Semantics of Ranking Queries for Probabilistic Data and Expected Ranks

Graham Cormode
AT&T Labs Research
Florham Park, NJ, USA

Feifei Li
Computer Science Department
FSU, Tallahassee, FL, USA

Ke Yi
Computer Science & Engineering Department
HKUST, Hong Kong, China

***Abstract*— When dealing with massive quantities of data, top-k queries are a powerful technique for returning only the k most relevant tuples for inspection, based on a scoring function. The problem of efficiently answering such ranking queries has been studied and analyzed extensively within traditional database settings. The importance of the top-k is perhaps even greater in probabilistic databases, where a relation can encode exponentially many possible worlds. There have been several recent attempts to propose definitions and algorithms for ranking queries over probabilistic data. However, these all lack many of the intuitive properties of a top-k over deterministic data. Specifically, we define a number of fundamental properties, including *exact-k*, *containment*, *unique-rank*, *value-invariance*, and *stability*, which are all satisfied by ranking queries on certain data. We argue that all these conditions should also be fulfilled by any reasonable definition for ranking uncertain data. Unfortunately, none of the existing definitions is able to achieve this.**

To remedy this shortcoming, this work proposes an intuitive new approach of *expected rank*. This uses the well-founded notion of the expected rank of each tuple across all possible worlds as the basis of the ranking. We are able to prove that, in contrast to all existing approaches, the expected rank satisfies all the required properties for a ranking query. We provide efficient solutions to compute this ranking across the major models of uncertain data, such as attribute-level and tuple-level uncertainty. For an uncertain relation of N tuples, the processing cost is $O(N \log N)$—no worse than simply sorting the relation. In settings where there is a high cost for generating each tuple in turn, we provide pruning techniques based on probabilistic tail bounds that can terminate the search early and guarantee that the top-k has been found. Finally, a comprehensive experimental study confirms the effectiveness of our approach.

I. Introduction

Ranking queries are a powerful concept in focusing attention on the most important answers to a query. To deal with massive quantities of data, such as multimedia search, streaming data, web data and distributed systems, tuples from the underlying database are ranked by a score, usually computed based on a user-defined scoring function. Only the top-k tuples with the highest scores are returned for further inspection. Following the seminal work by Fagin *et al.* [13], such queries have received considerable attention in traditional relational databases, including [23], [19], [36] and many others. See the excellent survey by Ilyas *et al.* [20] for a more complete overview of the many important studies in this area.

Within these motivating application domains—distributed, streaming, web and multimedia applications—data arrives in massive quantities, underlining the need for ordering by score. But an additional challenge is that the data is also typically inherently fuzzy or uncertain. For instance, multimedia and unstructured web data frequently require data integration or schema mapping [15], [7], [16]. Data items in the output of such operations are usually associated with a confidence, reflecting how well they are matched with other records from different data sources. In applications that handle measurement data, e.g., sensor readings and distances to a query point, the data is inherently noisy, and is better represented by a probability distribution rather than a single deterministic value [9], [11]. In recognition of this aspect of the data, there have been significant research efforts devoted to producing *probabilistic database management systems*, which can represent and manage data with explicit probabilistic models of uncertainty. The notable examples of such systems include MystiQ [10], Trio [1], and MayBMS [2].

With a probabilistic database, it is possible to represent a huge number of possible (deterministic) realizations of the (probabilistic) data—an exponential blow-up from the size of the relation representing the data. A key problem in such databases is how to extend the familiar semantics of the top-k query to this setting, and how to answer such queries efficiently. To this end, there has been several recent works outlining possible definitions, and associated algorithms. Ré *et al.* [28] base their ranking on the confidence associated with each query result. Soliman *et al.* [33] extend the semantics of ranking queries from certain data and study the problem of ranking tuples when there is both a score and probability for each tuple. Subsequently, there have been several other approaches to ranking based on combining score and likelihood [39], [34], [37], [18] (discussed in detail in Section III-B).

For certain data with a single score value, there is a clear total ordering based on score from which the top-k is derived, which leads to a clean and intuitive semantics. This is particularly natural, by analogy with the many occurrences of top-k lists in daily life: movies ranked by box-office receipts, athletes ranked by race times, researchers ranked by number of publications (or other metrics), and so on. With uncertain data, there are two distinct orders to work with: ordering by score, and ordering by probability. There are many possible ways of combining these two, leading to quite different results, as evidenced by the multiple definitions that have been proposed in the literature, such as U-Top$k$ [33], U-kRanks [33], Global-Top$k$ [39] and PT-k [18]. In choosing a definition to work with,

we must ask, what are the conditions that we want the resulting query answer to satisfy. We address this issue following a principled approach and return to the properties of ranking queries on certain data. We define the following properties that should hold on the output of such a ranking query:

- *Exact-k:* The top-k list should contain exactly k items;
- *Containment:* The top-$(k+1)$ list should contain all items in the top-k;
- *Unique-ranking:* Within the top-k, each reported item should be assigned exactly one position: the same item should not be listed multiple times within the top-k.
- *Value-invariance:* The scores only determine the relative behavior of the tuples: changing the score values without altering the relative ordering should not change the top-k;
- *Stability:* Making an item in the top-k list more likely or more important should not remove it from the list.

We define these properties more formally in Section III-A.

