o_i) \cdot \textstyle\sum_{o_j \diamond o_i, j>i} p(o_j)
\end{aligned} \tag{6}$$

where the first equation is from [11]. Compared with R_E in the independent case, there is one extra term $p(o_i) \cdot \sum_{o_j \diamond o_i, j>i} p(o_j)$.

When randomly accessing relation P, the lower bound for the unseen tuples is $R_E(o_i) \geq \sum_{j \leq i} p(o_j)$, which is the same as Eq. (3). However, the $R_E(o_i)$ for the seen tuples can not be bounded tightly. Even though we have retrieved all the tuples with higher score and their probabilities, we still do not know those tuples in the same x-tuple with o_i, i.e., the term $\sum_{o_j \diamond o_i, j>i} p(o_j)$ is unknown.

When sequentially accessing relation P, the bounding scheme is more complicated compared to that discussed in Section 4.1. In Section 4.1, $p(o_i)$ for unknown probability is bounded by $0 \leq p(o_i) \leq \bar{p}$. But, when mutually exclusive exists, it is upper bounded by

$$\min\{\bar{p}, 1 - \sum_{o_j \diamond o_i, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j)\}$$

When $\sum_{o_j \diamond o_i, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j) > 1 - \bar{p}$, $p(o_i)$ must be less than $\bar{p}$. In order to bound $R_E(o_i)$ when mutually exclusive exists, we reorganize Eq. (6) in the granularity of x-tuples. It is possible to get better bounds, because $\sum_{o_j \in \tau} p(o_j) \leq 1$. Eq. (6) is reorganized as follows.

$$\begin{aligned}
& R_E(o_i) \\
= \; & E[|I|] - p(o_i) \cdot \Big(E[|I|] + 1 - \sum_{j<i} p(o_j) - p(o_i) - \sum_{\substack{o_j \diamond o_i \\ j>i}} p(o_j)\Big) \\
= \; & E[|I|] - p(o_i) \cdot \Big(E[|I|] + 1 - \sum_{\substack{\tau \in \mathcal{X} \\ o_i \notin \tau}} \sum_{\substack{o_j \in \tau \\ j<i}} p(o_j) - \sum_{o_j \in \tau_{o_i}} p(o_j)\Big)
\end{aligned} \tag{7}$$

The x-tuples are independent, so are their bounds. For the term $\sum_{o_j \in \tau, j<i} p(o_j)$, the lower bound is obtained by replacing all the unknown $p(o_j)$ with 0, i.e., $\sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+, j<i} p(o_j)$. There exist two possible upper bounds. One is to replace all of the unknown $p(o_j)$ with $\bar{p}$, i.e., $\sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+, j<i} p(o_j) + \bar{p} \cdot |\{o_j \in \tau \mid o_j \in \mathcal{H}_s^-, j < i\}|$, where $|\cdot|$ is the size of a set. The other is one minus the summation of the probabilities for tuples in τ that has a larger score than $score(o_i)$, i.e., $1 - \sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+ \&\& j>i \| o_j \in \mathcal{H}_{\neg s}} p(o_j)$. The upper bound is the minimum of the two. Similarly, for the term $\sum_{o_j \in \tau_{o_i}} p(o_j)$, its lower bound can be obtained by replacing all the unknown $p(o_j)$ with 0, i.e., $\sum_{o_j \in \tau_{o_i}, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j)$, the upper bound can be obtained by replacing all the unknown $p(o_j)$ for tuples in τ_{o_i} with $\bar{p}$, if we can get the size information about x-tuple τ_{o_i}, otherwise, it can only be trivially upper bounded by 1. By combining the corresponding lower bounds and upper bounds for each term, we get the lower bound and upper bound for Eq. (7).

In a summary, the bounds of $E[|I|] - R_E$ for tuples in different sets are shown in Table 7.

Example 4.5: Consider the two relations S and P in Table 3, assume there is a mutually exclusive relationship between tuple o_1 and o_6, i.e. $\tau_1 = \{o_1, o_6\}$, and the other tuples are independent. The x-tuple information can be possibly maintained in relation S, in an additional column named XID. The tuples identified by unique OID share the same XID if they belong to the same x-tuple. In addition, we add an additional column called Xcnt which records the number of alternatives an x-tuple has. With this additional column Xcnt, we can achieve tighter bound. It is achieved by the following information. For example, when retrieving o_6 from relation S by sequential access, we get the information that all the alternatives of x-tuple τ_1 have been retrieved, because τ_1 has only two alternatives and the other alternative o_1 has been retrieved. $E[|I|] = 1.8$. After three sequential accesses on both S and P, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4)\}$, $\mathcal{H}_{\neg s} = \{o_5(-, 0.45)\}$, and $\bar{p} = 0.3$. We also know that tuple o_2 and o_3 have no other alternatives from the same x-tuple, and there are more alternatives from the x-tuple that contains o_1. For tuple o_1, $R_E(o_1) =$

