Efficient Processing of Top-k Queries in Uncertain Databases

Ke Yi[†] Feifei Li[‡] Divesh Srivastava[†] George Kollios[‡]

[†]AT&T Labs, Inc.
{yike, divesh}@research.att.com

[‡]Department of Computer Science
Boston University
{lifeifei, gkollios}@cs.bu.edu

Abstract

This work introduces new algorithms for processing top-k queries in uncertain databases, under the generally adopted model of x-relations. An x-relation consists of a number of x-tuples, and each x-tuple randomly instantiates into one tuple from one or more alternatives. Soliman et al. [25] first introduced the problem of top-k query processing in uncertain databases, but their solutions are highly inefficient. Under the x-relation model, our new results significantly improve the state of the art, in terms of both running time and memory usage. In the single-alternative case, our new algorithms are 2 to 3 orders of magnitude faster than the previous algorithms. In the multi-alternative case, the improvement is even more dramatic: while the previous algorithms have exponential complexity in both time and space, our algorithms run in near linear or low polynomial time. Our study covers both types of top-k queries proposed in [25]. We provide both the theoretical analysis and an extensive experimental evaluation to demonstrate the superiority of the new approaches over existing solutions.

1. Introduction

Uncertain databases have received a lot of attention recently due to the large number of applications that require management of uncertain and/or fuzzy data. Examples of such applications include: data integration [15], data cleaning [14, 7, 16], and mobile and sensor data management [11, 8], just to name a few. It is interesting to note that some important works on this topic appeared sporadically in the last two decades, including possible world semantics and probabilistic databases [1, 12, 17, 5, 18]. However, only recently we witness a more systematic and persistent effort to address uncertainty data management issues such as data modeling and representation [13, 3, 21, 6], general query processing [10, 8, 22], indexing [26, 23, 20], and development of query languages [4].

The uncertain data model. Quite a few uncertain data models have been proposed in the literature [21, 6, 2, 10], trying to represent the probability distribution of all the possible instances of the database. They range from the basic model in which each tuple appears with a certain probability independently, to powerful models that are *complete*, i.e., models that can represent any probability distribution of the database instances. However, complete models have exponential complexities and are hence infeasible to handle efficiently, so some extensions to the basic model have been introduced to expand the expressiveness of the model while keeping computation tractable. Notably, in the TRIO [2] system, an uncertain data set, which they call an *x-relation*, consists of a number of *x-tuples*. Each x-tuple includes a number of alternatives, associated with probabilities, which represent a discrete probability distribution of these alternatives being selected. Independence is still assumed among the x-tuples. This model has been frequently used in the study of uncertain databases as it is a reasonable approximation of the uncertain nature of the data.

In this paper, we also adopt the x-relation model, augmented with a score attribute, on which we rank the tuples. More precisely, each *tuple* t consists of four components: a unique identifier $id(t)$, a *score* $s(t)$, a *confidence* $p(t)$ that is the probability of t appearing in the database, and all the other attributes $A(t)$. An *x-tuple* τ is a set of tuples (up to a constant number), subject to the constraint that $\sum_{t_i \in \tau} p(t_i) \leq 1$. These t_i's are called the *alternatives* of τ. An x-tuple represents a discrete probability distribution of the possible values τ may make in a randomly instantiated database, i.e., τ takes t_i with probability $p(t_i)$, for $i = 1, \ldots, |\tau|$[1], or does not appear at all with probability $1 - \sum_{i=1}^{d} p(t_i)$[2] We define an *uncertain database* $\mathcal{D}$ as a collection of M pairwise disjoint x-tuples. We use D to denote the set of all tuples in $\mathcal{D}$, and let $|D| = \sum_{\tau \in \mathcal{D}} |\tau| = N$. Without loss of generality, we assume that all scores are dis-

[1] $|\tau|$ is the number of alternatives in τ.

[2] We denote the number of alternatives for an x-tuple τ as $d = |\tau|$.

tuples	$s(t)$	$p(t)$
t_1	100	0.5
t_2	92	0.4
t_3	80	0.6
t_4	70	0.3

x-tuples	
τ_1	$\{t_1, t_4\}$
τ_2	$\{t_2\}$
τ_3	$\{t_3\}$

world W	$\Pr[W]$
$\emptyset$	$(1-p(t_1)-p(t_4))(1-p(t_2))(1-p(t_3)) = .048$
$\{t_1\}$	$p(t_1)(1-p(t_2))(1-p(t_3)) = .12$
$\{t_2\}$	$p(t_2)(1-p(t_1)-p(t_4))(1-p(t_3)) = .032$
$\{t_3\}$	$p(t_3)(1-p(t_1)-p(t_4))(1-p(t_2)) = .072$
$\{t_4\}$	$p(t_4)(1-p(t_2))(1-p(t_3)) = .072$
$\{t_1, t_2\}$	$p(t_1)p(t_2)(1-p(t_3)) = .08$
$\{t_2, t_4\}$	$p(t_2)p(t_4)(1-p(t_3)) = .048$
$\{t_1, t_3\}$	$p(t_1)p(t_3)(1-p(t_2)) = .18$
$\{t_3, t_4\}$	$p(t_3)p(t_4)(1-p(t_2)) = .108$
$\{t_2, t_3\}$	$p(t_2)p(t_3)(1-p(t_1)-p(t_4)) = .048$
$\{t_1, t_2, t_3\}$	$p(t_1)p(t_2)p(t_3) = .12$
$\{t_2, t_3, t_4\}$	$p(t_2)p(t_3)p(t_4) = .072$

Figure 1. An example uncertain database and all its possible worlds with their probabilities.

tinct in D.

An uncertain database $\mathcal{D}$ is instantiated into a *possible world* assuming mutual independence of the x-tuples [2]. More precisely, let $\tau_1, \ldots, \tau_M$ be the x-tuples of $\mathcal{D}$, and let W be any subset of the tuples appearing in $\mathcal{D}$, the probability of W occurring is $\Pr[W] = \prod_{j=1}^{M} p_W(\tau_j)$, where for any $\tau \in \mathcal{D}$, $p_W(\tau)$ is defined as

$$p_W(\tau) = \begin{cases} p(t), & \text{if } \tau \cap W = \{t\}; \\ 1 - \sum_{t_i \in \tau} p(t_i), & \text{if } \tau \cap W = \emptyset; \\ 0, & \text{otherwise.} \end{cases}$$

If $\Pr[W] > 0$, we say W is a *possible world*, and let $\mathcal{W}$ be the set of all possible worlds. Thus, $\mathcal{D}$ represents a probability distribution over $\mathcal{W}$ in a succinct format. Please refer to Figure 1 for an example.

We distinguish between two cases. In the single-alternative case, each x-tuple has only one alternative; in the multi-alternative case, there could be more than one alternative for an x-tuple.

Top-k queries in an uncertain database. This paper investigates query processing issues under the setting of uncertain data, and in particular we concentrate on top-k queries. Top-k queries received increasing interest in relational databases recently [19], mainly as a way to integrate the imprecise query answering semantics of information retrieval with the highly structured storage and representation of relational data. Because of their particular semantics, top-k queries are even more meaningful in the context of uncertain and probabilistic databases. Some recent efforts started to investigate top-k queries in uncertain databases [25, 9], although with different emphases. This work focuses on the top-k queries defined in [25] and the difference to [9] is discussed in Section 7. In particular, Soliman et al. [25] extend the semantics of top-k queries from relational to uncertain databases. They propose two different definitions for top-k queries in such databases and provide algorithms to compute the query results for each definition. The first definition is the Uncertain Top-k query (U-Top$k$), where the result is the set of tuples with the highest aggregated probability to be the top-k tuples across all possible worlds. The second definition is the Uncertain k-Ranks query (U-kRanks), where each tuple in the result is the most probable tuple to appear at a given rank over all possible worlds.

	U-Top$k$		U-$k$ranks	
alter.	single	multi	single	multi
running time				
ours	$n \log k$	$n \log k$	nk	$n^2 k$
[25]	nk	exponential	$n^2 k$	exponential
space usage				
ours	k	n	k	n
[25]	k^2	exponential	nk	exponential

Table 1. Comparison of the asymptotic results under the x-relation model, where n is the scan depth (Definition 3).

However, although the work in [25] is pioneering and important for top-k query processing in uncertain databases, their solutions are highly inefficient. The basic idea in their algorithms is to map each configuration (a combination of appearing and not appearing tuples) to a state, and formulate the problem as a search problem in the state-graph. Then some generic, A*-type search algorithm is invoked on this gigantic graph to search for the best goal state. Due to the generic nature, their algorithms are able to work under any uncertain data model [24], and for the same reason, the algorithms are exponentially expensive in both time and space. For the basic model where all tuples are mutually independent, i.e., the single-alternative case of the x-relation model, more efficient algorithms are given [25], but they are still far from optimal.

