Robust Ranking of Uncertain Data

Da Yan and Wilfred Ng

The Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{yanda,wilfred}@cse.ust.hk

Abstract. Numerous real-life applications are continually generating huge amounts of uncertain data (e.g., sensor or RFID readings). As a result, top-k queries that return only the k most promising probabilistic tuples become an important means to monitor and analyze such data. These "top" tuples should have both high scores in term of some ranking function, and high occurrence probability. The previous works on ranking semantics are not entirely satisfactory in the following sense: they either require user-specified parameters other than k, or cannot be evaluated efficiently in real-time scale, or even generating results violating the underlying probability model. In order to overcome all these deficiencies, we propose a new semantics called U-Pop$k$ based on a simpler but more fundamental property inherent in the underlying probability model. We then develop an efficient algorithm to evaluate U-Pop$k$. Extensive experiments confirm that U-Pop$k$ is able to ensure high ranking quality and to support efficient evaluation of top-k queries on probabilistic tuples.

1 Introduction

Many emerging applications, such as environmental surveillance and mobile object tracking, involve the generation of uncertain data which are inherently fuzzy and noisy. As a result, various probabilistic DBMSs are developed to support the storage and querying of these uncertain data [2–4]. Since precise query expressions like SQL may not be ideal to evaluate such data, top-k queries become an important means to extract information from them.

Top-k queries on deterministic data have been well studied [5, 6]. Unlike top-k queries on deterministic data, ranking probabilistic tuples requires taking both the tuple score and its occurrence probability into account, which gives rise to new challenges in defining the query semantics. Despite the many recent attempts to study top-k query semantics in the context of probabilistic relations [1, 7–10], these seemingly natural semantics lead to quite different query results.

Recently, [11] proposes a unified framework that incorporates several of the semantics and gives an approach to learn the ranking function from user preference. [12] proposes to return a number of typical top-k results to users. While these works mitigate the inconsistency of the previous semantics and provide more flexibility, they exert extra burden on users by requiring their intervention.

The work of [1] proposes five intuitive properties for top-k queries, and shows that only ExpectedRank satisfies all of them and it is considerably more efficient

to evaluate than the previous semantics. However, ExpectedRank has two significant deficiencies: First, its results may contradict with the probability model, which will be detailed in Section 4. Second, its results deviate considerably from the results of the other semantics, which will be detailed in Section 6.1.

In this paper, we first identify a simple but fundamental property inherent in the probability model, which any robust ranking semantics for probabilistic data should satisfy. Then we describe our new semantics, U-Pop$k$, that is founded on this property to rank probabilistic tuples. It can be proved that U-Pop$k$ satisfies all the five properties of [1]. The efficiency of our evaluation algorithm and the ranking quality of U-Pop$k$ are evaluated using both real and synthetic datasets.

Compared to the state-of-the-art semantics, our proposal has many desirable features. First, U-Pop$k$ gives more reasonable results than ExpectedRank in general, with comparable evaluation cost. Second, unlike the work in [11, 12], U-Pop$k$ requires no user intervention except for the parameter k, and is thus easier to use. Finally, the evaluation of U-Pop$k$ takes considerably less time than other semantics, and thus U-Pop$k$ paves a much better way towards real time analyses.

The rest of the paper is organized as follows. Section 2 defines our probabilistic data model. We review the related work in Section 3. The robustness property and our ranking semantics are proposed in Section 4, and the corresponding algorithms are presented in Section 5. Extensive experiments are conducted in Section 6 to demonstrate the efficiency of our algorithm and the ranking quality of our semantics. Finally, we conclude the paper in Section 7.

2 Probabilistic Data Model

Among the many uncertain data models proposed in the literature, the *tuple-level probabilistic model* [1, 8–12] is one of the most important models. In this model, each tuple t is accompanied with the probability p of its occurrence. The model is able to capture the form of uncertain data that is common in many real life applications, such as sensor readings with confidence on their sensor states, and data tuples with confidence on their information sources. We adopt the tuple-level probabilistic model throughout the paper due to its popularity in real life applications.

Figure 1(a) shows our running example relation conforming to the tuple-level model, where the ranking score is defined according to the attribute "Speed", which records the car speed readings detected by different radars in a sampling moment. In this relation, a confidence field "Conf." is attached with each tuple to indicate its occurrence probability. The occurrence probability of t_1, denoted as $Pr(t_1)$, is 0.4. In contrast, the probability of the event that t_1 does not occur is given by $Pr(\neg t_1) = 1 - Pr(t_1) = 0.6$.

Note that both tuples t_2 and t_6 record the speed reading of the same car. Since a car can only have one speed in a given moment, t_2 and t_6 cannot co-exist, which we denote as $t_2 \oplus t_6$. We call such a constraint an *exclusion rule*. We have another *exclusion rule* $t_3 \oplus t_5$ defined in the relation for a similar reason. Different forms of constraints can be adopted in a probabilistic relation [3, 2] but

	Radar Location	Car Make	Plate No.	Speed	Confidence
t_1	L_1	Honda	X-123	130	0.4
t_2	L_2	Toyota	Y-245	120	0.7
t_3	L_3	Mazda	W-541	110	0.6
t_4	L_4	Nissan	L-105	105	1.0
t_5	L_5	Mazda	W-541	90	0.4
t_6	L_6	Toyota	Y-245	80	0.3

Exclusion Rules: $(t_2 \oplus t_6), (t_3 \oplus t_5)$

(a)

Possible World	Probability
PW^1={ t_1, t_2, t_4, t_5 }	0.112
PW^2={ t_1, t_2, t_3, t_4 }	0.168
PW^3={ t_1, t_4, t_5, t_6 }	0.048
PW^4={ t_1, t_3, t_4, t_6 }	0.072
PW^5={ t_2, t_4, t_5 }	0.168
PW^6={ t_2, t_3, t_4 }	0.252
PW^7={ t_4, t_5, t_6 }	0.072
PW^8={ t_3, t_4, t_6 }	0.108

(b)

Fig. 1. (a)Probabilistic Relation with Exclusion Rules (b)Possible World Space

exclusion rules are the most popular in the literature due to their simplicity and usefulness. We considered only exclusion rules in this paper.