These properties are clearly satisfied for certain data, and capture much of our intuition on how a "ranking" query should behave. Moreover, they should seem intuitive and natural (indeed, they should appear almost obvious). A general axiom of work on extending data management from certain data to the uncertain domain has been that basic properties of query semantics should be preserved to the extent possible [10], [4]. But, as we subsequently demonstrate, none of the prior works on ranking queries for probabilistic data satisfies all of these "obvious" properties. Lastly, we note that prior work stated results primarily in the *tuple-level* uncertainty model [1], [10]; here, we show our results for both the tuple-level and *attribute-level* uncertainty models [9], [35].

Our contributions. To remedy the shortcomings we identify, this work proposes an intuitive new approach for ranking based on *expected rank*. It uses the well-founded notion of the expected value of the rank of each tuple across all possible worlds as the basis of the ranking. We are able to prove that, in contrast to all existing approaches, the expected rank satisfies all the required properties for a ranking query across major models of uncertain data. Furthermore, these nice properties do not come at a price of higher computational costs. On the contrary, we design efficient $O(N \log N)$-time exact algorithms to compute under both the attribute-level model and the tuple-level model, while most of the previous top-k definitions rely on dynamic programming and require $\Omega(N^2)$ time to compute the results exactly, and errors have to be tolerated if one wants to process the queries faster by using random sampling or other approximation techniques [17]. In summary, our contributions are the followings:

- We formalize the necessary semantics of ranking queries in certain data and migrate them to probabilistic data (Section III-A), and show that no existing approaches for this problem achieve all these properties (Section III-B).
- We propose a new approach based on the expected rank of each tuple across all possible worlds that provably satisfies these requirements. The expected rank definition works seamlessly with both the *attribute-level* and *tuple-level* uncertainty models (Section III-C).
- We provide efficient algorithms for expected ranks in both models. For an uncertain relation of N tuples, the processing cost of our approach is $O(N \log N)$. In settings where there is a high cost for accessing tuples, we show pruning techniques based on probabilistic tail bounds that can terminate the search early and guarantee that the top-k has been found (Section IV and V).
- We present a comprehensive experimental study that confirms the effectiveness of our approach (Section VII).

II. Uncertain Data Models w.r.t Ranking Queries

Many models for describing uncertain data have been presented in the literature. The work by Sarma *et al.* [29] describes the main features and contrasts their properties and descriptive ability. Each model describes a probability distribution over *possible worlds*, where each possible world corresponds to a single deterministic data instance. The most expressive approach is to explicitly list each possible world and its associated probability; such a method is referred to as *complete*, as it can capture all possible correlations. However, complete models are very costly to describe and manipulate since there can be exponentially many combinations of tuples each generating a distinct possible world [29].

Typically, we are able to make certain *independence assumptions*, that unless correlations are explicitly described, events are assumed to be independent. Consequently, likelihoods can be computed using standard probability calculations (i.e. multiplication of probabilities of independent events). The strongest independence assumptions lead to the *basic model*, where each tuple has a probability of occurrence, and all tuples are assumed fully independent of each other. This is typically too strong an assumption, and so intermediate models allow the description of simple correlations between tuples. This extends the expressiveness of the models, while keeping computations of probability tractable. We consider two models that have been used frequently within the database community. In our discussion, without loss of generality, a probabilistic database contains simply one relation.

Attribute-level uncertainty model. In this model, the probabilistic database is a table of N tuples. Each tuple has one attribute whose value is uncertain (together with other certain attributes). This uncertain attribute has a discrete pdf describing its value distribution. When instantiating this uncertain relation to a certain instance, each tuple draws a value for its uncertain attribute based on the associated discrete pdf and the choice is independent among tuples. This model has many practical applications such as sensor readings [22], [11], spatial objects with fuzzy locations [35], [9], [5], [26], [25], etc. More important, it is very easy to represent this model using the traditional, relational database, as observed by Antova *et al.* [3]. For the purpose of ranking queries, the important case is when the uncertain attribute represents the score for the tuple, and we would like to rank the tuples based on this score attribute. Let X_i be the random variable denoting the score of tuple t_i. We assume that X_i has a discrete pdf with

tuples	score
t_1	$\{(v_{1,1}, p_{1,1}), (v_{1,2}, p_{1,2}), \ldots, (v_{1,s_1}, p_{1,s_1})\}$
t_2	$\{(v_{2,1}, p_{2,1}), \ldots, v_{2,s_2}, p_{2,s_2})\}$
$\vdots$	$\vdots$
t_N	$\{(v_{N,1}, p_{N,1}), \ldots, (v_{N,s_N}, p_{N,s_N})\}$

Fig. 1. Attribute-level uncertainty model.

tuples	score
t_1	$\{(100, 0.4), (70, 0.6)\}$
t_2	$\{(92, 0.6), (80, 0.4)\}$
t_3	$\{(85, 1)\}$

world W	$\Pr[W]$
$\{t_1 = 100, t_2 = 92, t_3 = 85\}$	$0.4 \times 0.6 \times 1 = 0.24$
$\{t_1 = 100, t_3 = 85, t_2 = 80\}$	$0.4 \times 0.4 \times 1 = 0.16$
$\{t_2 = 92, t_3 = 85, t_1 = 70\}$	$0.6 \times 0.6 \times 1 = 0.36$
$\{t_3 = 85, t_2 = 80, t_1 = 70\}$	$0.6 \times 0.4 \times 1 = 0.24$