In this paper, we show that under the popular x-relation model, it is possible to exploit the internal structure of the problem to design much more efficient algorithms for processing top-k queries in uncertain databases. Our algorithms are based on entirely different principles than those in [25]. We provide solutions for both U-Top$k$ queries and U-kRanks queries, both of which are significantly faster and use much less space than their counterparts in [25]. A comparison of the asymptotic results of the algorithms under the x-relation model are given in Table 1, from which we can see a clear and dramatic improvement.

The rest of the paper is organized as follows. Section 2

gives the two definitions (from [25]) for top-k queries. We set up the processing framework in Section 3. The improved algorithms for U-Top$k$ and U-kRanks queries appear in Section 4 and 5, respectively. An experimental study is performed in Section 6, followed by a review of related work and the conclusion.

2. Top-k Definitions

In [25], the following two definitions for top-k queries are defined.

Definition 1 [*Uncertain Top-k Query (U-Topk)*] Let $\mathcal{D}$ be an uncertain database with possible worlds space $\mathcal{W}$. For any $W \in \mathcal{W}$, let $\Psi(W)$ be the top-k tuples in W by the score attribute; if $|W| < k$, define $\Psi(W) = \emptyset$. Let T be any set of k tuples. The answer T^* to a U-Top$k$ query on $\mathcal{D}$ is $T^* = \arg\max_T \sum_{W \in \mathcal{W}, \Psi(W)=T} \Pr[W]$. Ties can be broken arbitrarily.

In other words, T^* is the set of k tuples that has the maximum probability of being at the top-k according to the score attribute in a randomly generated world. This definition fits in scenarios where we require the top-k tuples belong to the same world(s). For the example in Figure 1, the U-Top2 answer is $\{t_1, t_2\}$, with a probability of $0.08 + 0.12 = 0.2$.

Definition 2 [*Uncertain k-Ranks Query (U-kRanks)*] Let $\mathcal{D}$ be an uncertain database with possible worlds space $\mathcal{W}$. For any $W \in \mathcal{W}$, let $\psi_i(W)$ be the tuple with the i-th largest score, for $1 \leq i \leq |W|$. The answer to a U-kRanks query on $\mathcal{D}$ is a vector $(t_1^*, \ldots, t_k^*)$, where $t_i^* = \arg\max_t \sum_{W \in \mathcal{W}, \psi_i(W)=t} \Pr[W]$, for $i = 1, \ldots, k$. Ties can be broken arbitrarily.

The answer to a U-kRanks query is a vector of tuples that might not appear together in any possible world, but each of them has the maximum probability of appearing at its rank over all possible worlds. This definition fits in scenarios where the top-k tuples are not restricted to belong to the same world(s). For the example in Figure 1, the U-2Ranks answer is (t_1, t_3): t_1 has a probability of $0.12+0.08+0.18+0.12 = 0.5$ of being at rank 1, and t_3 has a probability of $0.18 + 0.048 + 0.072 = 0.3$ of being at rank 2.

3. Processing Overview

We use a similar system structure as in [25]. We store D, the set of all N tuples in a relational database table, called the *tuple table*, sorted by the decreasing score order. We store information about the x-tuples in an *x-table*. For each x-tuple that has more than one alternative, we store in a list all the alternatives, but with only their id, score, and confidence attributes. All other attributes are not stored in the x-table. By using a hash map, given the id of a tuple t, the score and confidence values for all its alternatives can be retrieved efficiently from the x-table in $O(1)$ time.

To process a top-k query, we retrieve tuples in the decreasing score order from the tuple table, while looking up information from the x-table when needed. We perform computation with the retrieved tuples, and stop as soon as we are certain that none of the unseen tuples may possibly affect the query result.

Why score order? It is curious to ask why we retrieve tuples in the score order instead of some other order, say the confidence order. In order to compare different ordering criteria, we define the *scan depth*, denoted by n, to be the minimum number of tuples that have to be retrieved so as to guarantee the correctness of the result. More formally,

Definition 3 [*Scan depth*] Suppose the tuples in an uncertain database $\mathcal{D}$ are $t_1, \ldots, t_N$ in some predefined order. For a U-Top$k$ or U-kRanks query, the *scan depth* n is the minimum n such that the following holds: for any $\mathcal{D}'$ where the first n tuples in $\mathcal{D}'$ under the same ordering criteria are the same as those of $\mathcal{D}$, i.e., $t_1, \ldots, t_n$, the query answer on $\mathcal{D}'$ is the same as that on $\mathcal{D}$.

The same concept is also introduced in [25]. Here we extend it to any ordering criteria. Note that for many $\mathcal{D}$'s, n is much smaller than N. However in the worst case, n can be as large as N, as illustrated in the examples below. In [24], it is shown that if N is unknown to the algorithm, then access in the score order has the optimal scan depth among all orderings. This applies to the scenario where the tuple table is not materialized; instead, tuples are supplied by an iterator interface that produces tuples in the designated order upon request, and it is difficult to estimate N beforehand. Here we in addition consider the case where N is known, and show that in this case there is no optimal ordering. Let us consider the following example first.

Example 1. Consider the basic case where each x-tuple has only one alternative, and we are to perform a U-Top$k$ query with $k = 1$ (or equivalently a U-$k$ranks query with $k = 1$). Assume that the N (> 2) tuples of $\mathcal{D}$ have $s(t_i) = N - i, p(t_i) = 1/N$ for $1 \leq i \leq N - 1$, and $s(t_N) = 0, p(t_N) = 1$. The query answer will be t_N, since the probability of t_N being the top-1 in a random possible world is $(1-1/N)^{N-1} \geq 1/e$, while the probability of any other tuple is at most $1/N$. Thus, sorting by score will have a scan depth of $n = N$. On the other hand, if we sort by confidence, we can stop as soon as we have retrieved 2 tuples. This is because after having observed that the second tuple has confidence $1/N$, we know that all the remaining tuples' confidences are at most $1/N$, thus we can conclude that t_N must be the answer since its probability is at least $(1-1/N)^{N-1} \geq 1/e$ (assuming pessimistically that all unseen tuples have higher scores and have confidence $1/N$), thus any unseen tuple cannot possibly beat t_N.

If $\mathcal{D}$ is a multi-alternative x-relation, things are slightly more complicated. But still is can be verified that in the worst case, t_N has a probability of $1/N$ of being the answer (when all remaining tuples have larger scores, confidence $1/N$, and are in one x-tuple), thus the algorithm can still stop after retrieving only 2 tuples.

On the other hand, it is also fairly easy to construct an example where sorting by score is much better than sorting by confidence.

Example 2. Still in the same setup as in Example 1, but now the tuples of $\mathcal{D}$ have $s(t_i) = N - i, p(t_i) = 0.5$ for $2 \leq i \leq N$, and $s(t_1) = N, p(t_1) = 0.4$. In this case, the query answer is t_1. It is not difficult to verify that sorting by score gives a scan depth of 2, while sorting by confidence yields $n = N$.

Now that neither choice gives us a satisfying order, one may be tempted to design other functions $f(s, p)$ that might give a good ordering (for example ordering by $s \cdot p$). Unfortunately, we obtained the following negative result, whose proof is given in Appendix A.

Theorem 1 *For any function $f : \mathbb{R} \times [0, 1] \to \mathbb{R}$ and any N, there exists a single-alternative uncertain database $\mathcal{D}$ with N tuples, such that if we retrieve tuples from $\mathcal{D}$ in the order of f, the scan depth is at least $\Omega(N)$ for answering a U-Topk or U-kRanks query with $k = 1$.*

This negative result precludes the existence of an ordering function that is good for all cases. Thus we settle for an ordering that is good for "typical" cases, and we argue that ordering by score is a good choice. First, ordering by order often makes the algorithms easier, by exploiting the fact that all unseen tuples have smaller scores. Second, in many practical situations, the scan depth under score ordering is actually very small, and nowhere near the worst case like the one in Example 1. This is evident from the empirical studies in both [25] and our own experiments in Section 6.

Therefore, the score order is arguably a good order whether N is known or unknown, and thus from now on we will stick to the score order. Without loss of generality we assume that tuples are $t_1, \ldots, t_N$ such that $s(t_1) > \cdots > s(t_N)$. We focus on the following two problems. (1) By definition the scan depth n is the lower bound on the number of tuples that have to be retrieved. Can this lower bound be attained, i.e., can we design an algorithm that immediately stops after reading n tuples? (2) If the answer to (1) is yes, how efficient can the algorithm be? Soliman et al. [25] answered the first question affirmatively, and designed algorithms that read exactly n tuples before termination. Therefore, their algorithms are optimal in the number of tuples retrieved. However, in terms of computation and memory consumption, their algorithms are highly inefficient. In Section 4 and 5, we present much more efficient algorithms for both types of queries.