Note that each exclusion rule corresponds to an entity (e.g. a car in our example) of the relation. Thus, each tuple appears in at most one exclusion rule. In our example, the rule $t_2 \oplus t_6$ means that the speed of the "Toyota" car takes the value 120 with probability 0.7 and 80 with probability 0.3. In general, given an exclusion rule $t_{i_1} \oplus t_{i_2} \oplus \cdots \oplus t_{i_m}$, we have (1) $Pr(t_{i_1}) + Pr(t_{i_2}) + \cdots + Pr(t_{i_m}) \leq 1$, and (2) at most one tuple in $\{t_{i_1}, t_{i_2}, \ldots, t_{i_m}\}$ can occur. There is still an event that no tuple in the rule occurs, which has probability $(1 - Pr(t_{i_1}) - Pr(t_{i_2}) - \ldots - Pr(t_{i_m}))$.

If a tuple t_i is independent of all other tuples, we say t_i itself is in a *trivial* rule. In addition, any two tuples from different rules are assumed to be independent. Thus, we have four rules in the example as follows: t_1, $t_2 \oplus t_6$, $t_3 \oplus t_5$ and t_4, where t_1 and t_4 are in trivial rules, and t_2 and t_3 are independent. This assumption simplifies the computation of the probabilities of possible worlds, which will be further elaborated when discussing the algorithm issues in Section 5.

In the tuple-level model, a possible world (PW) is a subset of the tuples in the probabilistic relation. Figure 1(b) shows the possible world space for the relation in Figure 1(a). The probability of each world is computed as the joint probability of the "occurrence events" of the tuples in the world, and the "absence events" of all the other tuples. For example, the probability of $PW^1 = \{t_1, t_2, t_4, t_5\}$ is $Pr(PW^1) = Pr(t_1) \times Pr(t_2) \times Pr(t_4) \times Pr(t_5) = 0.112$. Multiplication is used here because the occurrence events of t_1, t_2, t_4 and t_5 are independent of each other, and the absence events of t_3 and t_6 are already implied in the occurrence events of t_5 and t_2 due to the exclusion rules. Similarly, the probability of $PW^5 = \{t_2, t_4, t_5\}$ is $Pr(PW^5) = Pr(\neg t_1) \times Pr(t_2) \times Pr(t_4) \times Pr(t_5) = 0.168$.

3 Related Work

Several semantics for top-k queries on uncertain data have recently been proposed, such as U-Top$k$[8], U-kRanks[8], Global-Top$k$[9], PT-k[10] and Expecte-

dRank[1], all of which are defined on the possible world model. We now illustrate their semantics by performing a top-2 query on the relation in Figure 1(a).

1) U-Top$k$ returns the most probable top-k tuples that belong to a valid possible world. Consider U-Top2 and define $\langle t_i, t_j \rangle$ to be the event that t_i is ranked the first and t_j the second in a possible world. By merging the possible worlds in Figure 1(b) whose top-2 tuples are the same, we have $Pr(\langle t_1, t_2 \rangle) = Pr(PW^1) + Pr(PW^2) = 0.28$, which is the largest among all possible top-2 combinations. Therefore, the result of U-Top2 is $\langle t_1, t_2 \rangle$.

However, there can be a large number of valid possible worlds. As a result, the most probable top-k tuples that belong to a valid possible world can occur with a very small probability. [12] proposes to return c typical top-k tuple vectors in terms of the distribution of the total score of top-k tuples, from which users need to choose one, which is itself a non-trivial task for users.

2) U-kRanks returns the set of most probable top-$i\,th$ tuples across all possible worlds, where $i = 1, \ldots, k$. Let us compute U-2Ranks. First, consider the most probable tuple to appear in the 1st position. Tuple t_2 appears in the 1st position with probability $Pr(PW^5) + Pr(PW^6) = 0.42$, since it appears the first only in PW^5 and PW^6. Similarly, t_1 appears in the first position with probability $Pr(PW^1) + Pr(PW^2) + Pr(PW^3) + Pr(PW^4) = 0.4$. After considering all the tuples, we can see that t_2 appears in the $1st$ position with maximum probability. Thus the first answer to U-2Ranks is t_2. The second answer to U-2Ranks should be the most probable tuple to appear in the $2nd$ position, and similarly, we find that tuple t_3 appears in the $2nd$ position with maximum probability $Pr(PW^4) + Pr(PW^6) = 0.324$. To sum up, the result of U-2Ranks is $\langle t_2, t_3 \rangle$.

Since a tuple may be the most probable tuple to appear in more than one position, the same tuple may be listed multiple times in the result of U-kRanks, which is a very unnatural answer to users.

3) PT-k returns all tuples whose probability values of being in the top-k answers in possible worlds are above a threshold; **Global-Top$k$** returns k highest-ranked tuples according to their probability of being in the top-k answers in possible worlds.

As for Global-Top2 and PT-2, we check for each tuple the probability that it is within top-2. Tuple t_2 is within top-2 in worlds PW^1, PW^2, PW^5 and PW^6, and thus with probability $Pr(PW^1) + Pr(PW^2) + Pr(PW^5) + Pr(PW^6) = 0.7$. Similarly, the probability to be within top-2 is 0.4 for t_1, 0.432 for t_3, 0.396 for t_4, 0.072 for t_5 and 0 for t_6. Global-Top2 picks the two tuples with maximum probability to be within top-2, namely t_2 and t_3. On the other hand, PT-2 picks all the tuples with probability to be within top-2 higher than a pre-specified threshold. If the threshold is set to be 0.5, then only t_2 is returned. However, if the threshold is set to be 0.3, then the result would become $\{t_2, t_1, t_3, t_4\}$.