Fig. 2. An example of possible worlds for attribute-level uncertainty model.

tuples	score	$p(t)$
t_1	v_1	$p(t_1)$
t_2	v_2	$p(t_2)$
$\vdots$	$\vdots$	
t_N	v_N	$p(t_N)$

rules	
τ_1	$\{t_1\}$
τ_2	$\{t_2, t_4\}$
$\vdots$	$\vdots$
τ_M	$\{t_5, t_8, t_N\}$

Fig. 3. Tuple-level uncertainty model.

tuples	score	$p(t)$
t_1	100	0.4
t_2	92	0.5
t_3	80	1
t_4	70	0.5

rules	
τ_1	$\{t_1\}$
τ_2	$\{t_2, t_4\}$
τ_3	$\{t_3\}$

world W	$\Pr[W]$
$\{t_1, t_2, t_3\}$	$p(t_1)p(t_2)p(t_3) = 0.2$
$\{t_1, t_3, t_4\}$	$p(t_1)p(t_3)p(t_4) = 0.2$
$\{t_2, t_3\}$	$(1 - p(t_1))p(t_2)p(t_3) = 0.3$
$\{t_3, t_4\}$	$(1 - p(t_1))p(t_3)p(t_4) = 0.3$

Fig. 4. An example of possible worlds for tuple-level uncertainty model.

bounded size. This is a realistic assumption for many practical applications, including movie ratings [10], and string matching [7]. The general, continuous pdf case is discussed briefly in Section VI. In this model we are essentially ranking the set of independent random variables $X_1, \ldots, X_N$. A relation following this model is illustrated in Figure 1. For tuple t_i, the score takes the value $v_{i,j}$ with probability $p_{i,j}$ for $1 \leq j \leq s_i$.

Tuple-level uncertainty model. In the second model, the attributes of each tuple are fixed, but the entire tuple may or may not appear. In the basic model, each tuple t appears with probability $p(t)$ independently. In more complex models, there are dependencies among the tuples, which can be specified by a set of *generation rules*. These can be in the form of *x-relations* [1], [4], complex events [10], or other forms.

All previous work concerned with ranking queries in uncertain data has focused on the tuple-level uncertainty model with *exclusion rules* [18], [33], [39], [37] where each tuple appears in a single rule τ. Arbitrary generation rules have been discussed in [33], [34], but they have been shown to require exponential processing complexity [18], [37]. Hence, as with many other works in the literature [33], [18], [37], [38], we primarily consider exclusion rules in this model, where each exclusion rule has a constant number of choices. In addition, each tuple appears in at most one rule. The total probability for all tuples in one rule must be less or equal than one, so that it can be properly interpreted as a probability distribution. To simplify our discussion, we allow rules containing only one tuple and require that all tuples must appear in one of the rules. This is essentially equivalent to the popular x-relations model [1]. This tuple-level uncertainty model is a good fit for applications where it is important to capture the correlations between tuples; this model has been used to fit a large number of real-life examples [4], [10], [33], [18], [38]. An example of a relation in this uncertainty model is shown in Figure 3. This relation has N tuples and M rules. The second rule says that t_2 and t_4 cannot appear together in any certain instance of this relation. It also constrains that $p(t_2) + p(t_4) \leq 1$.

The possible world semantics. We denote the uncertain relation as $\mathcal{D}$. In the attribute-level uncertainty model, an uncertain relation is instantiated into a *possible world* by taking one independent value for each tuple's uncertain attribute according to its distribution. Denote a possible world as W and the value for t_i's uncertain attribute in W as w_{t_i}. In the attribute-level uncertainty model, the probability that W occurs is $\Pr[W] = \prod_{j=1}^{N} p_{j,x}$, where x satisfies $v_{j,x} = w_{t_j}$. It is worth mentioning that in the attribute-level case we always have $\forall W \in \mathcal{W}, |W| = N$, where $\mathcal{W}$ is the space of all the possible worlds. The example in Figure 2 illustrates the possible worlds for an uncertain relation in this model.

For the tuple-level uncertainty model, a possible world W from $\mathcal{W}$ is now a *subset* of tuples from the uncertain relation $\mathcal{D}$. 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

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

A notable difference for the tuple-level uncertain model is that given a random possible world W, not all tuples from $\mathcal{D}$ will appear. Hence, the size of the possible world can range from 0 to N. The example in Figure 4 illustrates the possible worlds for an uncertain relation in this model.

We iterate that every uncertain data model can be seen as a succinct description of a distribution over possible worlds $\mathcal{W}$. Each possible world is a certain table on which we can evaluate any traditional query. The focus of uncertain query processing is (1) how to "combine" the query results from all the possible worlds into a meaningful result for the query; and (2) how to process such a combination efficiently without explicitly materializing the exponentially many possible worlds.

Difference of the two models under ranking queries. We would like to emphasize that there is a significant difference for the two models in the context of *ranking tuples*. More

specifically, the semantic of ranking queries in uncertain databases is to derive a meaningful ordering for all tuples in the database $\mathcal{D}$. Note that this is not equivalent to deriving an ordering for all values that tuples in $\mathcal{D}$ may take. In the attribute-level model, all tuples in $\mathcal{D}$ will participate in the ranking process in every possible world. In contrast, in the tuple-level model only a subset of tuples in $\mathcal{D}$ will participate in the ranking process for a given possible world.