4. Uncertain Top-k Queries

Define $\mathcal{D}_i$ to be the uncertain database when $\mathcal{D}$ is *restricted* on $D_i = \{t_1, \ldots, t_i\}$, for $i = 1, \ldots, N$, i.e., $\mathcal{D}_i = \{\tau' \mid \tau' = \tau \cap D_i, \tau \in \mathcal{D}\}$. For the database from Figure 1, this means that $\mathcal{D}_1 = \{\tau'_1 = \{t_1\}\}$, $\mathcal{D}_2 = \{\tau'_1 = \{t_1\}, \tau'_2 = \{t_2\}\}$, $\mathcal{D}_3 = \{\tau'_1 = \{t_1\}, \tau'_2 = \{t_2\}, \tau'_3 = \{t_3\}\}$ and $\mathcal{D}_4 = \{\tau'_1 = \{t_1, t_4\}, \tau'_2 = \{t_2\}, \tau'_3 = \{t_3\}\}$. We use $W|\mathcal{D}_i$ to denote a possible world W generated from $\mathcal{D}_i$, with probability $\Pr[W|\mathcal{D}_i]$. For $i \geq k$, let S_i be the most probable world generated from $\mathcal{D}_i$ that consists of k tuples, i.e., $S_i = \arg\max_{|W|=k} \Pr[W|\mathcal{D}_i]$, and let $\rho_i = \Pr[S_i|\mathcal{D}_i]$. Our algorithms for both the single-alternative and the multi-alternative case follows the same general framework: we read tuples one by one, and progressively compute S_i as i goes from k to N. Finally we take the S_i with the maximum ρ_i as the final answer T^*. The correctness of this general framework is guaranteed by the following lemma.

Lemma 1 $\Pr[\Psi(W|\mathcal{D}) = T^*] = \max\{\rho_i \mid k \leq i \leq N\}$.

Proof: Let $i^* = \max\{i \mid t_i \in T^*\}$. It is clear that $\Pr[\Psi(W|\mathcal{D}) = T^*] = \Pr[\Psi(W|\mathcal{D}_{i^*}) = T^*] = \rho_{i^*}$, so $\Pr[\Psi(W|\mathcal{D}) = T^*] \leq \max\{\rho_i \mid k \leq i \leq N\}$.

On the other hand, consider any T' and let $i' = \max\{i \mid t_i \in T'\}$. By definition $\Pr[\Psi(W|\mathcal{D}) = T^*] \geq \Pr[\Psi(W|\mathcal{D}) = T'] = \rho_{i'}$ for any i'. Thus we have $\Pr[\Psi(W|\mathcal{D}) = T^*] = \max\{\rho_i \mid k \leq i \leq N\}$. □

Using Lemma 1, instead of computing T^* by Definition 1, i.e., enumerating all the worlds and calculating the maximum aggregated probability, we could simply compute the ρ_i's, and the S_i corresponding to the maximum ρ_i will be T^*. Therefore, the problem boils down to computing S_i and ρ_i for $i = k, k + 1, \ldots, N$. In fact, we can stop the process as soon as we are certain that none of the remaining ρ_i's is going to be larger than the current maximum ρ_i we have found so far, i.e., as soon as we have read n tuples, where n is the scan depth. However, we still need an efficient algorithm to compute these S_i's and ρ_i's, as well as a method that can tell us if the scan depth is reached or not. Below we first tackle the easier single-alternative case; then we move on to the more challenging multi-alternative case following the same general idea.

4.1. The single-alternative case

Lemma 2 *For a single-alternative database $\mathcal{D}$ and any $k \leq i \leq N$, S_i consists of the k tuples with the largest confidences in D_i, and*

$$\rho_i = \prod_{t_j \in S_i} p(t_j) \cdot \prod_{t_j \in D_i \setminus S_i} (1 - p(t_j)).$$

Proof: Since $\Pr[W|\mathcal{D}_i]$ is the product of two factors, the probability that all tuples in W appear and the probability that none of the rest appears, both of which are maximized when W consists of the k largest-confidence tuples. Once we have S_i, ρ_i is immediate. □

We next characterize the scan depth for this case.

Lemma 3 *For a single-alternative uncertain database $\mathcal{D}$ and a U-Top$k$ query, the scan depth is the minimum n such that*

$$\max_{1\le i\le n} \rho_i \ge \prod_{1\le i\le n} \max\{p(t_i), 1-p(t_i)\}. \qquad (1)$$

Proof: We first show that when (1) happens, no more tuples need to be fetched. This is because the LHS of (1) is the current best answer we have found after reading n tuples; while the RHS of (1) is an upper bound on $\Pr[W|\mathcal{D}_i]$ for any W, regardless of its cardinality, and any $i > n$.

Next we prove that if (1) does not hold, then we must have not reached the scan depth yet, i.e., the condition is tight. This guarantees that our algorithm will not read more than the necessary n tuples. We first prove the following claim: If we have seen k tuples with confidence $\ge 1/2$, then (1) must hold. Indeed, consider the first time we have seen k such tuples, say after reading t_s. Since the k tuples with the largest confidences in D_s must be those k tuples with confidences $\ge 1/2$, combining with Lemma 2 we have $\max_{1\le i\le s} \rho_i \ge \rho_s = \prod_{1\le i\le s} \max\{p(t_i), 1-p(t_i)\}$. Furthermore, since the LHS of (1) never decrease and the RHS of (1) never increase, it must still hold when we have read n tuples.

Now, we construct another $\mathcal{D}'$, whose first n tuples are the same as $\mathcal{D}$, while all of its remaining tuples have confidence 1, and argue that we can find a better U-Top$k$ answer from $\mathcal{D}'$ than the claimed best answer for $\mathcal{D}$ if (1) has not met yet. Since (1) does not hold, there are $\ell < k$ tuples with confidences $\ge 1/2$ in the first n tuples of $\mathcal{D}$ and $\mathcal{D}'$ as we have just argued. Since all the remaining tuples in $\mathcal{D}'$ have confidence 1, putting together these ℓ seen tuples and the first $k-\ell$ unseen tuples gives us a candidate top-k answer for $\mathcal{D}'$ with probability $\prod_{1\le i\le n} \max\{p(t_i), 1-p(t_i)\}$, larger than the current best answer claimed for $\mathcal{D}$. Therefore, by definition we have not reached the scan depth. □

Using Lemma 2 and 3, it is easy to obtain an efficient algorithm for processing a U-Top$k$ query. The algorithm reads the tuples one by one, maintains the k largest-confidence tuples seen so far, and computes each ρ_i using Lemma 2. We can use a heap of size k for this purpose, costing $O(\log k)$ time per tuple. Meanwhile, it maintains the RHS of (1) so as to be able to stop immediately after reading n tuples. This can be easily done in constant time per tuple. Therefore we conclude with the following.

Theorem 2 *For a single-alternative uncertain database, our algorithm can process a U-Top$k$ query by reading n tuples and spending $O(n \log k)$ time. The space requirement is $O(k)$.*

This is to be compared with the previous algorithm [25] that uses $O(nk)$ time and $O(k^2)$ space.

4.2. The multi-alternative case

Next we move on to the multi-alternative case, where each x-tuple may have several (up to some constant) choices. Our algorithm follows the the same framework as the single-alternative case, but we need new, generalized forms of Lemma 2 and 3.

Let $\mathcal{D}$ be a multi-alternative uncertain database. For any i, and any tuple $t_j \in \tau \in \mathcal{D}$, let $q_i(t_j)$ be the probability that none of the tuples in $\tau \cap D_i$ appears in a randomly generated world W, i.e., $q_i(t_j) = 1 - \sum_{t_\ell \in \tau, \ell \le i} p(t_\ell)$. In other words, $q_i(t_j)$ is the probability that none of t_j's alternatives, including t_j, among the first i tuples of D appears.

For any two tuples $t_i, t_j, i \ne j$, that belong to the same x-tuple, if $p(t_i) > p(t_j)$, or $p(t_i) = p(t_j)$ and $i < j$, then we say that t_i *dominates* t_j. For any i, define $\widehat{D}_i$ be the *pruned* version of D_i, i.e., $\widehat{D}_i$ consists of all tuples of D_i that are not dominated by any other tuple in D_i. Note that to compute ρ_i, it is sufficient to consider only $\widehat{D}_i$, since for any $W \subseteq D_i$, we can replace each dominated tuple in W with its dominator, which may only increase $\Pr[W|\mathcal{D}_i]$.

We now extend Lemma 2 and 3 to the multi-alternative case.

Lemma 4 *For a multi-alternative database $\mathcal{D}$ and any i such that $|\widehat{D}_i| \ge k$, S_i consists of the k tuples with the largest $p(t_j)/q_i(t_j)$ ratios[3] in $\widehat{D}_i$, and*

$$\rho_i = \prod_{t_j \in S_i} p(t_j) \cdot \prod_{t_j \in \widehat{D}_i \setminus S_i} q_i(t_j).$$

Proof: Let $Z = \{t_j \mid t_j \in \widehat{D}_i, q_i(t_j) = 0\}$. This implies that any randomly generated world $W \subseteq \widehat{D}_i$ will contain Z. If $|Z| > k$, then for any $W \subseteq \widehat{D}_i$ and $|W| = k$, $\Pr[W|\mathcal{D}_i] = 0$, then any S_i achieves the maximum probability, which is zero. So we only consider the case $|Z| \le k$. For any $W \subseteq \widehat{D}_i$, W must include Z in order to have a non-zero probability, thus we have

$$\begin{aligned}\Pr[W|\mathcal{D}_i] &= \prod_{t_j \in Z} p(t_j) \prod_{t_j \in W \setminus Z} p(t_j) \prod_{t_j \in \widehat{D}_i \setminus W} q_i(t_j) \\ &= \prod_{t_j \in Z} p(t_j) \prod_{t_j \in W \setminus Z} \frac{p(t_j)}{q_i(t_j)} \prod_{t_j \in \widehat{D}_i \setminus Z} q_i(t_j).\end{aligned}$$

[3] We define $x/0 = \infty$ for any $x > 0$.