One limitation of PT-k is that the number of returned tuples may not be k but depends on the user-specified threshold, which is difficult for a user to set.

4) ExpectedRank(k) returns k tuples whose *expected ranks* across all possible worlds are the highest. However, if a tuple does not appear in a possible world, its rank is then undetermined. To solve this, in a possible world with m

tuples, ExpectedRank ranks absent tuples to be in the $(m+1)th$ position. Thus, t_2 and t_5 are both ranked $5th$ in PW^4 shown in Figure 1(b), although t_2 have both higher score and higher occurrence probability than t_5. As a more detailed illustration, we consider tuple t_5 which is ranked $4th$ in PW^1, $5th$ in PW^2 due to its absence, $3rd$ in PW^3, $5th$ in PW^4 due to its absence, $3rd$ in PW^5, $4th$ in PW^6 due to its absence, $2nd$ in PW^7, and $4th$ in PW^8 due to its absence. Therefore the expected rank for t_5 is $0.112 \cdot 4 + 0.168 \cdot 5 + 0.048 \cdot 3 + 0.072 \cdot 5 + 0.168 \cdot 3 + 0.252 \cdot 4 + 0.072 \cdot 2 + 0.108 \cdot 4 = 3.88$. Similarly, we have the expected rank 2.8 for t_1, 2.3 for t_2, 3.02 for t_3, 2.7 for t_4, 4.1 for t_6. Therefore, the result of ExpectedRank(2) is $\{t_2, t_4\}$, since their expected ranks are the highest.

In fact, ExpectedRank defines an order on the tuples, e.g. $t_2 \succ t_4 \succ t_1 \succ t_3 \succ t_5 \succ t_6$ in the previous example, while the ranking function of Global-Top$k$ and PT-k is dependent on k, which means that a tuple in the top-k result may not appear in the top-$(k+1)$ result.

5) PRF [11] presents a unified framework that uses *parameterized ranking functions* (PRF) for ranking probabilistic data. The work proposes a basic function called PRF^e and employs a linear combination of PRF^e functions to approximate PRF by using Discrete Fourier transformation. This approach is able to achieve good performance at the expense of result quality.

PRF also gives a learning algorithm that learns the parameter from user preference. The training data come from explicit user feedback, which assumes that users know the interplay between high score and high occurrence probability of the tuples. However, users usually expect reasonable score-probability tradeoff automatically in the evaluation of such top-k queries and assume no extra effort to give explicit feedback. Another problem is that high-quality training data may not be available from casual users, even if they are willing to give feedback.

4 Robust Ranking Semantics

In this section, we first formalize a property called *top-1 robustness*, which is founded on the tuple-level probability model. Then a fundamental property of ranking deterministic data, which is called *top-stability*, is extended to rank probabilistic tuples. Top-stability enables repeated applications of *top-1 robustness*, based on which we define our new semantics.

Property 1 (Top-1 Robustness). The top-1 query on an uncertain relation D returns the tuple $t \in D$ such that $\forall t' \in D, Pr(r(t) = 1) \geq Pr(r(t') = 1)$, where $r(t)$ denotes the rank of t.

From now on, we will use "top-1 probability" to denote the probability for a tuple to be ranked top-1. Property 1 states that any top-k query semantics for probabilistic tuples should return the tuple with maximum top-1 probability when $k = 1$. Note that the semantics of U-Top1, U-1Ranks and Global-Top1 are equivalent, and they all satisfy Property 1. Although the number of tuples returned by PT-1 is determined by the threshold, the top-1 tuple defined in Property 1 must appear in the result of PT-1 if the result is not empty.

Unfortunately, ExpectedRank may violate this robustness property. Consider an example relation $\{t_1, t_2, t_3, t_4, t_5\}$ with an exclusion rule $t_1 \oplus t_2 \oplus t_3 \oplus t_5$, where tuples t_1 to t_5 are already sorted in descending order of their scores. We use p_i to denote the occurrence probability of t_i, and $p_1 = p_2 = p_3 = 0.2, p_4 = 0.45, p_5 = 0.4$. For top-1 query on this relation, t_5 should be returned since its top-1 probability $(1-p_4)p_5 = 0.22$ is the highest. However, ExpectedRank picks t_4 whose top-1 probability $(1-p_1-p_2-p_3)p_4 = 0.18$ is smaller, which contradicts Property 1 and thus the underlying probability model.

Property 2 (Top-Stability). The top-$(i+1)th$ tuple should be the top-$1st$ after the removal of the top-i tuples.

Property 2 is intuitive in the context of certain data. *Top-stability* implies that, in principle, we are able to adopt the following approach to obtain the top-k tuples: The top-1 tuple is repeatedly removed from the current relation until k tuples are obtained. To generalize this approach to ranking probabilistic tuples for answering a top-k query, we define a new semantics U-Pop$k$ as follows:

Definition 1 (U-Pop$k$). *Tuples are picked in order from a relation according to Property 2 until k tuples are picked, where the top-1st tuple is defined according to Property 1.*

According to Definition 1, when a tuple is picked as the result, it is removed from the relation and thus will never be considered again in later evaluation. This avoids the problem of multiple occurrences of the same tuple in the result.

Although U-Pop$k$ changes the probabilistic relation in each round of evaluation and hence the set of possible worlds, this enables the use of *top-1 robustness* to pick the result tuple in each round. If k is small compared with the size of the relation, which is not unusual in applications, the modified relation in each round is still a good approximation of the original one. Besides, by removing the "top" tuples from the relation, we only need to make comparison among the remaining tuples in the pool from which the next "top" tuple will be picked.

Our approach shares a similar spirit of the work that uses a simplification assumption to facilitate the application of a robust property. For example, a naïve Bayes classifier uses the simplification assumption that all observations are independent to the facility of the evaluation of the robust property (i.e., the Bayes' rule), and so does maximum likelihood estimation. Even in top-k queries on uncertain data, we have the independence assumption among tuples from different exclusion rules to simplify the computation of the probability of possible worlds, where the possible world model is robust. For U-Pop$k$, the simplification property is *top-stability* and the robust property is *top-1 robustness*.