III. Ranking Query Semantics

A. Properties of Ranking Queries

We now define a set of properties for ranking tuples. These are chosen to describe the key properties of ranking certain data, and hence to give properties which a user would naturally expect of a ranking over uncertain data to have.

The first property is very natural, and is also used in [39].

Definition 1 (Exact-k): Let R_k be the set of tuples (associated with their ranks) in the top-k query result. If $|\mathcal{D}| \geq k$, then $|R_k| = k$.

The second property captures the intuition that if an item is in the top-k, it should be in the top-k' for any $k' > k$. Equivalently, the choice of k is simply a slider that chooses how many results are to be returned to the user, and changing k should only change the number of results returned, not the underlying set of results.

Definition 2 (Containment): For any k, $R_k \subset R_{k+1}$. ■

Replacing "$\subset$" with "$\subseteq$", gives the *weak containment* property.

The next property stipulates that the rank assigned to each tuple in the top-k list should be unique.

Definition 3 (Unique ranking): Let $r_k(i)$ be the identity of the tuple from the input assigned rank i in the output of the ranking procedure. The *unique ranking* property requires that $\forall i \neq j . r_k(i) \neq r_k(j)$. ■

The next property captures the semantics that the score function is assumed to only give a relative ordering, and is not an absolute measure of the value of a tuple.

Definition 4 (Value invariance): Let $\mathcal{D}$ denote the relation which includes score values $v_1 \leq v_2 \leq \ldots$. Let s'_i be any set of score values satisfying $v'_1 \leq v'_2 \leq \ldots$, and define $\mathcal{D}'$ to be $\mathcal{D}$ with all scores v_i replaced with v'_i. The *value invariance* property requires that $R_k(\mathcal{D}) = R_k(\mathcal{D}')$ for any k. ■

For example, consider the relation with tuple-level uncertainty illustrated in Figure 4. Here, the scores are $70 \leq 80 \leq 92 \leq 100$. The value invariance property demands that we could replace these scores with, say, $1 \leq 2 \leq 3 \leq 1000$, and the result of the ranking would still be the same.

Finally, Zhang and Chomicki [39] proposed the *stability* condition in the tuple-level uncertainty model[1]. We adopt this property and generalize it to the attribute-level model:

Definition 5 (Stability): In the tuple-level model, given a tuple $t_i = (v_i, p(t_i))$ from $\mathcal{D}$, if we replace t_i with $t_i^{\uparrow} = (v_i^{\uparrow}, p(t_i^{\uparrow}))$ where $v_i^{\uparrow} \geq v_i, p(t_i^{\uparrow}) \geq p(t_i)$, then

$$t_i \in R_k(\mathcal{D}) \Rightarrow t_i^{\uparrow} \in R_k(\mathcal{D}'),$$

where $\mathcal{D}'$ is obtained by replacing t_i with $t_i^{\uparrow}$ in $\mathcal{D}$.

[1]The *faithfulness* property from [39] is discussed in Section VI.

For the attribute-level model, the statement for stability remains the same but with $t_i^{\uparrow}$ defined as follows. Given a tuple t_i whose score is a random variable X_i, we obtain $t_i^{\uparrow}$ by replacing X_i with a random variable $X_i^{\uparrow}$ that is *stochastically greater or equal than* [31] X_i, denoted as $X_i^{\uparrow} \succeq X_i$. ■

Stability captures the intuition that if a tuple is already in the top-k, making it "probabilistically larger" should not eject it. Stability also implies that making a non-top-k probabilistically smaller should not bring it into the top-k.

Note, these conditions make little explicit reference to probability models, and can apply to almost any ranking setting. They trivially hold for the top-k semantics over certain data. Yet perhaps surprisingly, none of the existing definitions for top-k over uncertain data satisfy these natural requirements!

B. Top-k Queries on Probabilistic Data

We now consider how to extend ranking queries to uncertain data. Details differ slightly for the two uncertainty models: In the attribute-level model, a tuple has a random score but it always exists in any random possible world, i.e., every tuple participates in the ranking process in all possible worlds, and we rank these N tuples based on their score distribution. In contrast, in the tuple-level model, a tuple has a fixed score but it may not always appear, i.e., it may not participate in the ranking process in some possible worlds. We still aim to produce a ranking on all N tuples, taking this into account.

Considering the tuple-level model, the difficulty of extending ranking queries to probabilistic data is that there are now two distinct orderings present in the data: that given by the score, and that given by the probabilities. These two types of information need to be combined in some way to produce the top-k (this can be orthogonal to the model used to describe the uncertainty in the data). We now detail a variety of approaches that have been taken, and discuss their shortcomings with respect to the conditions we have defined. The key properties are summarized in Figure 5.

Combine two rankings. There has been much work on taking multiple rankings and combining them (e.g. taking the top 50 query web search results from multiple search engines, and combining them to get an overall ranking) based on minimizing disagreements [12]. Likewise, skyline-based approaches extract points which do not dominate each other, and are not themselves dominated, under multiple ordered dimensions [6]. But such approaches fail to account for the inherent semantics of the probability distribution: it is insufficient to treat it simply as an ordinal attribute, as this loses the meaning of the relative likelihoods, and does not guarantee our required properties.