Since the first and third products are fixed while the second one is maximized when $W \setminus Z$ consists of the $k - |Z|$ tuples with the largest $p(t_j)/q_i(t_j)$ ratios in $\widehat{D}_i \setminus Z$, and by definition the tuples in Z have an infinite ratio, the lemma is proved. □

Lemma 5 *For a multi-alternative uncertain database $\mathcal{D}$ and a U-Topk query, the scan depth is the minimum n such that*

$$\max_{1 \le i \le n} \rho_i \ge \prod_{t_i \in \widehat{D}_n} \max\{p(t_i), q_n(t_i)\}. \quad (2)$$

Proof: The proof follows the the same lines of reasoning as the proof of Lemma 3.

First, the LHS of (2) is the current best answer we have found after reading n tuples, while the RHS of (2) is an upper bound on $\Pr[W|\mathcal{D}_i]$ for any W, regardless of its cardinality, and any $i > n$. Therefore, (2) is a sufficient condition upon which we can terminate the algorithm.

Next we show that (2) is also a necessary condition. We first prove the following claim: If we have seen k tuples t_i in $\widehat{D}_n$ such that $p(t_i) \ge q_n(t_i)$, then (2) must hold. Indeed, consider the minimum s such that there are exactly k tuples in $\widehat{D}_s$ with $p(t_i) \ge q_s(t_i)$. Since these k tuples must have the largest $p(t_i)/q_s(t_i)$ ratios in $\widehat{D}_s$ (they have ratios ≥ 1 while the others < 1), by Lemma 4 we have $\max_{1 \le i \le s} \rho_i \ge \rho_s = \prod_{1 \le i \le s} \max\{p(t_i), q_s(t_i)\}$. So (2) must hold when $n = s$. Furthermore, as n increases, the LHS of (2) never decreases and the RHS of (2) never increases (since $q_n(t_i)$ never increases), it must still hold when we have read $n \ge s$ tuples.

Now, we construct another $\mathcal{D}' = \mathcal{D}_n \cup \{\{t'_{n+1}\}, \ldots, \{t'_N\}\}$, with $s(t_n) > s(t'_{n+1}) > \cdots s(t'_N)$, and $p(t'_{n+1}) = \cdots = p(t'_N) = 1$, i.e., the first n tuples in $\mathcal{D}'$ are the same as those in $\mathcal{D}$, with all of its remaining tuples having confidence 1 and independent of the first n tuples. We argue that if (2) does not hold, we can find a better U-Top$k$ answer from $\mathcal{D}'$ than the claimed best answer for $\mathcal{D}$. By the above claim, there are $\ell < k$ tuples t_i with $p(t_i) \ge q_n(t_i)$ in $\widehat{D}_n$. Since all the remaining tuples in $\mathcal{D}'$ have confidence 1, putting together these ℓ seen tuples and the first $k - \ell$ unseen tuples gives us a candidate top-k answer for $\mathcal{D}'$ with probability $\prod_{1 \le i \le n} \max\{p(t_i), q_n(t_i)\}$, larger than the current best answer claimed for $\mathcal{D}$. □

Using Lemma 4 and 5, our algorithm proceeds as follows. As i goes from k to N, we keep in a table of size $O(n)$ the $p(t_j)$ and $q_i(t_j)$ values for all tuples that have been seen. These probabilities can be maintained in $O(1)$ time per tuple, since the $p(t_j)$'s stay the same, and at most one of the $q_i(t_j)$'s changes as a new tuple is retrieved. We also maintain $\widehat{D}_i$, i.e., all the dominators among these tuples. This can be done in $O(1)$ time per tuple, too, since there is at most one insertion or one replacement in $\widehat{D}_i$ in each step. We construct a binary tree on the k dominators with the largest $p(t_j)/q_i(t_j)$ ratios in sorted order. We update the binary tree for each incoming tuple. In each step, we need to either insert a new tuple and delete one, or increase the ratio of an existing tuple. In both cases the cost is $O(\log k)$. Finally, it is also easy to maintain the RHS of (2) in constant time per tuple. Therefore we have the following, in contrast with the previous algorithm [25] that has both time *and* space complexities exponential in k.

Theorem 3 *For a multi-alternative uncertain database, our algorithm can process a U-Topk query by reading n tuples and spending $O(n \log k)$ time. The space requirement is $O(n)$.*

5. Uncertain k-Ranks Queries

In this section we consider U-kRanks queries. We first give a dynamic programming algorithm for answering U-kRanks queries in the single-alternative case, and then extend it to the multi-alternative case. Note that Soliman et al. [25] also used a dynamic programming approach based on totally different formulation that works only for the single-alternative case, where they capture exactly which tuple appears at rank $j-1$ in order to get the probability for a given tuple to be at rank j. However, our new formulation not only runs faster, but also naturally extends to the multi-alternative case, for which only an exponential algorithm is given in [25]. Our algorithms are based on the following simple intuition: The probability that a tuple t_i appears at rank j depends only on the event that exactly $j-1$ tuples from the first $i-1$ tuples appear, no matter *which* tuples appear.

5.1. The single-alternative case

Let $\mathcal{D}$ be a single-alternative uncertain database. For $1 \le j \le i \le N$, let $r_{i,j}$ be the probability that a randomly generated world from $\mathcal{D}_i$ has exactly j tuples, i.e., $r_{i,j} = \sum_{|W|=j} \Pr[W|\mathcal{D}_i]$. We also define $r_{0,0} = 1$. It is clear that the probability that t_i ranks the j-th in a randomly generated world from $\mathcal{D}$ is $p(t_i) \cdot r_{i-1,j-1}$. Therefore, the answers to a U-kRanks query on $\mathcal{D}$ are $t_{\chi(j)}$ where

$$\chi(j) = \arg \max_{j \le i \le N} \{p(t_i) \cdot r_{i-1,j-1}\}, \quad (3)$$

for $j = 1, \ldots, k$.

We are now left with the task of computing the $r_{i,j}$'s, which are related by the following equation.

$$r_{i,j} = \begin{cases} p(t_i) r_{i-1,j-1} + (1 - p(t_i)) r_{i-1,j}, & \text{if } i \ge j > 0; \\ 1, & \text{if } i = j = 0; \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

The correctness of (4) is obvious: To get j tuples from $\mathcal{D}_i$, we either choose t_i and $j-1$ tuples from $\mathcal{D}_{j-1}$, or not choose t_i and take all j tuples from $\mathcal{D}_{i-1}$.

Upon reading each tuple t_i, our algorithm computes $r_{i,j}$ using (4) for $j = 0, 1, \ldots, \min\{i, k\}$. It also keeps the current best answers $\chi(j)$ found so far according to (3). Since to compute $r_{i,j}$, only the $r_{i-1,j}$'s are needed, our algorithm only requires $O(k)$ space throughout the computation.

Finally, we have the following characterization of the scan depth n, so that our algorithm can terminate as soon as the answers are known, retrieving only n tuples from the tuple table, which is the minimum possible.

Lemma 6 *For a single-alternative uncertain database $\mathcal{D}$ and a U-kRanks query, the scan depth is the minimum n such that the following holds for each $j = 1, \ldots, k$:*

$$\max_{j \le i \le n} \{p(t_i) r_{i-1,j-1}\} \ge \max_{0 \le \ell \le j-1} r_{n,\ell}. \tag{5}$$

Proof: Since the LHS of (5) is the current best answer for the tuple at rank j, it is sufficient to prove that, for any $\mathcal{D}'$ whose tuples are $t_1, \ldots, t_n, t'_{n+1}, \ldots, t'_N$, the RHS of (5) is an upper bound on the probability of any t'_i being at rank j for $j = 1, \ldots, k$, and this upper bound is attainable.

First, for any $i > n$, consider the probability of t'_i being at rank j in a randomly generated world from $\mathcal{D}'$. Letting ξ_s be the probability that exactly s tuples from $\{t'_{n+1}, \ldots, t'_{i-1}\}$ appear (define $\xi_0 = 1$ if $i = n+1$), we have

$$\begin{aligned} \Pr[\psi_j(W|\mathcal{D}') = t'_i] &= p(t'_i) \left(\sum_{\ell=0}^{j-1} r_{n,\ell} \cdot \xi_{j-1-\ell} \right) \\ &\le \sum_{\ell=0}^{j-1} r_{n,\ell} \cdot \xi_{j-1-\ell} \le \max_{0 \le \ell \le j-1} r_{n,\ell}, \end{aligned}$$

where the last inequality holds because $\sum_{s=0}^{j-1} \xi_s \le 1$. Thus, we need to access at most n tuples before we can report the correct answers.