Tuples are assumed to be pre-sorted in the descending order of tuple scores in all previous work, since a tuple t' with a score lower than another tuple t will be ranked lower in any possible world and thus has no influence on the probability computation for the rank of t, which is also adopted in U-Pop$k$. Recall that the tuples in Figure 1(a) are already pre-sorted according to the attribute speed. The following example illustrates how U-Pop2 works for the relation in Figure 1(a):

Example The first step is to compute the top-1 tuple in the relation. Tuple t_1 is ranked the first with probability $Pr(t_1) = 0.4$. The probability is $Pr(\neg t_1)Pr(t_2) = 0.42$ for t_2, since t_1 must not occur and t_2 must occur in this case, while the other tuples are immaterial. Since the probability that a tuple other than t_1 and t_2 is ranked the first is $Pr(\neg t_1)Pr(\neg t_2) = 0.18$, which is smaller than the probability for t_2 to be ranked the first, we can conclude that t_2 is ranked the first with maximum probability and is thus picked.

After removing t_2 from the relation, we have the tuples t_1, t_3, t_4, t_5, t_6 remained in the pool. Tuple t_1 is ranked the first with probability $Pr(t_1) = 0.4$. The probability is $Pr(\neg t_1)Pr(t_3) = 0.36$ for t_3. Since the probability that a tuple other than t_1 and t_3 is ranked the first is $Pr(\neg t_1)Pr(\neg t_3) = 0.24$, which is smaller than the probability for t_1 to be ranked the first, we thus conclude that t_1 is ranked the first with maximum probability and it is thus picked.

Therefore, the result for U-Pop2 on the relation in Figure 1(a) is $\langle t_2,\ t_1 \rangle$.

5 U-Pop*k* Algorithms

In this section, we first present the algorithms to evaluate U-Pop*k* for the case that all tuples are independent of each other (i.e., no exclusion rule is considered). Then, we extend them to handle the general case that *exclusion rules* are given.

5.1 Algorithm for Independent Tuples

We consider the special case where all tuples in a given probabilistic relation are independent of each other, and assume that tuples are pre-sorted in descending order of score.

Consider a relation having tuples $\{t_1, t_2, \ldots, t_n\}$, where t_1 to t_n are already sorted and p_i is the occurrence probability of t_i. In order to rank t_i as top-1, t_1 to t_{i-1} should not appear but t_i should appear, while all tuples after t_i (i.e., tuples with lower scores) are immaterial. Therefore, the top-1 probability of t_i is $(1 - p_1)(1 - p_2) \cdots (1 - p_{i-1})p_i$.

If we define $accum_i = (1 - p_1)(1 - p_2) \cdots (1 - p_{i-1})$ with the special case of $accum_1 = 1$, we have $accum_{i+1} = accum_i(1 - p_i)$, and the probability for t_i to be top-1 can be written as $accum_i \cdot p_i$.

Algorithm 1 Find the Top-1 Tuple

```
1: accum ⟵ 1;  max ⟵ −∞;  result ⟵ null
2: while accum > max and there are more tuples do
3:    {Process the next tuple t_i}
4:    top1Prob ⟵ accum · p_i
5:    if top1Prob > max then
6:       max ⟵ top1Prob;  result ⟵ t_i
7:    accum ⟵ accum · (1 − p_i)
8: return result
```

Algorithm 1 finds the top-1 tuple among a list of pre-sorted tuples. The parameter *accum* is initialized to 1 in Line 1, and updated after each iteration (Line 7). Lines 2–7 check the tuples one by one, and in each iteration a tuple t_i is read and its probability to be top-1 is computed as *top1Prob* (Line 4). If *top1Prob* is found to be larger than the maximum top-1 probability currently found (Line 5), i.e., max, it is updated and t_i is recorded (Line 6).

Note that we do not need to check all the tuples. Suppose we have checked t_i and updated *accum* to $(1-p_1)(1-p_2)\cdots(1-p_i)$, where the current maximum top-1 probability is max. If $accum \leq max$, then the tuple with top-1 probability equal to max must be the result. This is because the top-1 probability of any succeeding tuple t_j(for $j > i$) is $(1-p_1)(1-p_2)\cdots(1-p_i)(1-p_{i+1})\cdots(1-p_{j-1})p_j \leq (1-p_1)(1-p_2)\cdots(1-p_i)\cdot 1\cdots 1\cdot 1 \leq max$.

Intuitively, the parameter *accum* acts as an upperbound of the top-1 probabilities for the tuples to be checked, which enables early termination (Line 2).

It is straightforward to construct a naïve algorithm for U-Pop$k$ that uses Algorithm 1: All the sorted tuples are read into a memory *buffer* first. Then in each iteration, a top-1 tuple is picked from the current tuple *buffer* using Algorithm 1, removed from the *buffer* and added to the result set. This is repeated until k tuples are obtained.

However, the naïve approach is not efficient enough and can be much improved by reusing the parameters obtained from previous computation. To illustrate this, we suppose that t_3 have been checked, that the iteration stops due to $accum \leq max$, and that t_2 is found to have maximum top-1 probability. Then after removing t_2, the top-1 probability of t_1 is still p_1, which can be reused, while that of t_3 is now $(1-p_1)p_3$ rather than $(1-p_1)(1-p_2)p_3$.

In general, we can reuse the already computed top-1 probabilities of those tuples whose positions are before the picked top-1 tuple t_{top1}, since the removal of t_{top1} does not change their top-1 probabilities. Figure 2 shows an example where t_i is picked due to $accum \leq max$, after checking t_j and updating *accum*. The top-1 probabilities for t_1 to t_{i-1} can be reused, t_i is removed from the buffer, and the top-1 probabilities for t_{i+1} to t_j should be updated (i.e., re-scanned). The update is simply to divide the original probability by $(1-p_i)$ so as to rule out the consideration for t_i. After the update, the next iteration starts from t_{j+1}. We define $(j-i)$ to be the *rescan length* for this iteration.