Most likely top-k. Since a probabilistic relation can define exponentially many possible worlds, one approach to the top-k problem finds the top-k set that has the highest support over all possible worlds. In other words, (conceptually) extract the top-k from each possible world, and compute the support (probability) of each distinct top-k set found. The *U-Topk* approach [33] reports the most likely top-k as the answer to the ranking query. This method has the advantage that it more

Ranking method	*Exact-k*	*Containment*	*Unique-Rank*	*Value-Invariant*	*Stability*
U-top$k$ [33]	×	×	✓	✓	✓
U-kRanks [33], [24]	✓	✓	×	✓	×
PT-k [18]	×	weak	✓	✓	✓
Global-top$k$ [39]	✓	×	✓	✓	✓
Expected score	✓	✓	✓	×	✓
Expected rank	✓	✓	✓	✓	✓

Fig. 5. Summary of Ranking Methods for Uncertain Data

directly incorporates the likelihood information, and satisfies unique ranking, value invariance, and stability. But it may not always return k tuples when $\mathcal{D}$ is small, as also pointed out in [39]. More importantly, it violates the containment property. In fact, there are simple examples where the top-k can be completely disjoint from the top-$(k+1)$. Consider the attribute-level model example in Figure 2. The top-1 result under the U-Top$k$ definition is t_1, since its probability of having the highest score in a random possible world is 0.24+0.16=0.4, larger than that of t_2 or t_3. However, the top-2 result is (t_2, t_3), whose probability of being the top-2 is 0.36, larger than that of (t_1, t_2) or (t_1, t_3). Thus, the top-2 list is completely disjoint from the top-1. Similarly one can verify that for the tuple-level model example in Figure 4, the top-1 result is t_1 but the top-2 is (t_2, t_3) or (t_3, t_4). No matter what tie-breaking rule is used, the top-2 is completely disjoint from the top-1.

Most likely tuple at each rank. The previous approach fails because it deals with top-k sets as immutable objects. Instead, we could consider the property of a certain tuple being ranked kth in a possible world. In particular, let $X_{i,j}$ be the event that tuple j is ranked i within a possible world. Computing $\Pr[X_{i,j}]$ for all i, j pairs, this approach reports the ith result as $\arg\max_j \Pr[X_{i,j}]$, i.e., the tuple that is most likely to be ranked ith over all possible worlds. This is the *U-kRanks* approach [33]; essentially the same definition is proposed as *PRank* in [24] and analyzed in the context of distributions over spatial data. This definition overcomes the shortcomings of U-Top$k$ and satisfies exact-k and containment. However, it fails on unique ranking, as one tuple may dominate multiple ranks at the same time. A related issue is that some tuples may be quite likely, but never get reported. So in Figure 2, the top-3 under this definition is t_1, t_3, t_1: t_1 appears twice and t_2 never; for Figure 4, there is a tie for the third position, and there is no fourth placed tuple, even though $N = 4$. These issues have also been pointed out in [18], [39]. In addition, it fails on stability, as shown in [39], since when the score of a tuple becomes larger, it may leave its original rank but cannot take over any higher ranks as the dominating winner.

Rank by top-k probability. Attempting to patch the previous definition, we can replace the event "tuple i is at rank k" with the event "tuple i is at rank k or better", and reason about the probability of this event. That is, define the top-k probability of a tuple as the probability that it is in the top-k over all possible worlds. The *probabilistic threshold top-k* query (*PT-k* for short) returns the set of all tuples whose top-k probability exceeds a user-specified probability p [18]. However, for a user specified p, the "top-k" list may not contain k tuples, violating exact-k. If we fix p and increase k, the top-k lists do expand, but they only satisfy the weak containment property. For instance consider the tuple-level example in Figure 2. If we set $p = 0.4$, then the top-1 list is (t_1). But both the top-2 and top-3 lists contain the same set of tuples: t_1, t_2, t_3. A further drawback of using PT-k for ranking is that user has to specify the threshold p which greatly affects the result.

Similarly, the *Global-Topk* method ranks the tuples by their top-k probability, and then takes the top-k of these [39] based on this probability. This makes sure that exactly k tuples are returned, but it again fails on containment. In Figure 2, under the Global-Top$k$ definition, the top-1 is t_1, but the top-2 is (t_2, t_3). In Figure 4, the top-1 is t_1, but the top-2 is (t_3, t_2).

Further, note that as k increases towards N, then the importance of the score approaches zero, and these two methods reduce to simply ranking by probability alone.

Expected score. The above approaches all differ from traditional ranking queries, in that they do not define a single ordering of the tuples from which the top-k is taken—in other words, they do not resemble "top-k" in the literal interpretation of the term. A simple approach in this direction is to just compute the expected score of each tuple, and rank by this score, then take the top-k. It is easy to check that such an approach directly implies exact-k, containment, unique ranking, and stability. However, this is very dependent on the values of the scores: consider a tuple which has very low probability but a score that is orders of magnitude higher than others—then it gets propelled to the top of the ranking, since it has the highest expected score, even though it is unlikely. But if we reduce this score to being just greater than the next highest score, the tuple will drop down the ranking. It therefore violates value invariance. Furthermore, in the tuple-level model, simply using the expected score ignores all the correlation rules completely.