Set	S	P	Type	Bounds for $E[\lvert I\rvert] - R_E$
$\mathcal{H}_s^+$	✓	✓	*upper*	$p(o_i) \cdot (E[\lvert I\rvert] + 1 - \sum_{\tau\in\mathcal{X}, o_i\notin\tau} \sum_{o_j\in\tau, o_j\in\mathcal{H}_s^+, j<i} p(o_j) - \sum_{o_j\diamond o_i, o_j\in\mathcal{H}_s^+\cup\mathcal{H}_{\neg s}} p(o_j) - p(o_i))$
$\mathcal{H}_s^+$	✓	✓	*lower*	$p(o_i) \cdot (E[\lvert I\rvert] + 1$ $- \sum_{\substack{\tau\in\mathcal{X}\\ o_i\notin\tau}} \min\{\sum_{\substack{o_j\in\tau\\ o_j\in\mathcal{H}_s^+\\ j<i}} p(o_j) + \bar{p}\cdot\lvert\{o_j\in\tau \mid o_j\in\mathcal{H}_s^-, j<i\}\rvert, 1 - \sum_{\substack{o_j\in\tau\\ o_j\in\mathcal{H}_s^+\\ j>i}} p(o_j) - \sum_{\substack{o_j\in\tau\\ o_j\in\mathcal{H}_{\neg s}}} p(o_j)\}$ $- \min\{\sum_{o_j\diamond o_i, o_j\in\mathcal{H}_s^+\cup\mathcal{H}_{\neg s}} p(o_j) + p(o_i) + \bar{p}\cdot(\lvert\tau_{o_i}\rvert - \lvert\{o_j\diamond o_i \mid o_j\in\mathcal{H}_s^+\cup\mathcal{H}_{\neg s}\}\rvert - 1), 1\})$
$\mathcal{H}_s^-$	✓	×	*upper*	$\min\{\bar{p}, 1 - \sum_{\substack{o_j\diamond o_i\\ o_j\in\mathcal{H}_s^+\cup\mathcal{H}_{\neg s}}} p(o_j)\} \cdot (E[\lvert I\rvert] + 1 - \sum_{\substack{\tau\in\mathcal{X}\\ o_i\notin\tau}} \sum_{\substack{o_j\in\tau\\ o_j\in\mathcal{H}_s^+\\ j<i}} p(o_j) - \sum_{\substack{o_j\diamond o_i\\ o_j\in\mathcal{H}_s^+\cup\mathcal{H}_{\neg s}}} p(o_j))$
$\mathcal{H}_{\neg s}$	×	✓	*upper*	$p(o_i) \cdot (E[\lvert I\rvert] + 1 - \sum_{o_j\bar{\diamond} o_i, o_j\in\mathcal{H}_s^+} p(o_j) - \sum_{o_j\in\tau_{o_i}, o_j\in\mathcal{H}_s^+\cup\mathcal{H}_{\neg s}} p(o_j))$
U	×	×	*upper*	$\bar{p}\cdot(E[\lvert I\rvert] + 1 - \sum_{o_j\in\mathcal{H}_s^+} p(o_j))$

Table 7: $E[|I|] - R_E$ **bounds for tuple** o_i **(independent/exclusive)**

$E[|I|] - p(o_1)(E[|I|] + 1 - \sum_{o_j\in\tau_{o_1}} p(o_j)) = 0.96 + 0.3\sum_{o_j\in\tau_{o_1}} p(o_j)$, where $\sum_{o_j\in\tau_{o_1}} p(o_j)$ is lower bounded by $p(o_1) = 0.3$ and upper bounded by $p(o_1) + \bar{p} = 0.6$. Then $R_E^{low}(o_1) = 1.05$ and $R_E^{up}(o_1) = 0.96 + 0.3\times 0.6 = 1.14$. For tuple o_2, $R_E(o_2) = E[|I|] - p(o_2)(E[|I|] - p(o_1) - p(o_2) + 1)$, and then $R_E^{low}(o_2) = 1.8 - 0.3\times(1.8 - 0.3 + 1) = 1.05$. For tuple o_3, $R_E^{low}(o_3) = 0.96$ and $R_E^{up}(o_3) = 1.08$. For tuple o_5, $R_E^{low}(o_5) = 1.0575$. And $R_E^{low}(o) = 1.17$ for unseen tuples. □

In this example, although we know both score and probability for o_1 and o_1 has the highest score, we can not get the exact $R_E(o_1)$, because there may exist some tuples with a smaller score and in the same x-tuple where o_1 belongs to.

4.3.2 P_{HR} *function*

Assume the tuples $o_1, \cdots, o_n$ are in descending score order, the $P_{HR}(o_i)$ with exclusive relationship is as follows.

$$P_{HR}(o_i) = p(o_i) \times \prod_{\tau\in\mathcal{X}, o_i\notin\tau} (1 - \sum_{o_j\in\tau, j<i} p(o_j)) \tag{8}$$

which is the multiplication of the existence probability of o_i and the nonexistence probabilities of other tuples with higher score. Note that, the multiplication is in the granularity of x-tuple, because the tuples from an x-tuple are mutually exclusive.

When randomly accessing relation P, we only need to upper bound the unseen tuples. Let o_i be the last seen tuple, the P_{HR} for unseen tuples can be upper bounded by $\prod_{\tau\in\mathcal{X}}(1 - \sum_{o_j\in\tau, j<i} p(o_j))$.

When sequentially accessing relation P in descending probability order, we need both upper and lower bounds for seen tuples. In Eq. (8), if $p(o_i)$ is unknown, then the upper bound is $\min\{\bar{p}, 1 - \sum_{o_j\diamond o_i} p(o_j)\}$ and the lower bound is 0. For each unknown $p(o_j)$ in the second part of Eq. (8), we replace it by 0 for the upper bound, and by $\bar{p}$ for the lower bound. If the lower bound is negative, then its lower bound is 0.

For each $o_i \in \mathcal{H}_{\neg s}$ whose score is unknown, we upper bound it by $P_{HR}^{up}(o_i) = p(o_i) \times \prod_{\tau\in\mathcal{X}, o_i\notin\tau}(1 - \sum_{o_j\in\tau, o_j\in\mathcal{H}_s^+} p(o_j))$. For the unseen tuples, we can upper bound it by

$$\max_{\tau\in\mathcal{X}}\{\frac{\min\{\bar{p}, 1 - \sum_{o_j\in\mathcal{H}_s^+\cup\mathcal{H}_{\neg s}} p(o_j)\}}{1 - \sum_{o_j\in\mathcal{H}_s^+} p(o_j)}\} \prod_{\tau\in\mathcal{X}}(1 - \sum_{o_j\in\tau, o_j\in\mathcal{H}_s^+} p(o_j))$$

4.4 Discussions

In this section, we discuss two issues. One is the advantages and disadvantages related to random/sequential access. The other is the bounding scheme for other possible top-k probabilistic ranking queries.

Random vs Sequential access: We discuss the advantages and disadvantages of the bounds for random and sequential accesses. Consider at the iteration i, we distinguish the whole set of (seen or unseen) tuples into two sets $\mathcal{H}_s$ and $\mathcal{H}_{\neg s} \cup U$.