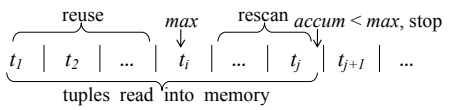

Fig. 2. Snapshot of the End of an Iteration

In order to delete the picked top-1 tuple from the memory buffer in $O(1)$ time for each iteration, we organize the buffer as a doubly-linked list and attach each tuple t in the buffer with the following three fields: (1) $t.prob$: its top-1

probability, (2) $t.id$: its index in the original sorted tuple list, and (3) $t.max$: the tuple whose top-1 probability is the maximum before t being put in the buffer.

Note that if t_i is picked as top-1, $t_i.max$ already records the tuple with maximum top-1 probability among the tuples whose positions are before t_i in the buffer (i.e., the current maximum top-1 probability for the tuples with position before t_{i+1}). While updating the top-1 probabilities from t_{i+1}, we update the current maximum top-1 probability if the updated probability is larger. Therefore, after all the updates, we get the current maximum top-1 probability for all the tuples with position before t_{j+1}. Then, the next iteration starts from t_{j+1}.

Algorithm 2 shows the process of top-1 probability adjustment between consecutive iterations discussed above, where t_i has already been identified as top-1.

Algorithm 2 Top-1 Probability Adjustment between Iterations

```
1: maxTuple ⟵ t_i.max
2: if maxTuple == null then
3:    max ⟵ −∞
4: else
5:    max ⟵ maxTuple.prob
6: for each tuple t after t_i in buffer do
7:    t.prob ⟵ t.prob/(1 − p_i);  t.max ⟵ maxTuple
8:    if t.prob > max then
9:       max ⟵ t.prob;  maxTuple ⟵ t
10: Delete t_i from buffer
```

The variable $maxTuple$ records the tuple with the maximum top-1 probability currently, and max is its probability. Before updating from tuple t_{i+1} (Line 6), $maxTuple$ is set to $t_i.max$. However, $t_i.max$ may be $null$, because for t_1, no tuple with the maximum top-1 probability exists before it. In this case, max is set to $-\infty$. Otherwise, it is set to $maxTuple.prob$ which is the maximum top-1 probability among the tuples before t_i.

For each tuple t after t_i, $t.prob$ is divided by $(1 - p_i)$ and $t.max$ records $maxTuple$ (Line 7). If $t.prob > max$, t now has the maximum top-1 probability, and thus $maxTuple$ and max are updated accordingly (Line 9). Finally, t_i is removed from the doubly-linked list *buffer* in $O(1)$ time in Line 10.

Algorithm 3 is a more efficient algorithm for U-Pop$k$ on independent tuples, which makes use of Algorithm 2. After the initialization in Lines 1–2, tuples are read into the memory *buffer* one by one in Line 3. For each tuple, its fields are set in Line 4, and it is added to the end of *buffer* (Line 5). If its top-1 probability is larger than the current maximum, the current maximum is updated accordingly (Lines 6–7). Then $accum$ is updated in Line 8 and checked in Line 9. If $accum \leq max$, the current top-1 is found to be $maxTuple$, which is thus put into the result set (Line 10). Then Algorithm 2 is called to update *buffer* to reflect the removal of $maxTuple$ (Line 13), and so does $accum$ (Line 14).

Algorithm 3 U-Pop$k$ Algorithm for Independent Tuples

```
1: Create an empty doubly-linked list buffer
2: accum ⟵ 1;  resultSet ⟵ φ;  maxTuple ⟵ null;  max ⟵ −∞
3: for each tuple t_i do
4:    t_i.prob ⟵ accum · p_i;  t_i.max ⟵ maxTuple
5:    Append t_i to the end of the doubly-linked list buffer
6:    if t_i.prob > max then
7:       max ⟵ t_i.prob;  maxTuple ⟵ t_i
8:    accum ⟵ accum · (1 − p_i)
9:    if accum ≤ max then
10:      Put maxTuple into resultSet
11:      if |resultSet| == k then
12:         return resultSet
13:      accum ⟵ accum/(1 − p_maxTuple.id)
14:      Call Algorithm 2 to adjust buffer
15: while |resultSet| < k and buffer is not empty do
16:    Put maxTuple into resultSet
17:    if |resultSet| == k then
18:       return resultSet
19:    accum ⟵ accum/(1 − p_maxTuple.id)
20:    Call Algorithm 2 to adjust buffer
21: return resultSet
```

If k results are picked, we do not need to read in more tuples (Lines 11–12). However, it is possible that all tuples are read into *buffer* before k results are picked. Thus, after the for loop in Lines 3–14, if there are still less than k results, we need to pick $maxTuple$ into the result set (Line 16), and use Algorithm 2 to adjust the buffer and set the next $maxTuple$ (Line 19). This process is repeated until k results are picked, which is done by the while loop in Lines 15–20.

At most n tuples are read into *buffer* where n is the total number of tuples in the probabilistic relation. Since all the steps in the for loop in Lines 3–14 take constant time except the adjustment in Line 13, they take $O(n)$ time in total. The adjustment of Lines 13 and 19 is executed exactly k times, since whenever a top-1 tuple is picked, Lines 14–19 or 23–28 are executed once, and thus Algorithm 2. It is straightforward to see that Algorithm 2 takes $O(L)$ where L is the *rescan length* for an iteration. The time complexity of Algorithm 3 is then $O(n + k \cdot L_{avg})$, where L_{avg} is the average *rescan length* among the k iterations.

Note that L_{avg} tends to be small since every factor of $accum$ is at most 1, which makes $accum$ smaller than max after a few tuples are read into *buffer*. For small k, it is not likely that all n tuples are read into *buffer*, but just a few top ones, the number of which is defined to be *scan depth* in [8].