C. Ranking by Expected Ranks

Motivated by the deficiencies of existing definitions, we propose a new ranking method which we call *expected rank*. The intuition is that top-k over certain data is defined by first providing a total ordering of the tuples, and then selecting the k "best" tuples under the ordering. Any such definition immediately provides the containment and unique-ranking properties. After rejecting expected score due to its sensitivity to the score values, a natural candidate is the expected rank of the tuple over the possible worlds. More formally,

Definition 6 (Expected Rank): The rank of a tuple t_i in a possible world W is defined to be the number of tuples whose score is higher than t_i (so the top tuple has rank 0), i.e.,

$$\text{rank}_W(t_i) = |\{t_j \in W | v_j > v_i\}|$$

In the attribute-level uncertain model, we compute the expected rank $r(t_i)$ as above and then return the top-k tuples

with the lowest $r(t_i)$. More precisely,

$$r(t_i) = \sum_{W \in \mathcal{W}, t_i \in W} \Pr[W] \cdot \text{rank}_W(t_i) \quad (1)$$

In the tuple-level model, we have to define how to handle possible worlds where t_i does not appear. For such a world W where t_i does not appear, we define $\text{rank}_W(t_i) = |W|$, i.e. we imagine that it follows after all the appearing tuples. So,

$$\begin{aligned} r(t_i) &= \sum_{t_i \in W} \Pr[W] \, \text{rank}_W(t_i) + \sum_{t_i \notin W} \Pr[W] \cdot |W| \quad (2) \\ &= \sum_{W \in \mathcal{W}} \Pr[W] \, \text{rank}_W(t_i), \end{aligned}$$

where $\text{rank}_W(t_i)$ is defined to be $|W|$ if $t_i \notin W$. ∎

For the example in Figure 2, the expected rank for t_2 is $r(t_2) = 0.24 \times 1 + 0.16 \times 2 + 0.36 \times 0 + 0.24 \times 1 = 0.8$. Similarly $r(t_1) = 1.2$, $r(t_3) = 1$. So the final ranking is (t_2, t_3, t_1). For the example in Figure 4, $r(t_2) = 0.2 \times 1 + 0.2 \times 3 + 0.3 \times 0 + 0.3 \times 2 = 1.4$. Note that t_2 does not appear in the second and the fourth worlds, so its ranks are taken to be 3 and 2, respectively. Similarly $r(t_1) = 1.2$, $r(t_3) = 0.9$, $r(t_4) = 1.9$. So the final ranking is (t_3, t_1, t_2, t_4).

We now prove some properties of this definition. For simplicity, we assume that the expected ranks are unique, and so the ranking forms a total ordering. In practice, ties can be broken arbitrarily e.g. based on having the lexicographically smaller id. The same tie-breaking issues affect the ranking of certain data as well.

Theorem 1: Expected rank satisfies exact-k, containment, unique ranking, value invariance, and stability.

Proof: The first three properties follow immediately from the fact that the expected rank is used to give an ordering. Value invariance follows by observing that changing the score values will not change the rankings in possible worlds, and therefore does not change the expected ranks.

For stability we show that when we change a tuple t_i to $t_i^{\uparrow}$, its expected rank will not increase, while the expected rank of any other tuple will not decrease. Let r' be the expected rank in the uncertain relation $\mathcal{D}'$ after changing t_i to $t_i^{\uparrow}$. We need to show that $r(t_i) \geq r'(t_i^{\uparrow})$ and $r(t_{i'}) \leq r'(t_{i'})$ for any $i' \neq i$.

Consider the attribute-level model first. By definition 6 and linearity of expectation, we have

$$\begin{aligned} r(t_i) &= \sum_{j \neq i} \Pr[X_i < X_j] = \sum_{j \neq i} \sum_{\ell} p_{j,\ell} \Pr[X_i < v_{j,\ell}] \\ &\geq \sum_{j \neq i} \sum_{\ell} p_{j,\ell} \Pr[X_i^{\uparrow} < v_{j,\ell}] \quad \text{(because } X_i \preceq X_i^{\uparrow}\text{)} \\ &= \sum_{j \neq i} \Pr[X_i^{\uparrow} < X_j] = r'(t_i^{\uparrow}). \end{aligned}$$

For any $i' \neq i$,

$$\begin{aligned} r(t_{i'}) &= \Pr[X_{i'} < X_i] + \sum_{j \neq i', j \neq i} \Pr[X_{i'} < X_j] \\ &= \sum_{\ell} p_{i',\ell} \Pr[v_{i',\ell} < X_i] + \sum_{j \neq i', j \neq i} \Pr[X_{i'} < X_j] \\ &\leq \sum_{\ell} p_{i',\ell} \Pr[v_{i',\ell} < X_i^{\uparrow}] + \sum_{j \neq i', j \neq i} \Pr[X_{i'} < X_j] \\ &= \Pr[X_{i'} < X_i^{\uparrow}] + \sum_{j \neq i', j \neq i} \Pr[X_{i'} < X_j] = r'(t_{i'}) \end{aligned}$$

Next consider the tuple-level model. If $t_i^{\uparrow}$ has a larger score than t_i but the same probability, then $r(t_i) \geq r'(t_i^{\uparrow})$ follows easily from (2) since $\text{rank}_W(t_i)$ can only get smaller while the second term of (2) remains unchanged. For similar reasons, $r(t_{i'}) \leq r'(t_{i'})$ for any $i' \neq i$.