Secondly, we show that for any j, there is a $\mathcal{D}'$ with some unseen tuple that achieves this upper bound. Set $p(t'_{n+1}) = \cdots = p(t'_N) = 1$, and let $\ell^* = \arg\max_{0 \le \ell \le j-1} r_{n,\ell}$. Consider the tuple $t'_{n+j-\ell^*}$. The probability that it appears at rank j in a random world from $\mathcal{D}'$ is exactly r_{n,ℓ^*}. Therefore, we also need to access at least n tuples to avoid any mistakes. □

Since we can check the inequality (5) for all $1 \le j \le k$ easily in $O(k)$ time per tuple, the theorem below immediately follows.

Theorem 4 *For a single-alternative uncertain database, our algorithm can process a U-kRanks query by reading n tuples and spending $O(nk)$ time. The space requirement is $O(k)$.*

Note that the previous algorithm [25] for this problem runs runs in $O(n^2k)$ time[4] and uses $O(nk)$ space.

5.2. The multi-alternative case

Our U-kRanks algorithm for the multi-alternative case will follow the same framework as the single-alternative case. However, several difficulties need to be resolved with regard to the alternatives.

The first difficulty is that the $r_{i,j}$'s cannot be related simply as in (4) any more, because if t_i has some preceding alternatives, the event that t_i appears is no longer independent of the event that exactly $j-1$ tuples in $\mathcal{D}_{i-1}$ appear. The trick to overcome this difficulty is to convert $\mathcal{D}_i$ into a single-alternative $\bar{\mathcal{D}}_i$, and then apply the previous algorithm to compute $r_{i,j}$, for $j = 0, \ldots, k$.

We construct $\bar{\mathcal{D}}_i$ as follows. For each x-tuple $\tau \in \mathcal{D}_i$, we create an x-tuple $\bar{\tau} = \{\bar{t}\}$ in $\bar{\mathcal{D}}_i$ where $p(\bar{t}) = \sum_{t \in \tau} p(t)$, with all of $\bar{t}$'s other attributes set to *null*. In other words, we merge all tuples in τ into one *representative* $\bar{t}$, whose probability is the sum of all their probabilities. We claim that $r_{i,j}$ computed from $\bar{\mathcal{D}}_i$ is the same as the probability that exactly j tuples in $\mathcal{D}_i$ appear. Intuitively, because here we only care about the number of tuples appearing, merging does not affect anything since the probability that $\bar{t}$ appears is the same as the probability that exactly one tuples in τ appears. The following lemma gives a more rigorous argument.

Lemma 7 *For any $0 \le j \le k$,*

$$\sum_{|W|=j} \Pr[W|\mathcal{D}_i] = \sum_{|W|=j} \Pr[W|\bar{\mathcal{D}}_i].$$

Proof: For each x-tuple $\tau \in \mathcal{D}_i$, let I_τ be the indicator random variable such that $I_\tau = 1$ if exactly one tuples from τ appears, and 0 otherwise. It is easy to see that these I_τ's are mutually independent and $\Pr[I_\tau = 1] = \sum_{t \in \tau} p(t) = p(\bar{t})$. So we have

$$\begin{aligned} \sum_{|W|=j} \Pr[W|\mathcal{D}_i] &= \Pr\left[\sum_{\tau \in \mathcal{D}_i} I_\tau = j \right] = \Pr\left[\sum_{\bar{\tau} \in \bar{\mathcal{D}}_i} I_{\bar{\tau}} = j \right] \\ &= \sum_{|W|=j} \Pr[W|\bar{\mathcal{D}}_i]. \end{aligned}$$

□

Now we have a way to compute all the $r_{i,j}$'s, but the second difficulty is that the probability of t_i ranking at j is no longer simply $p(t_i) \cdot r_{i-1,j-1}$, if t_i has some preceding alternatives in $\mathcal{D}_{i-1}$, because the existence of t_i would exclude

[4] In fact, the algorithm described in [25] has a worst-case running time of $O(N^2k)$, due to the termination condition of that algorithm not being tight, which may cause the algorithm to read far more tuples, up-to N in the worst case, than the necessary scan depth n. See Appendix B for such an example. However, these contrived cases rarely happen in practice, so we still consider the bound as $O(n^2k)$.

all its other alternatives, while $r_{i-1,j-1}$ includes the probability of the possible worlds that contain one of them. To cope with this exclusiveness, we define $\mathcal{D}^-_{i-1} = \mathcal{D}_{i-1} \backslash \{\tau \in \mathcal{D}_{i-1} \mid \tau$ includes an alternative of $t_i\}$, that is, $\mathcal{D}^-_{i-1}$ is the version of $\mathcal{D}_{i-1}$ that excludes all the alternatives of t_i. Similarly, letting $r^-_{i-1,j-1}$ be the probability of exactly $j-1$ tuples from $\mathcal{D}^-_{i-1}$ appearing, (3) becomes

$$\chi(j) = \arg \max_{j \le i \le N} \{p(t_i) \cdot r^-_{i-1,j-1}\}. \quad (6)$$

Similarly define $\bar{\mathcal{D}}^-_i$ to be the single-alternative version of $\mathcal{D}^-_i$, i.e., after merging all tuples of each x-tuple of $\mathcal{D}^-_i$ into a representative tuple. Thus, we can compute the $r^-_{i-1,j-1}$'s on $\bar{\mathcal{D}}^-_{i-1}$ using the dynamic program.

Finally, the condition (5) for the scan depth in Lemma 6 becomes

$$\max_{j \le i \le n} \{p(t_i) r^-_{i-1,j-1}\} \ge \max_{0 \le \ell \le j-1} r_{n,\ell}, \quad (7)$$

since the LHS of (7) is the current best answer for the rank-j tuple, while the RHS is still the attainable upper bound on the probability of any unseen tuple being at rank j.

The algorithm. Having resolved all the difficulties, our algorithm proceeds as follows. Initially, we have $\bar{\mathcal{D}}_1 = \mathcal{D}_1$. Next, for each fetched tuple t_i, we incrementally build $\bar{\mathcal{D}}_i$, compute $r_{i,j}$ (and $r^-_{i-1,j-1}$ when necessary), and update $\chi(j)$. More precisely, if t_i does not have any preceding alternatives, $\bar{\mathcal{D}}_i$ is simply $\bar{\mathcal{D}}_{i-1}$ appended with $\{t_i\}$, the $r_{i,j}$'s and $\chi(j)$'s can be computed as before, according to (3) and (4). This takes only $O(k)$ time. If t_i has one or more preceding alternatives, we first construct $\bar{\mathcal{D}}^-_{i-1}$ from $\bar{\mathcal{D}}_{i-1}$, by simply setting the probability of the representative tuple for the x-tuple τ (that t_i belongs to) in $\bar{\mathcal{D}}_{i-1}$ to zero. Next we compute $r^-_{i-1,j-1}$ from $\bar{\mathcal{D}}^-_{i-1}$ and update the $\chi(j)$'s according to (6). This process takes $O(nk)$ time. Next, we construct $\bar{\mathcal{D}}_i$ from $\bar{\mathcal{D}}_{i-1}$ by increasing the corresponding representative tuple's confidence by $p(t_i)$, and then compute $r_{i,j}$ using the dynamic program. This process also takes $O(nk)$ time. Finally, we check the condition (7) to determine if we should terminate. The detailed algorithm is given in Algorithm 1.

Assume there are m tuples with preceding alternatives in the first n tuples from $\mathcal{D}$, we have the following result.