The tricky case is when $p_i = 1$, where $accum$ will be updated to 0 in Line 8, and thus $accum \leq max$ and Algorithm 2 is then called in Line 13. So, t_{i+1} can never be read into *buffer* until t_i is picked, and therefore Line 7 in Algorithm 2 is not executed and it does not cause division by 0. However, we cannot restore $accum$ using Line 14. Instead, if $p_i = 1$, we save $accum$ before executing Line 8

for later restoration when t_i is picked. The saved value is updated using Line 14, if other tuples are picked before the restoration. These details are not included in Algorithm 3 in order to make it more readable.

5.2 Algorithm for Tuples With Exclusion Rules

Now, we present the general case where each tuple is involved in an exclusion rule $t_{i_1} \oplus t_{i_2} \oplus \cdots \oplus t_{i_m} (m \geq 1)$. We assume that $t_{i_1}, t_{i_2}, \ldots, t_{i_m}$ in the rule are already pre-sorted in descending order of the tuple scores.

The upper bound of the top-1 probability for all the tuples starting from t_i is no longer $accum_i = (1-p_1)(1-p_2)\cdots(1-p_{i-1})$. We denote $t_{j_1}, t_{j_2}, \ldots, t_{j_\ell}$ to be all the tuples with position before t_i and in the same exclusion rule of t_i. Then, there is a factor $(1 - p_{j_1} - p_{j_2} - \ldots - p_{j_\ell})$ in $accum_i$. Let us define $accum_{i+1} = accum_i \cdot (1 - p_{j_1} - p_{j_2} - \ldots - p_{j_\ell} - p_i)/(1 - p_{j_1} - p_{j_2} - \ldots - p_{j_\ell})$, which changes the factor of t_i's rule, in the top-1 probabilities of the tuples after t_i, from $(1-p_{j_1}-p_{j_2}-\ldots-p_{j_\ell})$ to $(1-p_{j_1}-p_{j_2}-\ldots-p_{j_\ell}-p_i)$. Now the top-1 probability of t_i is $accum_i \cdot p_i/(1 - p_{j_1} - p_{j_2} - \ldots - p_{j_\ell})$, where the denominator rules out the influence of the tuples exclusive with t_i.

Consider the factor in $accum_i$ that corresponds to rule $t_{j_1} \oplus t_{j_2} \oplus \cdots \oplus t_{j_\ell} \oplus \cdots \oplus t_{j_m}$. If the tuples $t_{j_1}, t_{j_2}, \ldots, t_{j_\ell}$ $(\ell < m)$ are now before the current tuple considered, then the factor is $(1-p_{j_1}-p_{j_2}-\ldots-p_{j_\ell})$. Within the same iteration, this factor can only be decreasing as we read in more tuples. For example, if we read in more tuples such that $t_{j_{\ell+1}}$ is also positioned before the current tuple, the factor then becomes $(1 - p_{j_1} - p_{j_2} - \ldots - p_{j_\ell} - p_{j_{\ell+1}})$, which is smaller.

The top-1 probability for t_i is no longer $accum_i \cdot p_i < accum_i$, but $accum_i \cdot p_i/(1-p_{j_1}-p_{j_2}-\ldots-p_{j_\ell})$ to rule out the factor corresponding t_i's rule. Therefore, if we keep track of the factors corresponding to all the rules, where $factor_{min}$ is the smallest, the top-1 probability upper bound is $\mathbf{accum_i/factor_{min}}$ for all the tuples starting from t_i. This is due to two reasons. First, the top-1 probability of t_i is computed as $\frac{accum_i}{factor} \times p_i (< \frac{accum_i}{factor_{min}})$, where $factor$ is the factor corresponding to t_i's rule. Second, the factors in $accum$ that correspond to the rules can only decrease as i increases. We organize the rules in the memory by using *MinHeap* on $factor$. Thus, $factor_{min}$ can be retrieved from the top of the heap immediately, and the upper bound can be computed in $O(1)$ time. We call this MinHeap *Active Rule set* and denote it AR.

Note that we do not need to keep all the rules in AR. If all the tuples in a rule are *after* the current tuple, we do not need to fetch it into AR. Otherwise, if the rule of the current tuple is not in AR, we insert it into AR, which takes $O(\log |AR|)$ time. For each rule r in AR, we attach it with the following two fields: (1) $r.pivot$: the last tuple in rule r that is before the current tuple, and (2) $r.factor$: the current factor of r in $accum$. Suppose $r = t_{i_1} \oplus t_{i_2} \oplus \cdots \oplus t_{i_\ell} \oplus \cdots \oplus t_{i_m}$ and $r.pivot = t_{i_\ell}$, then $r.factor = (1 - p_{i_1} - p_{i_2} - \ldots - p_{i_\ell})$.

After checking a tuple and updating $accum$, if the upper bound is smaller than the current maximum top-1 probability, the tuple must be top-1 and thus the iteration terminates. Before next iteration, we need to update the top-1

probabilities of the tuples after the picked one. In this case, we need to update their probabilities segment by segment as illustrated in Figure 3.

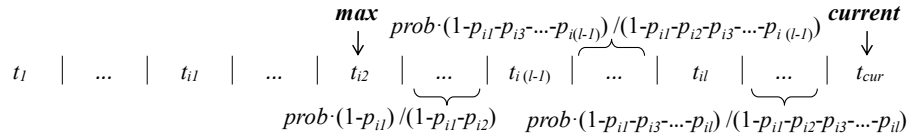

Fig. 3. Top-1 Probability Adjustment

In Figure 3, we assume that after processing the current tuple t_{cur}, the upper bound is smaller than tuple t_{i_2}'s top-1 probability, which is the current maximum, and thus the iteration ends. Besides, assume that $t_{i_1}, \ldots, t_{i_l}$ are the tuples in t_{i_2}'s rule and they are positioned before t_{cur}. Then the top-1 probability *prob* of a tuple after t_{i_2} and between t_{i_h} and $t_{i_{h+1}}$ (defined to be a segment) in the *buffer* should be updated as $prob \cdot (1 - p_{i_1} - p_{i_2} - p_{i_3} - p_{i_h})/(1 - p_{i_1} - p_{i_3} - p_{i_h})$ to reflect the removal of t_{i_2} from *buffer*.