If $t_i^{\uparrow}$ has the same score as t_i but a larger probability, $\text{rank}_W(t_i)$ stays the same for any possible world W, but $\Pr[W]$ may change. We divide all the possible worlds into three categories: (a) those containing t_i, (b) those containing one of the tuples in the exclusion rule of t_i (other than t_i), and (c) all other possible worlds. Note that $\Pr[W]$ does not change for any W in category (b), so we only focus on categories (a) and (c). Observe that there is a one-to-one mapping between the possible worlds in category (a) and (c): $W \rightarrow W \cup \{t_i\}$. For each such pair, its contribution to $r(t_i)$ is

$$\Pr[W] \cdot |W| + \Pr[W \cup \{t_i\}] \cdot \text{rank}_W(t_i). \quad (3)$$

Suppose the tuples in the exclusion rule of t_i are $t_{i,1}, \ldots, t_{i,s}$. Note that W and $W \cup \{t_i\}$ differs only in the inclusion of t_i, so we can write $\Pr[W] = \pi \left(1 - \sum_{\ell} p(t_{i,\ell}) - p(t_i)\right)$ and $\Pr[W \cup \{t_i\}] = \pi p(t_i)$ for some π. When $p(t_i)$ increases to $p(t_i^{\uparrow})$, the increase in (3) is

$$\begin{aligned} & \pi(p(t_i) - p(t_i^{\uparrow}))|W| + \pi(p(t_i^{\uparrow}) - p(t_i)) \, \text{rank}_W(t_i) \\ = \; & \pi(p(t_i) - p(t_i^{\uparrow}))(|W| - \text{rank}_W(t_i)) \leq 0. \end{aligned}$$

The same holds for each pair of possible worlds in categories (a) and (c). Therefore we have $r(t_i) \geq r'(t_i^{\uparrow})$.

For any $i' \neq i$, the contribution of each pair is

$$\Pr[W] \cdot \text{rank}_W(t_{i'}) + \Pr[W \cup \{t_i\}] \cdot \text{rank}_{W \cup \{t_i\}}(t_{i'}). \quad (4)$$

When $p(t_i)$ increases to $p(t_i^{\uparrow})$, the increase in (4) is

$$\pi(p(t_i) - p(t_i^{\uparrow}))(\text{rank}_W(t_{i'}) - \text{rank}_{W \cup \{t_i\}}(t_{i'})) \geq 0.$$

The same holds for each pair of possible worlds in categories (a) and (c). Therefore we have $r'(t_{i'}) \geq r(t_{i'})$. ∎

IV. Expected Ranks in the Attribute-Level Uncertainty Model

This section presents efficient algorithms for calculating the expected rank of an uncertain relation $\mathcal{D}$ with N tuples in the attribute-level uncertain model. We first show an exact algorithm that can calculate the expected ranks of all tuples in $\mathcal{D}$ with $O(N \log N)$ processing cost. We then propose an approximate algorithm that can terminate the search as soon as the top-k tuples with the k smallest expected ranks are guaranteed to be found without accessing all tuples.

Algorithm 1: A-ERank($\mathcal{D}, k$)

```
Create U containing values from t_1.X_1, ..., t_N.X_N, in order;
Compute q(v) ∀v ∈ U by one pass over U;
Initialize a priority queue A sorted by expected rank;
for i = 1, ..., N do
    Compute r(t_i) using q(v)'s and X_i using Eqn. (6);
    Insert (t_i, r(t_i)) into A;
    if |A| > k then Drop element with largest expected rank from A;
return A;
```

A. Exact Computation

By Definition 6 and linearity of expectation, we have

$$r(t_i) = \sum_{i \neq j} \Pr[X_j > X_i]. \tag{5}$$

The brute-force search (BFS) approach requires $O(N)$ time to compute $r(t_i)$ for one tuple and $O(N^2)$ time to compute the ranks of all tuples. The quadratic dependence on N is prohibitive when N is large. Below we present an improved algorithm that runs in $O(N \log N)$ time. We observe that (5) can be written as:

$$\begin{aligned} r(t_i) &= \sum_{i \neq j} \sum_{\ell=1}^{s_i} p_{i,\ell} \Pr[X_j > v_{i,\ell}] = \sum_{\ell=1}^{s_i} p_{i,\ell} \sum_{j \neq i} \Pr[X_j > v_{i,\ell}] \\ &= \sum_{\ell=1}^{s_i} p_{i,\ell} \big(\sum_{j} \Pr[X_j > v_{i,\ell}] - \Pr[X_i > v_{i,\ell}] \big) \\ &= \sum_{\ell=1}^{s_i} p_{i,\ell} \big(q(v_{i,\ell}) - \Pr[X_i > v_{i,\ell}] \big), \end{aligned} \tag{6}$$

where we define $q(v) = \sum_j \Pr[X_j > v]$. Let U be the universe of all possible values of X_i, $i = 1, \ldots, N$. Because we assume each pdf has constant size bounded by s, we have $|U| \leq |sN|$. When s is a constant, we have $|U| = O(N)$.