Theorem 5 *For a multi-alternative uncertain database, our algorithm can process a U-kRanks query by reading n tuples and spending $O(nmk)$ time. The space requirement is $O(n)$.*

Proof: The time bound follows from the fact that we invoke the dynamic program only m times, while for tuples that do not have preceding alternatives, the cost is only $O(k)$ per tuple. Since at any i, we only keep the current $\mathcal{D}_i, \bar{\mathcal{D}}_i$, $r_{i,j}$'s, and possibly $\mathcal{D}^-_{i-1}$ and the $r^-_{i-1,j-1}$'s, which take $O(n)$ space in total. Finally, the dynamic program takes only $O(k)$ space. Therefore, the space complexity of our algorithm is $O(n+k) = O(n)$. □

Algorithm 1: Processing U-kRanks queries

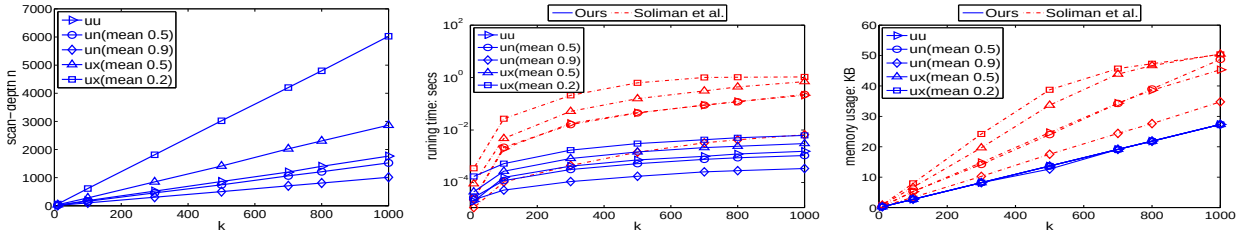
```
p̄ := [0, ..., 0];   // vector p̄ stores the current D̄_i
χ := [0, ..., 0];   // stores the best answers
b := [0, ..., 0];   // stores the prob. of the best answers
r := [0, ..., 0];   // stores r_{i,1}, ..., r_{i,k}
for i = 1, ..., N do
    retrieve t_i;
    if t_i has no preceding alternatives then
        p̄[i] := p(t_i);
        r' := r;   // r' keeps r_{i-1,1}, ..., r_{i-1,k}
        update χ and b using r' and (3);
        compute r using r' and (4);
    else
        Let t_ℓ be the first alternative of t_i;
        p' := p̄[ℓ];   // p' temporarily saves p̄[j]
        p̄[ℓ] := 0;
        compute r^-_{i-1,j-1} on p̄ by dynamic program;
        update χ and b using (6);
        p̄[ℓ] := p' + p(t_i);
        compute r on p̄ by dynamic program;
    y := 0;   // RHS of (7)
    for j = 1, ..., k do
        if r[j-1] > y then y := r[j-1];
        if b[j] < y then continue with next tuple;
    halt and output χ and b;
```

Note that m is at most n, so the worst-case running time of our algorithm is $O(n^2k)$. Our new algorithm has another appealing feature: the running time degrades gracefully as the number of tuples with alternatives increases. In cases where few x-tuples have only more than one alternatives, our multi-alternative algorithm is able to cope with them without a significant loss in efficiency compared with the simple single-alternative case. While the previous algorithm [25] has to use exponential time and space even if there is only one x-tuple having multiple alternatives.

6. Experiments

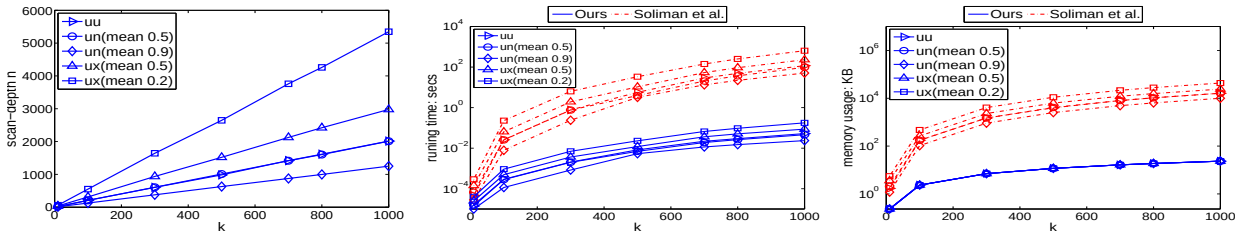
We have implemented both our and Soliman et al.'s algorithms [25] under GNU C++. We also optimized both implementations to our best effort. We generated synthetic data sets in the same way as [25] to test the performance of the algorithms[5]. The score and confidence values in these data sets follow a number of different distributions, also with different correlations. For each data set, we report its scan depth, as well as the running time and memory

[5] All source code, data sets, and data generators are available for download at http://cs-people.bu.edu/lifeifei/utopk/.



(a) k vs. scan depth n. (b) k vs. running time. (c) k vs. memory usage.

Figure 2. U-Top$k$: single-alternative, different distributions of confidence.



(a) k vs. scan depth n. (b) k vs. running time. (c) k vs. memory usage.

Figure 3. U-kRanks: single-alternative, different distributions of confidence.

usage of each algorithm. Note that the scan depth n is completely determined by the data set; all of our algorithms and [25]'s algorithms stop after retrieving n tuples. Since all algorithms consume tuples in the score order, the underlying ranking process in the database engine is the same, so we only measure the costs associated with the top-k processing on the tuple stream that is already sorted by score. The same step has been taken by [25]. Each data set we generated contains $N = 20,000$ tuples. Note that the algorithm's performance does not depend on N since we never exhaust the entire tuple stream. All experiments were executed on a Linux PC with a 2.8GHz Pentium processor and 2GB of main memory. In all cases, algorithms from this work and [25] produce the same results. This observation empirically verifies the correctness of our algorithms.

6.1 The single-alternative case

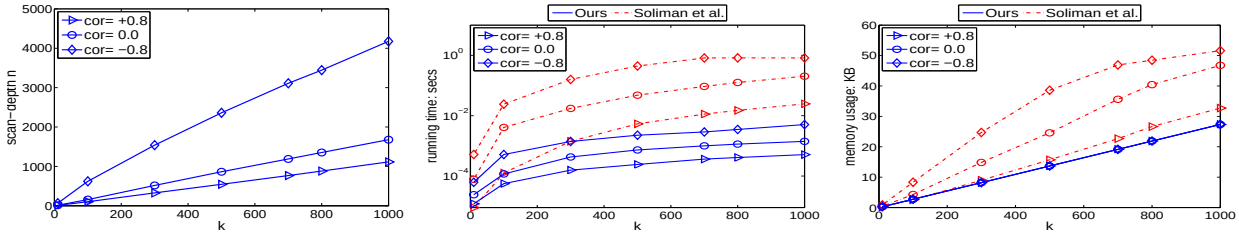
We first report the experimental results for the single-alternative algorithms.

Different distributions of confidence. We first study the case where there is no correlation between score and confidence. Since only the relative order of the scores matters, we fixed the scores to be $1, \ldots, N$, and generated the confidence values according to a few different distributions. Specifically, we have experimented with the following distributions: 1) uniform (denoted as uu); 2) normal (denoted as un) with 0.5 or 0.9 as mean using 0.2 standard deviation; 3) exponential (denoted as ux) with 0.5 or 0.2 as mean.

The experimental results for U-Top$k$ queries are shown in Figure 2. Figure 2(a) shows the scan depth for different data sets, from which we can see that it is always linear in k for all distributions, and worst-case situations like the one in Example 1 never occur. This confirms our earlier claim that the score order is typically a good order. However, different distributions do affect the coefficient in the linear relation between n and k: A lower mean value for confidence increases it, and so does a skewer distribution. Intuitively, when the mean is low, later tuples in the score-ranked tuple stream are more likely to be in the top-k result, hence leads to a larger scan depth. In terms of running time, our algorithm is around 10 to 100 times faster (Figure 2(b)), which is expected from the bounds in Table 1. Our algorithm also consumes less memory as indicated from Figure 2(c), which is linear in k regardless of the distribution. While the algorithm of [25] has a worst-case $O(k^2)$ memory space since it keeps k representative states for states of length 1 to k, one for each length, but in practice it is usually much better than $O(k^2)$ due to the pruning of smaller-length states when there is at least one state with a larger length and a higher probability. Nevertheless it still takes more space than our algorithm in all test cases.

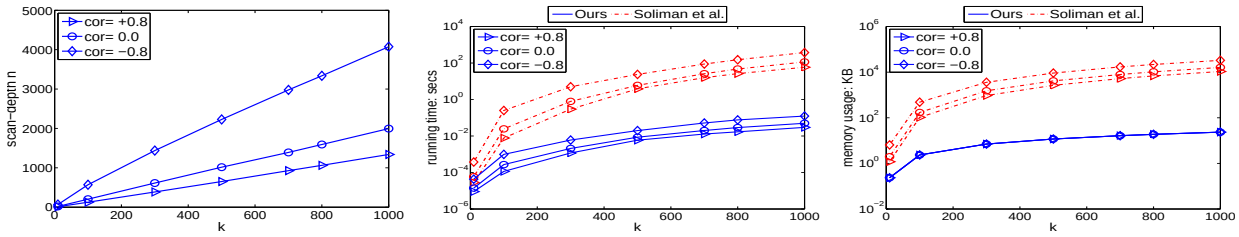
Figure 3 reports the experimental results on the same data sets on U-kRanks queries. The same trend has been observed for the scan depth (Figure 3(a)). Our algorithm is the clear winner by a factor of 10^2 to 10^3 in both running time (Figure 3(b)) and memory usage (Figure 3(c)). The gap gets larger as k (hence n) increases. This naturally follows the bounds in Table 1 where we expect an $O(n)$-factor saving in time and space.

Score-confidence correlations. The correlation between score and confidence will affect the scan depth and the performance of the algorithms, too, as observed in [25],



(a) k vs. scan depth n. (b) k vs. running time. (c) k vs. memory usage.

Figure 4. U-Top$k$: single-alternative, different correlations between score and confidence.



(a) k vs. scan depth n. (b) k vs. running time. (c) k vs. memory usage.