This actually changes the factor of t_{i_2}'s rule in the top-1 probabilities, which can be done by a single pass of the tuples after t_{i_2}, and a single pass over t_{i_2}'s rule. Note that the top-1 probabilities of $t_{i_1}, t_{i_3}, \ldots, t_{i_l}$ remain the same.

After the removal of the current top-1 tuple (i.e., t_{i2} in the above example) and the update of the top-1 probabilities, the tuple is also removed from its rule r. This increases $r.factor$ (i.e., by p_{i2} in the above example), and so r's position in the MinHeap AR should be adjusted, which takes $O(\log |AR|)$ time. If no tuple remains in a rule after the removal, the rule is deleted from AR.

Since the adjustment after each iteration includes $O(\log |AR|)$ time rule position adjustment in AR and a scan of the rule of the picked tuple, the total time is $O(k(\log |AR| + len_{max}))$, where len_{max} is the largest length of a rule. Besides, since each rule will be inserted into AR at most once in $O(\log |AR|)$ time, the total time is $O(|R| \log |AR|)$, where R is the rule set. So the overall time complexity is $O(n + |R| \log |AR| + k(L_{avg} + len_{max} + \log |AR|))$. We do not present the complete algorithm here due to space limitation.

6 Experiments

We conduct extensive experiments using both real and synthetic datasets on HP EliteBook with 3 GB memory, and 2.53 Hz Intel Core2 Duo CPU. The real dataset is IIP Iceberg Sightings Databases[1], which is commonly used by the related work such as [10, 11] to evaluate the result quality of ranking semantics. We study the performance of the algorithms on synthetic datasets. Our algorithms are implemented in Java.

[1] http://nsidc.org/data/g00807.html (IIP: International Ice Patrol)

6.1 Ranking Quality Comparison on IIP Iceberg Databases

The pre-processed verion of IIP by [10] is used to evaluate the ranking quality of different semantics. Figure 4(a) lists the occurrence probabilities of some tuples from the dataset, where the tuples are pre-sorted by scores. Figure 4(b) shows the top-10 query results of different semantics.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{14}	t_{18}
p_i	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

(a) Occurrence Probabilities of the Tuples

	1	2	3	4	5	6	7	8	9	10
U-Pop$k$	t_1	t_2	t_3	t_4	t_5	t_6	t_9	t_7	t_{10}	t_{11}
ExpRank	t_1	t_2	t_3	t_5	t_6	t_9	t_{11}	t_{18}	t_{23}	t_{33}
PT-k	t_1	t_2	t_3	t_4	t_5	t_6	t_9	t_{10}	t_{11}	t_{14}
U-Top$k$	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_9	t_{10}	t_{11}
U-kRanks	t_1	t_2	t_3	t_5	t_6	t_9	t_9	t_{11}	t_{11}	t_{18}

(b) Occurrence Probabilities of the Tuples

	U-Pop$k$	ExpRank	PT-k	U-Top$k$	U-kRanks	SUM
U-Pop$k$	0	13.5	3	1	5	22.5
ExpRank	13.5	0	13.5	12.5	2.5	42
PT-k	3	13.5	0	2	5	23.5
U-Top$k$	1	12.5	2	0	4	19.5
U-kRanks	5	2.5	5	4	0	16.5

(c) Neural Approach to Kendall's Tau Distance

	U-Pop$k$	ExpRank	PT-k	U-Top$k$	U-kRanks	SUM
U-Pop$k$	0	10	2	1	4	17
ExpRank	10	0	9	9	0	28
PT-k	2	9	0	1	3	15
U-Top$k$	1	9	1	0	3	14
U-kRanks	4	0	3	3	0	10

(d) Optimistic Approach to Kendall's Tau Distance

Fig. 4. Top-10 Results on IIP Iceberg Databases

Figure 4(b) shows that the results of U-Pop$k$, U-Top$k$ and PT-k are almost the same. U-Pop$k$ ranks t_9 before t_7, which is more reasonable, since $p_7 = 0.4$ is much smaller than $p_9 = 0.8$ and their score ranks are close. PT-k rules out t_7 but includes t_{14}. However, $p_{14} = 0.6$ is not much larger than $p_7 = 0.4$ but t_7 has a much higher score rank than t_{14}. Therefore, it is more reasonable to rank t_7 before t_{14}. In fact, the top-11th tuple for U-Pop$k$ is t_{14}. However, U-kRanks returns duplicate tuples. ExpectedRank promotes low-score tuples like t_{23} and t_{33} to the top, which is unreasonable and is also observed in [11]. This deficiency happens mainly because ExpectedRank assigns rank $(\ell + 1)$ to an absent tuple t in a world having ℓ tuples. As a result, low-score absent tuples are given relatively high ranking in those small worlds, leading to their overestimated rank. Overall, U-Pop$k$ gives the most reasonable results in this experiment.

Kendall's tau distance is extended to gauge the difference of two top-k lists in [14], which includes an optimistic approach and a neural approach. Figure 4(c) and (d) show the extended Kendall's tau distance between the top-10 lists of the different semantics in Figure 4(b), where the last column SUM is the sum of the distances in each row. We can see that ExpectedRank returns drastically different results from the other semantics, which means that ExpectedRank actually generates many unnatural rankings.