When randomly accessing relation P, we retrieve the probability by a random access each time we retrieve the corresponding tuple from relation S. $\mathcal{H}_{\neg s} = \emptyset$ and $\mathcal{H}_s = \mathcal{H}_s^+$. For tuples $o_i \in \mathcal{H}_s^+$, we can get the exact $\mathsf{pscore}(o_i)$ values, both upper bound and lower bound are $\mathsf{pscore}(o_i)$ which is absolutely tight. But, for tuples $o_i \in U$, we do not know any information about the score and probability. Then, the upper bound for $\mathsf{pscore}(o_i)$ can be arbitrarily loose.

When sequentially accessing relation P, we have additional information $\bar{p}$, which is the upper bound for all the unknown probabilities. We can upper bound $\mathsf{pscore}(o_i)$ for tuples $o_i \in \mathcal{H}_{\neg s} \cup U$ more tighter than that in random access. But, among tuples $o_i \in \mathcal{H}_s$, the probability for some tuples may be unknown, we can only bound it by 0 from below and $\bar{p}$ from above. The lower/upper bound for $\mathsf{pscore}(o_i)$ is a little looser than that in random access.

In summary, with random access on relation P, we can get better bounds of $\mathsf{pscore}(o_i)$ for tuples in $\mathcal{H}_s$. With sequential access on relation P, we can get better bounds of $\mathsf{pscore}(o_i)$ for tuples in $\mathcal{H}_{\neg s} \cup U$. But, we can not get better bounds for both tuples in $\mathcal{H}_s$ and tuples in $\mathcal{H}_{\neg s} \cup U$ at the same time, in either random access or sequential access.

The bounds for other functions: There are also other probabilistic ranking functions used in the literature, e.g., top-k probability [16], or the probability for a tuple to be ranked at the j-th position in possible worlds [34]. The same approach in the literature can be used to bound pscore in random access, because it is the same as to process in the same relation. However, it is very difficult to find an upper bound or nontrivial lower bound (other than 0) for these probabilistic ranking functions, in sequential access of relation P. Below, we discuss why it is difficult to find an upper bound for the probability that a tuple ranked at the j-th position in possible worlds.

Suppose we have done 4 sequential accesses, and we have $\mathcal{H}_s = \{o_1(100, p_1),\ o_2(95, p_2), o_3(90, p_3), o_4(85, 0.4)\}$, where p_i is short for $p(o_i)$ and is unknown. We only know that p_1, p_2, and p_3, are in the range of $[0, \bar{p}]$, where $\bar{p}$ is the last seen probability. Then the probability for o_4 to be ranked at the second place in possible worlds is as follows.

$$p(o_4)\cdot(p_1(1-p_2)(1-p_3) + p_2(1-p_1)(1-p_3) + p_3(1-p_1)(1-p_2))$$

which is $0.4\cdot(3p_1p_2p_3 - 2p_1p_2 - 2p_1p_3 - 2p_2p_3 + p_1 + p_2 + p_3)$. It is a polynomial of degree 3, it is hard to get the upper bound. A very loose upper bound is $0.4\cdot(3p_1p_2p_3 + p_1 + p_2 + p_3)$, where p_1, p_2, p_3 is replaced by $\bar{p}$. This upper bound can be very loose, and can be arbitrarily large when there are more than 3 tuples with unknown probability. Even worse, in order to get such upper bound, it takes $O(2^n)$ time in order to compute the probability for a tuple to be ranked at the second position, where n is the number of probabilities that are unknown.

Algorithm 1 PRR(S, P, k)

```
Input:   relation $S$, relation $P$, and a number $k$
Output:  Top-k tuples in sorted order based on $R_E$.

 1: initialize a priority queue of size $k$, $Q$, to be empty;
 2: while less than $k$ tuples reported do
 3:     $(o_i, s_i) \leftarrow next(S)$;
 4:     $(o_i, p_i) \leftarrow find(P, o_i)$;
 5:     compute $R_E(o_i)$ using Eq. (2);
 6:     insert $(o_i, R_E(o_i))$ into $Q$;
 7:     compute the lower bound of all the unseen tuples as $R_E^{low}(o)$;
 8:     while less than $k$ tuples reported do
 9:         let $o_i$ be the tuple with largest $R_E(o_i)$ in $Q$.
10:         if $R_E(o_i) \le R_E^{low}(o)$ then
11:             report $o_i$ as the next tuple in the top-k answers;
12:             delete $o_i$ from $Q$;
13:         else
14:             break;
```

Algorithm 2 PRS(S, P, k)

```
Input:   relation $S$, relation $P$, and a number $k$
Output:  Top-k tuples in sorted order based on $R_E$.

 1: while less than $k$ tuples reported do
 2:     $(o_i, s_i) \leftarrow next(S)$;
 3:     $(o_j, p_j) \leftarrow next(P)$;
 4:     $\bar{p} \leftarrow p_j$;
 5:     update $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$;
 6:     for all the tuples $o_i \in \mathcal{H}_s$, compute its upper bound and lower bound
        $R_E^{up}(o_i)$ and $R_E^{low}(o_i)$;
 7:     compute the lower bound for all the tuples in $\mathcal{H}_{\neg s}$ and all the unseen
        tuples;
 8:     while less than $k$ tuples reported do
 9:         let $o_i$ be the unreported tuple in $\mathcal{H}_s$ with smallest lower bound;
10:         if $R_E^{up}(o_i)$ is no larger than all the other lower bounds then
11:             report $o_i$ as the next tuple in the top-k answers;
12:         else
13:             break;
```

5. I/O EFFICIENCY

We discussed the bounding schema for both R_E and P_{HR} in Section 4. In this section, we discuss two algorithms for random access and sequential access of relation P respectively for Top-kERV queries when all x-tuples are independent. It is straightforward to extend the algorithms to support the general mutually exclusive case, and it is straightforward to extend the algorithms to compute probability of highest rank queries using the bounding scheme of P_{HR} instead of R_E. We also discuss how to combine the advantages from both random/sequential access.

5.1 Random Access on Relation P

The algorithm for random access on relation P is similar to the algorithms for probabilistic rank queries in a single relation.