Figure 5. U-kRanks: single-alternative, different correlations between score and confidence.

and we study its effects here. We generated data sets from bivariate normal distributions with different correlations $(+0.8, 0, -0.8)$ and treated score and confidence as the two dimensions, and then ran both U-Top$k$ and U-kRanks queries on them.

The results for U-Top$k$ queries are presented in Figure 4. Not surprisingly, a positive correlation decreases query costs and a negative correlation increases query costs, as a result of processing tuples in the score order. Our algorithm is still the clear winner in both running time and memory usage (see Figure 4(b) and 4(c)), and is always highly efficient. In the worst case, with strongly negatively correlated data and $k = 1000$, our algorithm takes less than 0.01 second of time and 30 KB of memory.

Figure 5 reports the results for U-kRanks queries and the trend is similar. Our algorithm still consistently beats [25] by orders of magnitude in both running time and memory usage. In the worst case, with strongly negatively correlated data and $k = 1000$, it takes less than 0.1 second of time and 50 KB of memory.

6.2 The multi-alternative case

We now shift attention to the multi-alternative case, which is handled by [24] with exclusiveness rules. Note that although the algorithms of [24] in principle support any uncertain data model, their experimental evaluations are limited to the x-relation model.

We introduce a couple of measures to control the characteristic of the data sets used in the experiments. The number of alternatives an x-tuple could have is denoted as the d_x, called the *x-degree*. The ratio of the number of tuples involved in all x-tuples over the total number of tuples is called the *x-percentage*, denoted δ_x. Note that the number of x-tuples is thus $\delta_x N/d_x$. The data sets are generated as follows. We first generate the score and confidence values for all tuples using a bivariate normal distribution with a given correlation, in the same way as the single-alternative case. Then we repeatedly pick d_x tuples at random and group them into an x-tuple; if their confidence values add up to more than 1, we relinquish them and take another set of tuples until we form a valid x-tuple. We repeat the process until we have reached the desired x-percentage. We use the default values $\delta_x = 0.1$ and $d_x = 2$ unless specified otherwise.

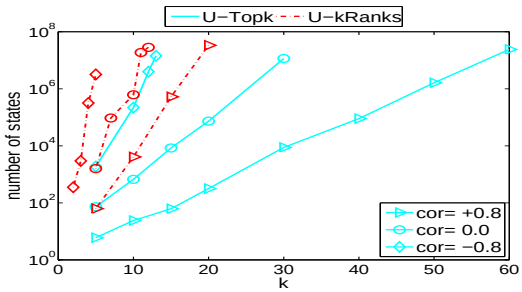


Figure 6. [25]: k vs. number of states kept.

The exponential nature of [25]'s algorithms. We will start with an illustration of the exponential nature, for both running time and memory usage, of [25]'s algorithms. Both their U-Top$k$ and U-$k$ranks algorithms essentially enumerate all possible combinations of the first n tuples in the data set, in the process of searching for the best goal state in the huge state-graph. The U-Top$k$ algorithm does slightly bet-

ter with a pruning strategy that reduces the space of states explored, but it is still exponential. While the U-kRanks algorithm virtually keeps and expands all possible states. The number of states kept by the two algorithms for various k's and correlations are shown in Figure 6. It is clearly increasing in an exponential fashion and one could only afford a very small k with the 2GB of main memory. Even with strongly positively correlated data, it could only tolerate a k up to 60 (resp. 20) for U-Top$k$ (resp. U-kRanks) queries, before the 2GB of main memory is used up. For strongly negatively correlated data, the maximum allowed k drops to less than 15 (resp. 10).

Score-confidence correlations. Varying the correlations, Figure 7 reports the experimental results on U-Top$k$ queries and Figure 8 for U-kRanks queries. Figure 7(a) and 8(a) show that our algorithms have linear scan depth. In both running time and memory usage, the algorithms of [25] are already dramatically more expensive than our new algorithms even for small values of k, as indicated from Figures 7(b), 7(c), 8(b), and 8(c), due to their exponential nature. For our algorithms, both of them occupy linear space w.r.t k. In terms of running time, the U-kRanks algorithms is more expensive with its $O(n^2k)$ cost compared to the $O(n \log k)$ cost of U-Top$k$. It is also worth noting that for U-Top$k$ queries, our algorithm for the multi-alternative case achieves almost the same running time as the single-alternative case. Our algorithms are extremely efficient in all the test cases: even in the most difficult case, 200KB of memory space and 0.01 second is more than enough to process a U-Top$k$ query, and 2 seconds for a U-kRanks query.

Varying x-percentage or x-degree. The last set of experiments studies the effects of x-percentage δ_x and x-degree d_x. All experiments are executed with $k = 300$. Figure 9 summarizes the findings for various δ_x's, from which we can see that δ_x does not significantly affect either our U-Top$k$ or U-kRanks algorithm. Figure 10 are the results for varying d_x. Similarly it does not affect our algorithms with the only exception in the strongly negatively correlated case for U-kRanks algorithm, where the running time demonstrates a linear increase.

7. Related Work

Modeling and building real systems for uncertain databases are the most important issue to be addressed. It is impossible to list all currently developing systems, nevertheless, TRIO [6, 2, 21], MayBMS [3, 4] and Probabilistic databases [10] are three promising representatives. There are also works focusing on the general query processing techniques for uncertain databases under the possible worlds semantics, such as the ConQuer project [13] and the discussion on generating proper and efficient query plans [10]. Special attention to imprecise information arising from mobile data management has been made in [8] where the main focus is querying and indexing on evolving data over continuous intervals. Indexing techniques of uncertain data also appear in [23, 26, 20] and probabilistic graph model is proposed to represent correlated tuples in uncertain databases [22].

There is another recent work concerning about top-k query processing in uncertain databases [9]. The problem is to find the k most probable answers for a given SQL query, where the ranking is purely based on the confidence of the resulting tuples and there is no additional scoring dimension involved to determine the final rank. The solution is based on Monte Carlo simulations. There the top-k definition is quite different from the work of [25] and ours. Our work, as a direct follow-up of [25], concentrates on extending the traditional top-k query definition from the relational database, in the sense that a scoring function is defined to compute the rank, together with the confidence of resulting tuples (in the same spirit as it is in [9]) to jointly determine the final result. Finally, Top-k query processing has been proved by many real applications as one of the most important types of queries in relational databases (see [19] and the references therein), and it is not surprising to see the same trend in uncertain databases.

8. Conclusion

This work introduces new algorithms for top-k query processing in uncertain databases that dramatically improve the state of the art. The proposed algorithms are demonstrated, both theoretically and experimentally, to have highly efficient running time and low memory usage, leading to excellent scalability. It is especially interesting to learn from this study that it is possible to answer top-k queries when tuples are not independent (under the x-relation model) without using exponential time and space, unlike the results presented by previous work. An important future direction is to extend the top-k query processing in uncertain databases with the imprecise query semantics that is becoming quite common in the relational model.

References

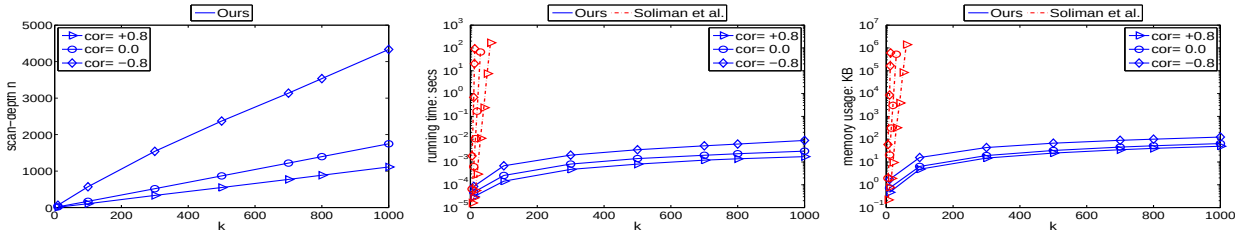
[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD*, 1987.

[2] 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.

[3] L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, 2007.

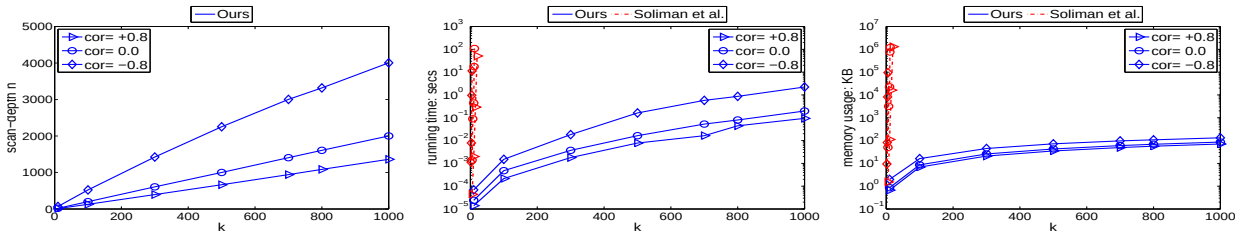
[4] L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *SIGMOD*, 2007.