6.2 Scalability Evaluation

We find that only U-Pop$k$ and ExpectedRank are efficient enough to support real time needs, while other semantics such as PT-k are much more expensive to evaluate. Figure 5(a1)–(a2) show the results on the IIP dataset described above, where 4 seconds are consumed by PT-k even when $k = 100$ and the high

probability threshold 0.5, while all the tuples can be ranked by U-Pop$k$ and ExpectedRank within 0.2 seconds. (a2) also shows that U-Pop$k$ is faster than ExpectedRank when k is within 1/10 of the data size and never slower by a factor of 2. Overall, the efficiency of U-Pop$k$ is comparable to that of ExpectedRank.

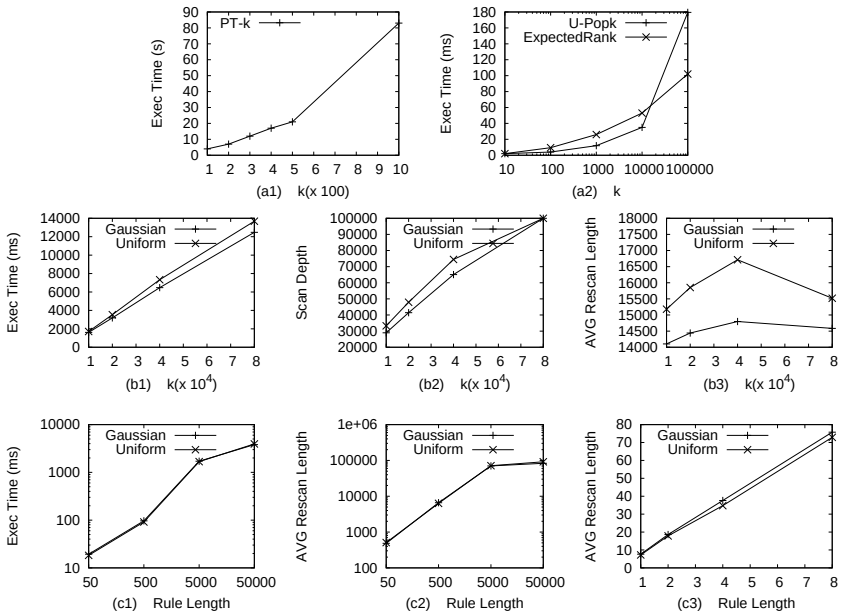

Fig. 5. Important Scalability Evaluation on Synthetic Data

We evaluate the scalability of U-Pop$k$ on synthetic data sets and show the important results in Figure 5. We find that the data size is not a major factor on the performance, and therefore we fix it to be 100K tuples. ExpectedRank is also not shown here, since it has similar performance in Figure 5(b1). The results in each experiment are averaged over 10 randomly generated data sets, whose tuple probabilities conform to either Uniform or Gaussian distribution.

Figures 5(b1)–(b3) show the effect of k on the performance, where the rule length is set to be 1000. Figures 5(b1)–(b2) show that the execution time and scan depth are almost linear to k, while in Figure 5(b3) the average rescan length first increases and then decreases as k increases. The drop happens because almost all the tuples are read into *buffer* at the end, and therefore the number of tuples decreases for each removal of a top-1 tuple.

Figures 5(c1)–(c2) show the effect of rule length on the performance, where we set all rules to the same length. Since in real life applications it is not likely to have too many tuples in a rule, we also explore the effect of rule length for short rules (i.e. less than 10 tuples in a rule). Figure 5(c3) shows that the average rescan length increases linearly with the rule length for small rule lengths.

The time complexity of U-Pop$k$ is shown to be $O(n + |R| \log |AR| + k(L_{avg} + len_{max} + \log |AR|))$ in Section 5.2, where the $O(n)$ part is actually the *scan depth*, which is shown to be almost $O(k)$ in Figure 5(b2). Also, when the maximum rule length len_{max} is small, we have $L_{avg} = O(len_{max})$ in Figure 5(c3). Since AR is a fraction of the rule set, we have $|AR| = O(|R|)$. Thus, the time complexity can be approximated as $O(k \cdot (len_{max} + \log |R|) + |R| \log |R|)$, that is linear to k and len_{max}, and $O(|R| \log |R|)$ for the rule set R, which scales well.

7 Conclusion

We propose U-Pop$k$ as a new semantics to rank probabilistic tuples, which is based on a robust property inherent in the underlying probability model. Compared with other known ranking semantics, U-Pop$k$ is the only semantics that is able to achieve all the following desirable features: high ranking quality, fast response time, and no additional user-defined parameters other than k.

Acknowledgements. This work is partially supported by RGC GRF under grant number HKUST 618509.

References

1. G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, 2009.
2. N. Dalvi, and D. Suciu, Efficient query evaluation on probabilistic databases, *VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.
3. P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, Trio: A system for data, uncertainty, and lineage, in *VLDB*, 2006.
4. L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *SIGMOD*, 2007.
5. R. Fagin, A. Lotem, and M. Naor, Optimal aggregation algorithms for middleware, in *PODS*, 2001.
6. I. F. Ilyas, G. Beskales, and M. A. Soliman, Survey of top-k query processing techniques in relational database systems, *ACM Computing Surveys*, 2008.
7. C. Re, N. Dalvi, and D. Suciu, Efficient top-k query evaluation on probabilistic databases, in *ICDE*, 2007.
8. M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, Top-k query processing in uncertain databases, in *ICDE*, 2007.
9. X. Zhang, and J. Chomicki, On the semantics and evaluation of top-k queries in probabilistic databases, in *DBRank*, 2008.
10. M. Hua, J. Pei, W. Zhang, and X. Lin, Ranking queries on uncertain data: A probabilistic threshold approach, in *SIGMOD*, 2008.
11. J. Li, B. Saha, and A. Deshpande, A unified approach to ranking in probabilistic databases, in *VLDB*, 2009.
12. T. Ge, S. Zdonik, and S. Madden, Top-k queries on uncertain data: On score distribution and typical answers, in *SIGMOD*, 2009.
13. C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. In *VLDB*, 2008.
14. R. Fagin, R. Kumar, and D. Sivakumar, Comparing top k lists. In *SODA*, 2003.