Algorithm 1 shows the detailed steps of computing Top-kERV queries. It takes three inputs: a relation S which is sorted in descending score order, a relation P, and a number k. It uses a priority queue Q of size k which is initialized to be empty (line 1). It outputs the top-k answers in a while loop (line 2-14), and will stop when top-k answers are output. In the while loop, it gets the next pair (o_i, s_i) from relation S where $s_i = score(o_i)$ (line 3). The score will be the largest among those unseen tuples in relation S, because relation S is sorted in descending score order. Then, it obtains $p(o_i)$ by calling $find(P, o_i)$ with the OID o_i by a random access (line 4). It computes $R_E(o_i)$ using Eq. (2) (line 5), and inserts the pair $(o_i, R_E(o_i))$ into the priority queue Q where tuples are sorted in ascending $R_E(o_i)$ order (line 6). If its size exceeds k when inserting a pair into Q, we delete the pair with the largest R_E value from Q. It also computes the lower bound for all the unseen tuples denoted as $R_E^{low}(o)$, using the bounding approach discussed in Section 4.1 (line 7). All the tuples o_i in Q with $R_E(o_i) \le R_E^{low}(o)$ can be determined to be the top-k answers (line 8-14). Note that in Algorithm 1 the numbers of tuples retrieved from relation S and relation P are the same.

Example 5.1: Consider the two relations, S and P, in Table 3. Let $k = 2$. Algorithm 1 executes as follows. In the first two iterations, $(o_1, 100)$ and $(o_1, 0.3)$, and $(o_2, 95)$ and $(o_2, 0.15)$, are retrieved from relation S and relation P. In the third iteration, $(o_3, 90)$ and $(o_3, 0.4)$ are retrieved. We have $R_E(o_1) = 1.05$, $R_E(o_2) = 1.4475$, and $R_E(o_3) = 1.02$. The lower bound is computed as $R_E^{low}(o) = 0.85$. All these $R_E(o_1)$, $R_E(o_2)$, and $R_E(o_3)$ are larger than the lower bound $R_E^{low}(o)$. Therefore, no tuples can be determined to be in the top-k answers in this iteration. In the fourth iteration, we retrieve o_4 from both relations: $(o_4, 85)$ and $(o_4, 0.1)$. $R_E(o_4) = 1.615$. The lower bound computed is $R_E^{low}(o) = 0.95$. No tuple can be determined to be the top-k answers. In the fifth iteration, we retrieve o_5 from both relations: $(o_5, 80)$ and $(o_5, 0.45)$. $R_E(o_5) = 1.17$. The lower bound computed is $R_E^{low}(o) = 1.4$. $R_E(o_3)$ and $R_E(o_1)$ are smaller than $R_E^{low}(o)$, then o_3 and o_1 can be determined to be the top-2 results. □

Theorem 5.1: *Algorithm 1 correctly finds the top-k tuples with respect to R_E, with sequential access on relation $\mathcal{S}$ and random access on relation $\mathcal{P}$.* □

Proof Sketch: For all the seen tuples o_i, we can compute $R_E(o_i)$ exactly by Eq. (2). For the unseen tuples o, it can be lower bounded as $R_E^{low}(o)$. All the tuples output by Algorithm 1 are guaranteed to be no larger than $R_E^{low}(o)$ (line 10). Then it is guaranteed to be in the top-k answers. □

5.2 Sequential Access on Relation P

Because random access usually is much expensive compared to sequential access, in this section, we consider sequential accessing relation P provided that relation P is sorted in descending order in terms of probability, as well as sequential accessing relation S which is sorted in descending score order.

We sequentially access relation S and relation P. In every iteration, we retrieve (o_i, s_i) from relation S, and (o_j, p_j) from relation P, update $\bar{p}$ to be p_j, where $\bar{p}$ is the upper bound for all the unseen probabilities. We update $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$ which may require joining (o_i, s_i) and (o_j, p_j) with the existing retrieved tuples in $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$. We also update the upper bound and lower bound for all the seen tuples, and compute the lower bound for all the unseen tuples $R_E^{low}(o)$. Let o_i be the tuple with smallest lower bound among all seen tuples. If $R_E^{up}(o_i)$ is no larger than all the other lower bounds, then tuple o_i can be determined to be the next tuple in the top-k answers. Algorithm 2 shows the detailed steps. We explain it using an example.

Example 5.2: Consider the two relations, S and P, in Table 3. Let $k = 2$. Algorithm 2 executes as follows. In the first iteration, $(o_1, 100)$ and $(o_5, 0.45)$ are retrieved from relation S and relation P (line 2-3). $\bar{p} = 0.45$ (line 4). Here, $\mathcal{H}_s = \{(o_1(100, -)\}$ and $\mathcal{H}_{\neg s} = \{(o_5(-, 0.45)\}$, where every entry in $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$ represents $o_i(score(o_i), p(o_i))$. In the second iteration, $(o_2, 95)$ and $(o_3, 0.4)$ are retrieved from relation S and relation P. $\bar{p} = 0.4$. Here, $\mathcal{H}_s = \{(o_1(100, -), o_2(95, -)\}$ and $\mathcal{H}_{\neg s} = \{(o_5(-, 0.45), o_3(-, 0.4)\}$. The upper bounds for R_E of all the seen tuples are $E(|I|)$, no tuples can be the top-k answers.

In the third iteration, after retrieving $(o_3, 90)$ from relation S and

$(o_1, 0.3)$ from relation P, we update $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$ (line 5). Here, $\mathcal{H}_s = \{(o_1(100, 0.3), o_2(95, -), o_3(90, 0.4)\}$ and $\mathcal{H}_{\neg s} = \{(o_5(-, 0.45)\}$. Note that the entry $o_3(-, 0.4)$ is deleted from $\mathcal{H}_{\neg s}$ and its probability is added into $o_3(90, 0.4)$ in $\mathcal{H}_s$. The same is applied to o_1. The upper bounds and lower bounds in the third iteration are shown in Table 8. The tuple with smallest lower bound is o_3 such that $R_E^{low}(o_3) = 0.96$. Its upper bound ($R_E^{up}(o_3) = 1.08$) is larger than the lower bound of o_2 because $R_E^{low}(o_2) = 1.05$, so we continue for the next iteration.