[5] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE TKDE*, 4(5):487–502, 1992.



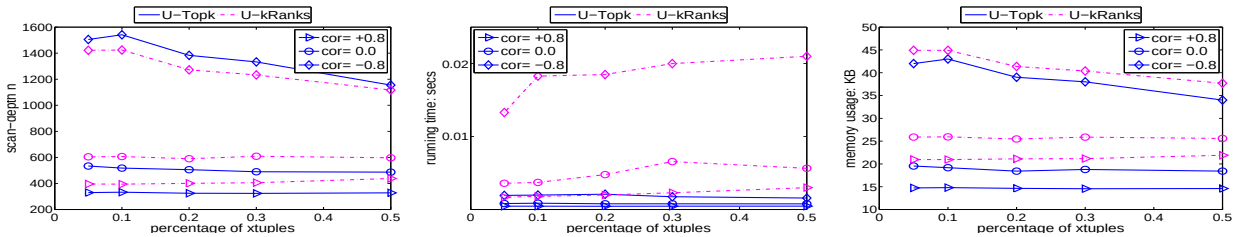
(a) k vs. scan depth n. (b) k vs. running time. (c) k vs. memory usage.

Figure 7. U-Top$k$: multi-alternative, different correlations, $\delta_x = 0.1$, and $d_x = 2$.



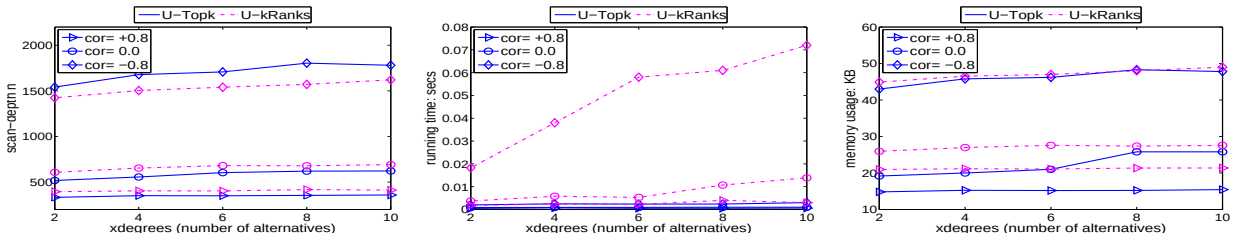
(a) k vs. scan depth n. (b) k vs. running time. (c) k vs. memory usage.

Figure 8. U-kRanks: multi-alternative, different correlations, $\delta_x = 0.1$, and $d_x = 2$.



(a) δ_x vs. scan depth n. (b) δ_x vs. running time. (c) δ_x vs. memory usage.

Figure 9. Varying x-percentage δ_x, with $k = 300$, cor$= 0.0$, $d_x = 2$.



(a) d_x vs. scan depth n. (b) d_x vs. running time. (c) d_x vs. memory usage.

Figure 10. Varying x-degree d_x, with $k = 300$, cor$= 0.0$, $\delta_x = 0.1$%.

[6] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: databases with uncertainty and lineage. In *VLDB*, 2006.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[8] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[9] C.Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probalistic databases. In *ICDE*, 2007.

[10] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[11] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[12] N. Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *VLDB*, 1990.

[13] A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: efficient management of inconsistent databases. In *SIGMOD*, 2005.

[14] H. Galhardas, D. Florescu, and D. Shasha. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, 2001.

[15] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage year. In *VLDB*, 2006.
[16] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
[17] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.
[18] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. ProbView: a flexible probabilistic database system. *ACM TODS*, 22(3):419–469, 1997.
[19] C. Li, K. Chang, I. Ilyas, and S. Song. RankSQL: Query algebra and optimization for relational top-k queries. In *SIGMOD*, 2005.
[20] V. Ljosa and A. K. Singh. APLA: Indexing arbitrary probability distributions. In *ICDE*, 2007.
[21] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.
[22] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.
[23] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *ICDE*, 2007.
[24] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. Technical report, University of Waterloo, 2006.
[25] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.
[26] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, 2005.

A Proof of Theorem 1

Proof: For simplicity we assume that N is even. The same arguments work for odd N. Consider a single-alternative uncertain database $\mathcal{D}$, with tuples $t_1, \ldots, t_N$ with $s(t_i) = N - i$ and $p(t_i) = \frac{1}{N-i+2}$ for all $1 \leq i \leq N$. It is easy to verify that the query answer could be any tuple as the probability of p_i being the top-1 tuple is exactly $1/(N+1)$ for any i. If we perturb $\mathcal{D}$ slightly by increasing the probability of any t_i by a small $\epsilon > 0$, then the balance is broken and t_i will be the (unique) answer. We can also perturb the score of t_i by ϵ. This will not change anything since only the relative order of the scores matters. In the following we will choose ϵ to be a sufficiently small positive.

Let t'_i be the perturbed t_i, i.e., $s(t'_i) = s(t_i) + \epsilon, p(t'_i) = p(t_i) - \epsilon$. Let $\mathcal{D}^i$ be $\mathcal{D}$ with only t_i replaced with t'_i. Consider the following f values: $f_i = f(s(t_i), p(t_i))$ and $f'_i = f(s(t'_i), p(t'_i))$. If there exists an i such that $f'_i \leq f_j$ for at least $N/2$ choices of j, then the scan depth of $\mathcal{D}^i$ is at least $N/2$, since the query answer of $\mathcal{D}^i$, t'_i, is the $(N/2)$-th tuple at best in $\mathcal{D}^i$ when ordered by f (when there is a tie among the tuples in terms of f, we should consider the worst-case ordering). Suppose otherwise for all i, $f'_i > f_j$ for at least $N/2$ choices of j. Then we have at least $N^2/2$ pairs of (i, j) such that $f'_i > f_j$. Therefore there must be some j^* such that $f'_i > f_{j^*}$ holds for at least $N/2$ choices of i. Now we construct a $\mathcal{D}^*$ whose scan depth is at least $N/2 - 1$. There are two cases. (a) If $j^* \geq 2$, then $\mathcal{D}^*$ consists of $t'_1, \ldots, t'_{j^*-2}, t_{j^*}, t'_{j^*}, \ldots, t'_N$. Note that here essentially we are replacing t'_{j^*-1} with t_{j^*} from $\{t'_1, \ldots, t'_N\}$. Since this replacement does not change the score order, and also effectively increase the confidence of t'_{j^*-1}, by choosing ϵ small enough, it is not hard to verify that the query answer of $\mathcal{D}^*$ is t_{j^*}. On the other hand, t_{j^*} is ordered after at least $N/2 - 1$ of these t'_i's. Thus, the scan depth of $\mathcal{D}'$ is at least $N/2$. (b) If $j^* = 1$, then $\mathcal{D}^*$ consists of $t_1, t'_3, \ldots, t'_N, t^0$, where t^0 is a dummy tuple with $p(t^0) = 0$. By choosing ϵ small enough, it is still the case that t^1 is the query answer to $\mathcal{D}'$, while t_1 is ordered after at least $N/2 - 2$ of these t'_i's, and the theorem is proved. □

B Termination Condition of the U-kRanks Algorithm of [25]

Let $p_{i,j}$ be the probability that t_i appears at rank j. The dynamic program given in [25] proceeds by computing $p_{i,j}$ for $1 \leq j \leq k$ for each fetched tuple i, with the following termination condition:

$$\max_{1 \leq \ell \leq i} p_{\ell,j} \geq 1 - \sum_{\ell=1}^{i} p_{\ell,j}, \qquad \text{for } 1 \leq j \leq k. \tag{8}$$

This termination is correct in the sense that it will not miss any true answers. However, it is not tight, i.e., it may lead to the scan of unnecessary tuples when the query answers are already known. Consider the following example: $p(t_1) = \ldots = p(t_4) = 1/2, p(t_6) = \ldots = p(t_N) = \epsilon$ for some sufficiently small $\epsilon > 0$. The U-3Ranks query results are clearly (t_1, t_2 or t_3, t_4). The scan depth in this case is $n = 4$, since after reading t_4, we have the current best answers: $p_{1,1} = 1/2$ for rank 1, $p_{2,1} = p_{3,1} = 1/4$ for rank 2, and $p_{4,3} = 3/8$ for rank 3. For any future tuple, it has probability at most $1/16$ to be at rank 1, at most $1/4$ at rank 2, and at most $3/8$ at rank 3 (by Lemma 6). On the other hand, let us consider the condition (8) at $j = 3$. Since $p_{1,3} = p_{2,3} = 0, p_{3,3} = 1/8$, the RHS of (8) is $1/2$ when $i = 4$. As we read more tuples, the LHS of (8) stays at $3/8$, which is achieved by t_4, while the RHS of (8) can be made arbitrarily close to 1/2 by choosing ϵ small enough. Therefore, the algorithm of [25] will read all of the N tuples, although the first $n = 4$ tuples are already sufficient to guarantee the correctness of the results.