Tuple	Iteration 3		Iteration 4	
	$R_E^{up}(o_i)$	$R_E^{low}(o_i)$	$R_E^{up}(o_i)$	$R_E^{low}(o_i)$
o_1	1.05	1.05	1.05	1.05
o_2	-	1.05	-	1.3
o_3	1.08	0.96	1.04	0.96
o_4	-	-	-	1.38
o_5	-	1.0575	-	1.0575
o_6	-	-	-	1.42
unseen	-	1.17	-	1.38

Table 8: Upper/lower bounds in 3rd and 4th iteration

In the fourth iteration, after retrieving $(o_4, 85)$ from relation S and $(o_6, 0.2)$ from relation P, we have $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4), o_4(85, -)\}$ and $\mathcal{H}_{\neg s} = \{o_5(-, 0.45), o_6(-, 0.2)\}$. $\bar{p} = 0.2$. We recompute the upper bounds and lower bounds R_E for the seen tuples, as shown in Table 8. The lower bound R_E value for the tuples in $\mathcal{H}_{\neg s}$ is $R_E^{low}(o_5) = 1.0575$, and the lower bound for the unseen tuples is $R_E^{low}(o) = 1.38$. The tuples with smallest lower bound is o_3, and its upper bound 1.04 is no larger than any other lower bounds. Therefore, o_3 can be determined to be the top-1 answer, although its exact $R_E(o_3)$ is still unknown. Then the unreported tuples with the smallest lower bound is o_1, and its upper bound 1.05 is no larger than any other unreported lower bounds. We can report o_1 as the top-2 answer. □

Compare Example 5.1 and Example 5.2. For random access on relation P, we report the top-2 answers after retrieving 5 tuples from relation S and 5 tuples from relation P. For sequential access on relation P, we can determine that tuples o_3 and o_1 must be in the top-2 answers, in the fourth iteration. It is does not only incur less expensive to conduct sequential access, but also retrieve less number of tuples compared to random access on relation P.

Theorem 5.2: *Algorithm 2 correctly computes the top-k tuples based on R_E, with sequential access on both relation S and relation P.* □

Proof Sketch: The correctness of Algorithm 2 directly follows from Lemma 4.1 and Lemma 4.2, and the correctness of the non-random access algorithms in [14, 32]. □

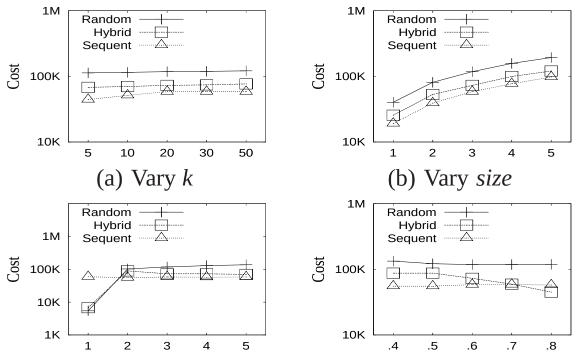
It is important to note that our sequential access is similar to the scenario discussed in [14, 32]. For each seen tuple, there is an upper bound and lower bound of $R_E(o_i)$. If $R_E^{low}(o_i) \leq R_E^{up}(o_j)$, then tuple o_i is guaranteed to rank higher than o_j. For the unseen tuples, $R_E(o)$ is guaranteed to be less than or equal to some lower bound value. However, both work reported in [14, 32] are for deterministic datasets and cannot be directly applied to probabilistic query processing.

5.3 Sequential and Random Access

In Section 5.1 and Section 5.2, we discussed algorithms to find the top-k answers, with either random access or sequential access on relation P. We also discussed the advantages and disadvantages of these two access methods in Section 4.4. It is hard to get better bounds of $\mathsf{pscore}(o_i)$ for both the seen tuples and unseen tuples.

Parameter	Range	Default
k	5, 10, 20, 30, 50	20
size	1, 2, 3, 4, 5 ($\times 10k$)	3
xsize	1, 2, 3, 4, 5	3
mean	0.4, 0.5, 0.6, 0.7, 0.8	0.6

Table 9: Parameters for all Testings



(a) Vary k (b) Vary *size*

(c) Vary *xsize* (d) Vary *mean*

Figure 3: Uniform Distribution

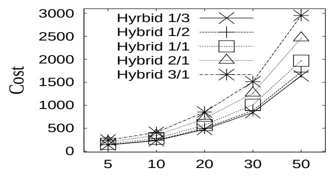
In this section, we discuss conducting both sequential and random access at the same time, to utilize both advantages of random access and sequential access. We can add random access of relation P into the framework of Algorithm 2, which is designed for sequential access only. In some iteration in Algorithm 2 (line 3), instead of sequentially retrieving the next tuple from relation P, we issue a random access to find the probability for the tuple in $\mathcal{H}_s$ with the largest score and unknown probability. Then, the bounds for all the tuples with smaller scores will become tighter. Note that, when a random access is issued, the $\bar{p}$ value will not be changed to the probability retrieved. Below, we give an example to show how random access helps bounding in sequential access.

Example 5.3: Consider the two relations, S and P, in Table 3. Let $k = 2$. Assume the probability of o_5 is changed to 0.5, i.e., $p(o_5) = 0.5$. $E[|I|] = 1.85$. After conducting four sequential accesses on relation S and relation P respectively, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4), o_4(85, -)\}$, $\mathcal{H}_{\neg s} = \{o_5(-, 0.5), o_6(-, 0.2)\}$, and $\bar{p} = 0.2$. Then the upper bounds and lower bounds are as follows. For tuple o_1, $R_E(o_1) = 1.085$. For tuple o_2, $R_E^{low}(o_2) = 1.34$. For tuple o_3, $R_E^{low}(o_3) = 0.99$, and $R_E^{up}(o_3) = 1.07$. For tuple o_4, $R_E^{low}(o_4) = 1.42$. For tuple o_5, $R_E^{low}(o_5) = 1.025$. For tuple o_6, $R_E^{low}(o_6) = 1.56$. For unseen tuple o, $R_E^{low}(o) = 1.42$. The upper bound of o_3, which is the tuple with smallest lower bound, is larger than the lower bound of o_5. Then no tuple can be determined to be in the top-k answers in this iteration.

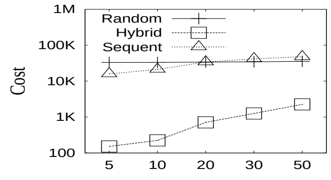
If, in the fourth iteration, we issue a random access on P instead of sequential access to the probability of o_2. Then, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, 0.15), o_3(90, 0.4), o_4(85, -)\}$, $\mathcal{H}_{\neg s} = \{o_5(-, 0.5)\}$, $\bar{p} = 0.3$. Note that $\bar{p}$ is larger than that of sequential access. The set of upper bounds and lower bounds are as follows. For tuple o_1, $R_E(o_1) = 1.085$. For tuple o_2, $R_E(o_2) = 1.49$. For tuple o_3, $R_E(o_3) = 1.05$. For tuple o_4, $R_E^{low}(o_4) = 1.25$. For tuple o_5, $R_E^{low}(o_5) = 1.1$. For unseen tuple o, $R_E^{low}(o) = 1.25$. Then, tuple o_3 can be determined with the highest rank, and tuple o_1 with the second highest rank. □

6. PERFORMANCE STUDIES

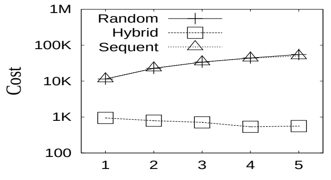
We conducted extensive performance studies to get top-k an-



(a) Vary *k*

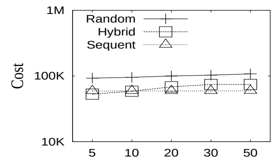


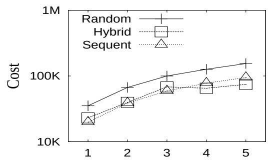
(b) Vary *k*

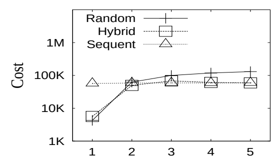


(c) Vary *size*

Figure 2: Real Data


(a) Vary *k*


(b) Vary *size*


(c) Vary *xsize*

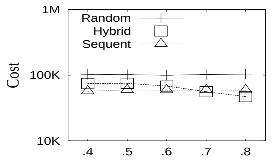
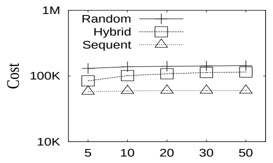
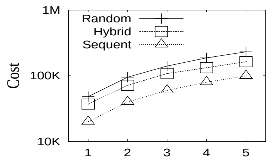
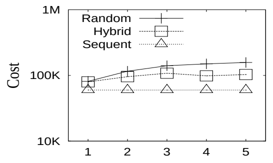

(d) Vary *mean*

Figure 4: Normal Distribution


(a) Vary *k*


(b) Vary *size*


(c) Vary *xsize*

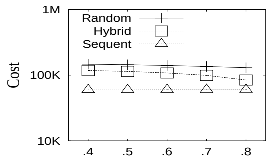
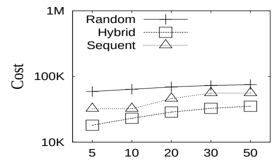
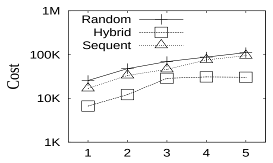
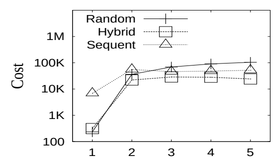

(d) Vary *mean*

Figure 6: Negative Correlated


(a) Vary *k*


(b) Vary *size*


(c) Vary *xsize*

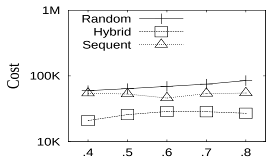
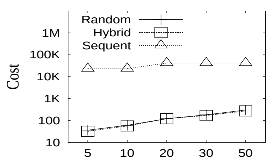

(d) Vary *mean*

Figure 5: Positive Correlated


(a) Vary *k*

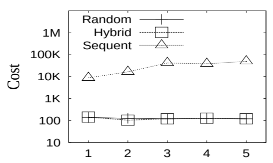

(b) Vary *size*

Figure 7: P_{HR} Testing

swers using the two pscore functions, namely R_E and P_{HR}. We tested 3 algorithms, namely Random, Sequent, and Hybrid. All the 3 algorithms sequentially access relation S, and access relation P as the names imply. Random randomly accesses relation P, Sequent sequentially accesses relation P, and Hybrid may sequentially and randomly access relation P.

We use both real datasets and synthetic datasets. For the real datasets, we extracted several sets of x-tuples from the International Ice Patrol (IIP) Iceberg Sightings Database (`http://nsidc.org/data/g00807.html`) which is a database that collects the activities of the iceberg in the North Atlantic. The data are collected through airborne Coast Guard reconnaissance missions and information from radar and satellites to monitor iceberg danger near the Grand Banks of Newfoundland. There are some imprecise information for each record which is recorded as the confidence level according to the source of sighting. The 6 confidence levels are converted to confidence probabilities 0.8, 0.7, 0.6, 0.5, 0.4, and 0.3 respectively. Each drifting activity may be recorded several times by several types of sources. The x-tuples are the records that are obtained at the same time and the same location. We collected records from 1998 to 2007 and generated 17,505 x-tuples which contain 50,879 tuples. For each x-tuple τ, we normalize the probabilities for each tuple $o \in \tau$ as follows, $p(o) = \frac{conf(o)}{\sum_{o' \in \tau} conf(o')} \cdot max\{conf(o') | o' \in \tau\}$, where $conf(o)$ is the confidence probability for the tuple o. For each tuple extracted, we set its score to be the number of days drifted because it is important in determining the status of icebergs. We extracted 5 datasets from the whole dataset, which are sized 10,000, 20,000, 30,000, 40,000 and 50,000 in terms of tuples respectively.

For the synthetic datasets, we have 4 types of distributions for the probabilities of x-tuples in the datasets, namely, uniform distribution, normal distribution, positive correlated distribution and negative correlated distribution. For the uniform distribution, given a mean value $0 < ave < 1$, suppose $d = min\{ave, 1 - ave\}$, all probabilities of x-tuples are distributed uniformly in the range $[ave - d, ave + d]$. For the normal distribution, for a mean value ave, all probabilities of x-tuples follow the normal distribution $N(ave, 0.2)$. For the positive and negative correlated distribution, the probabilities and the scores form a correlated bivariate with correlation 0.8 and −0.8 respectively.

The parameters used and their default values for both real and synthetic data are given in Table 9. k is for the top-k value in a top-k probabilistic query. *size* is the number of units for the dataset, where each unit contains 10,000 tuples. *xsize* is the average number of tuples in each x-tuple in the dataset. *mean* is the mean value under a certain distribution discussed above. The k and *size* parameters are used for both real and synthetic datasets, whereas *xsize* and *mean* parameters are used for the synthetic datasets only. We report the I/O cost follow the same approach given in [33]. The cost is measured in unit, one sequential I/O contributes one unit, and one random I/O contributes 5 units. All algorithms are implemented in Visual C++, and all tests were conducted on a 2.8GHz CPU, 2GB memory and 80GB disk space PC running Windows XP.

Exp-1 Real Datasets for R_E: The testing results for R_E using the real datasets are shown in Fig. 2. In Fig. 2(a), we also test the Hybrid algorithm with different accessing patterns between the random access and the sequential access. In our testing, `Hybrid i/j` means that the Hybrid algorithm performs i sequential accesses followed by j random accesses alternatively. From Fig. 2(a), `Hybrid i/j` performs well when $i < j$, and there is no much difference between different Hybrid algorithms with $i < j$ regarding the cost. Because `Hybrid 1/1` performs well among these variants, we use `Hybrid 1/1` as Hybrid to conduct our testing below. In Fig. 2(b), when k increases, the numbers of tuples visited for all the 3 algorithms increase because the top-k answers for all the 3 algorithms are incrementally generated. Random and Sequent have similar costs. Hybrid is much better than both Random and Sequent. One of the bottlenecks for Sequent is that, although the lower bounds of R_E for the unseen tuples increase in every iteration, the set of seen tuples with both scores and probabilities is small. Note that the seen tuples with both scores and probabilities have an upper bound, and thus can satisfy the stop conditions. When random access is integrated into sequential access, the number of seen tuples with both scores and probabilities increases. Thus Hybrid can stop in an early stage. In Fig. 2(c), when the number of tuples in the datasets increases, the cost for both Random and Sequent increases, but it decreases for Hybrid. The reason is that, in the same iteration the upper bound of R_E for the seen tuples with both scores and probabilities tend to be smaller in a dataset with a larger size, where $p(o_i)$ tends to be larger. Hybrid is also much better than both Random and Sequent.

Exp-2 Synthetic Datasets for R_E: We tested all the 3 algorithms for R_E using synthetic data. For each of the four probability distributions, namely, uniform, normal, positive correlated, and negative correlated, we vary k, *size*, *xsize*, and *mean*, to test the performance for each algorithm. The results are shown in Fig. 3, Fig. 4, Fig. 5 and Fig. 6 respectively.

Under the uniform distribution, Fig. 3(a) shows that when k increases, the cost for all the 3 algorithms increases, Hybrid does not perform as good as in the real dataset, because in the real dataset, some of the first several tuples tend to have high probabilities, which make the upper bounds of R_E for those tuples small, and thus output in early iterations. In Fig. 3(b), when the number of tuples in the dataset increases, the cost for all the three algorithms increases, Hybrid is between Sequent and Random. Fig. 3(c) shows that when *xsize* increases, the cost for all the 3 algorithms increases. Sequent performs badly when *xsize* is small, because the average probability for each tuple in each x-tuple is large. In this situation, $\overline{p}$ will be large in each iteration, thus the lower bound for unseen tuples will be loose. In addition, when the average probability for tuples is large, the lower bound for R_E in Sequent is small. Thus the performance is bad. On the other hand, the lower bound for the unseen tuples in Random increases faster, which makes it perform good. In Fig. 3(d), when the *mean* value increases, the cost for all the 3 algorithms decreases, Hybrid algorithm decreases faster. As shown in Fig. 4, The algorithms under the normal distribution perform similar as in the uniform distribution, Sequent does not perform well in the normal distribution, because there are not many tuples with very high probabilities or very low probabilities.

Fig. 5 and Fig. 6 show the distributions in two opposite situations, positive correlated and negative correlated. In the positive correlated data, the tuples with large scores tend to have high probabilities, whereas, in the negative correlated data, the tuples with large scores tend to have low probabilities. The curves are all similar to those in the normal distributions. The performance for all testings in the positive correlated data is much better than those in the negative correlated data. In the positive correlated data, the lower bound of R_E in Random increases fast in the first several iterations, because the first several tuples tend to have a large probability. For the same reason, the upper bound of R_E for each tuple in Sequent decreases fast in the first several iterations. For Sequent, the number of tuples with both scores and probabilities in the positive correlated data is much larger than that in the negative correlated data. It makes it faster in the positive correlated data.

Exp-3 Real Datasets for P_{HR}: We tested the P_{HR} function in the real dataset for all the 3 algorithms. The results are shown in Fig. 7. When either k or *size* increases, the cost for all the 3 algorithms increases. Random and Hybrid have similar performance, and is much better than Sequent. This is because, the upper bound of P_{HR} in the Random algorithm decreases fast. For example, even if all of the first 30 tuples have very low probability, say 0.1, the upper bound for the unseen tuples after 30 iterations becomes $(1-0.1)^{30} = 0.04$ which is very small. This means that we can output the top-k answers in an early stage.

7. RELATED WORK

Top-k Queries in Probabilistic Data: Uncertain/probability data has received increasing attention recently. There are several probabilistic data models and systems proposed, for example, Trio system [2, 6, 31], MystiQ system [12, 30], and MayBMS system [5, 4].

In the literature, several works study computing the top-k answers by the interplay of score and probability, based on the possible worlds semantics. Soliman et al. first study the ranking issues in probabilistic data under the possible world semantics [34], and propose two probabilistic ranking queries: U-Topk query and U-kRanks query. They also study ranking aggregate queries in probabilistic data [35]. Yi et al. [37] improve the performance of the U-Topk and U-kRanks queries using a dynamic programming approach. Hua et al. [16] study a PT-k query, which returns the set of tuples whose top-k probability is above a user-specified threshold, and propose three heuristic approaches to answer such PT-k queries. Cormode et al. [11] propose the expected rank query. They rank tuples based on their expected rank values. Jin et al. [22] study the U-Topk/U-kRanks/Pk-topk queries in an uncertain stream environment under a sliding-window model. Li et al. [23] compute the top-k answers in the scenario of uncertain distributed data, where subsets of the tuples are distributed at different places. All the existing studies assume that all the information about score and probability is stored in a single relation.

Join queries in uncertain database are studied in [10, 3]. Cheng et al. study probabilistic threshold join queries [10]. In [10], the values of join attributes are uncertain, which is represented as pdf. Two tuples can be joined with a probability, which is the probability that the two pdfs can join. Agrawal et al. [3] study the problem of finding top-k join answers in an uncertain database when memory is insufficient. Although they consider the join issues in a top-k query, they treat each probability attribute as an ordinary numeric attribute, and rank the answers based on the aggregated probabilities. In this paper, we are dealing with a different problem.

Top-k Queries in Deterministic Data: Top-k queries in deterministic data have been studied extensively. A detailed survey can be found in [19]. In general, it is to find the top-k answers with respect to a user specified score function by joining and aggregating multiple inputs(or relations).

The top-k algorithms by Fagin et al. are the most influential [13, 14]. They consider both random access and/or sequential access of the lists of base scores, where each list of a base score can be viewed as a separate relation. There are many works considering the scenario that random access is not supported by the underlying sources. The No Random Access (NRA) algorithm [14], the Stream-Combine algorithm [15], and the LARA-j algorithm [25] answer a top-k query by sequential accesses on the lists of base scores. The J^* algorithm [27], algorithms in [18], and the family of PBRJ algorithms [32] retrieve the join answers with top-k scores, using sequential access on the base relations. Marian et al. propose Upper and Pick algorithm to answer top-k queries, when sequential access is provided and also controlled random accesses is provided [26]. But, these work consider deterministic data, and can not be directly applied to probabilistic data. In probabilistic ranking, each tuple has both a score and a probability, the tuples are ranked based on the possible worlds semantics. In our work, we study the linkage between probabilistic ranking and traditional ranking, and we find the top-k answers in the framework of randomly and/or sequentially accessing relations.

8. CONCLUSION

In this paper we study probabilistic top-k ranking queries when scores and probabilities are stored in different relations. We focus on reducing the join cost in probabilistic top-k ranking. We investigate two probabilistic score functions, namely, R_E and P_{HR}, and explore two access methods, random access and sequential access. We give the upper/lower bounds for both access methods, and provide insights on the advantages and disadvantages of random/sequential access in terms of bounds. We propose random, sequential, and hybrid algorithms. The hybrid algorithm takes advantages from both random and sequential access. We conducted extensive performance studies using real and synthetic datasets, and we confirmed the efficiency of our approaches.

Acknowledgement

The work was supported by grants of the Research Grants Council of the Hong Kong SAR, China (No. 419008/419109), ARC Discovery Grants (No. DP0987557/DP0881035), and Google Research Award.

9. REFERENCES

[1] D. J. Abadi, S. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *Proc. of SIGMOD'08*, 2008.
[2] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *Proc. of VLDB'06*, 2006.
[3] P. Agrawal and J. Widom. Confidence-aware join algorithms. In *Proc. of ICDE'09*, 2009.
[4] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proc. of ICDE'08*, 2008.
[5] L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *Proc. of SIGMOD'07*, 2007.
[6] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. of VLDB'06*, 2006.
[7] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *PVLDB*, 1(1), 2008.
[8] C. Binnig, S. Hildenbrand, and F. Faerber. Dictionary-based order-preserving string compression for main memory column stores. In *Proc. of SIGMOD'09*, 2009.
[9] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. *VLDB J.*, 16(1), 2007.
[10] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. S. Vitter, and Y. Xia. Efficient join processing over uncertain data. In *Proc. of CIKM'06*, 2006.
[11] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *Proc. of ICDE'09*, 2009.
[12] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4), 2007.
[13] R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.
[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.
[15] U. Güntzer, W.-T. Balke, and W. Kießling. Towards efficient multi-feature queries in heterogeneous environments. In *Proc. of ITCC'01*, 2001.
[16] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. of SIGMOD'08*, 2008.
[17] S. Idreos, M. Kersten, and S. Manegold. Self-organizing tuple reconstruction in column-stores. In *Proc. of SIGMOD'09*, 2009.
[18] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3), 2004.
[19] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
[20] M. Ivanova, M. Kersten, and N. Nes. An architecture for recycling intermediates in a column-store. In *Proc. of SIGMOD'09*, 2009.
[21] M. Ivanova, M. L. Kersten, and N. Nes. Self-organizing strategies for a column-store database. In *Proc. of EDBT'08*, 2008.
[22] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on unceratin streams. In *Proc. of VLDB'08*, 2008.
[23] F. Li, K. Yi, and J. Jestes. Ranking distributed probabilistic data. In *Proc. of SIGMOD'09*, 2009.
[24] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.
[25] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-*k* aggregation of ranked inputs. *ACM Trans. Database Syst.*, 32(3):19, 2007.
[26] A. Marian, N. Bruno, and L. Gravano. Evaluating top-*k* queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2), 2004.
[27] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *Proc. of VLDB'01*, 2001.
[28] H. Plattner. A common database approach for oltp and olap using an in-memory column database. In *SIGMOD'09*, pages 1–2, 2009.
[29] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. of ICDE'07*, 2007.
[30] C. Re and D. Suciu. Managing probabilistic data with mystiq: The can-do, the could-do, and the can't-do. In *Proc. of SUM'08*, 2008.
[31] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *Proc. of ICDE'06*, 2006.
[32] K. Schnaitter and N. Polyzotis. Evaluating rank joins with optimal cost. In *Proc. of PODS'08*, 2008.
[33] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-effort top-k query processing under budgetary constraints. In *Proc. of ICDE'09*, 2009.
[34] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE'07*, 2007.
[35] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Probabilistic top-*k* and ranking-aggregate queries. *ACM Trans. Database Syst.*, 33(3), 2008.
[36] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-store: A column-oriented dbms. In *VLDB*, pages 553–564, 2005.
[37] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-Relations. *IEEE Trans. Knowl. Data Eng.*, 20(12), 2008.