Now observe that we can precompute $q(v)$ for all $v \in U$ with a linear pass over the input after sorting U which has a cost of $O(N \log N)$. Following (6), exact computation of the expected rank for a single tuple can now be done in constant time given $q(v)$ for all $v \in U$. While computing these expected ranks, we maintain a priority queue of size k that stores the k tuples with smallest expected ranks dynamically. When all tuples have been processed, the contents of the priority queue are returned as the final answer. Computing $q(v)$ takes time $O(N \log N)$; getting expected ranks of all tuples while maintaining the priority queue takes $O(N \log k)$ time. Hence, the overall cost of this approach is $O(N \log N)$. We denote this algorithm as *A-ERrank* and describe it in Algorithm 1.

B. Pruning by Expected Scores

A-ERank is very efficient even for large N values. However, in certain scenarios accessing a tuple is considerably expensive (if it requires significant IO access). It then becomes desirable to reduce the number of tuples accessed in order to find the answer. It is possible to find a set of (possibly more than k tuples) which is guaranteed to include the true top-k expected ranks, by pruning based on tail bounds of the score distribution. If tuples are sorted in decreasing order of their expected scores, i.e. $\mathrm{E}[X_i]$'s, we can terminate the search early. In the following discussion, we assume that if $i < j$, then $\mathrm{E}[X_i] \geq \mathrm{E}[X_j]$ for all $1 \leq i, j \leq N$. Equivalently, we can think of this as an interface which generates each tuple in turn, in decreasing order of $\mathrm{E}[X_i]$.

The pruning algorithm scans these tuples, and maintains an upper bound on $r(t_i)$, denoted $r^+(t_i)$, for each t_i seen so far, and a lower bound on $r(t_u)$ for any unseen tuple t_u, denoted r^-. The algorithm halts when there are at least k $r^+(X_i)$'s that are smaller than r^-. Suppose n tuples $t_1, \ldots, t_n$ have been scanned. For $\forall i \in [1, n]$, we have:

$$\begin{aligned} r(t_i) &= \sum_{j \leq n, j \neq i} \Pr[X_j > X_i] + \sum_{n < j \leq N} \Pr[X_j > X_i] \\ &= \sum_{j \leq n, j \neq i} \Pr[X_j > X_i] + \sum_{n < j \leq N} \sum_{\ell=1}^{s_i} p_{i,\ell} \Pr[X_j > v_{i,\ell}] \\ &\leq \sum_{j \leq n, j \neq i} \Pr[X_j > X_i] + \sum_{n < j \leq N} \sum_{\ell=1}^{s_i} p_{i,\ell} \frac{\mathrm{E}[X_j]}{v_{i,\ell}} \\ &\qquad \text{(Markov Inequality)} \\ &\leq \sum_{j \leq n, j \neq i} \Pr[X_j > X_i] + (N - n) \sum_{\ell=1}^{s_i} p_{i,\ell} \frac{\mathrm{E}[X_n]}{v_{i,\ell}}. \end{aligned} \tag{7}$$

The first term in (7) can be computed using only the seen tuples $t_1, \ldots, t_n$. The second term could be computed using X_i and X_n. Hence, from the scanned tuples, we can maintain an upper bound on $r(t_i)$ for each tuple in $\{t_1, \ldots, t_n\}$, i.e., we can set $r^+(t_i)$ to be (7) for $i = 1, \ldots, n$. The second term in $r^+(t_i)$ is updated for every newly scanned tuple t_n (as well as the first term for t_n).

Now we provide the lower bound r^-. Consider any unseen tuple $t_u, u > n$, we have:

$$\begin{aligned} r(t_u) &\geq \sum_{j \leq n} \Pr[X_j > X_u] = n - \sum_{j \leq n} \Pr[X_u \geq X_j] \\ &= n - \sum_{j \leq n} \sum_{\ell=1}^{s_j} p_{j,\ell} \Pr[X_u > v_{j,\ell}] \\ &\geq n - \sum_{j \leq n} \sum_{\ell=1}^{s_j} p_{j,\ell} \frac{\mathrm{E}[X_n]}{v_{j,\ell}}. \qquad \text{(Markov Ineq.)} \end{aligned} \tag{8}$$

This holds for any unseen tuple. Hence, we set r^- to be (8). Note that (8) only depends on the seen tuples. It is updated with every new tuple t_n.

These bounds lead immediately to an algorithm that maintains $r^+(t_i)$'s for all tuples $t_1, \ldots, t_n$ and r^-. For each new tuple t_n, the $r^+(t_i)$'s and r^- are updated. From these, we find the kth largest $r^+(t_i)$ value, and compare this to r^-. If it is less, then we know for sure that k tuples with smallest expected ranks *globally* are among the first n tuples, and can stop retrieving tuples. Otherwise, we move on to the next tuple. We refer to this algorithm as *A-ERank-Prune*.

A remaining challenge is how to find the k tuples with the smallest expected ranks using the first n tuples alone. This turns out to be difficult as it is not possible to obtain a precise order on their final ranks without inspecting all the N tuples in $\mathcal{D}$. Instead, we use the curtailed database $\mathcal{D}' = \{t_1, \ldots, t_n\}$, and compute the exact expected rank $r'(t_i)$ of every tuple (for $i \in [1, n]$) t_i in $\mathcal{D}'$. The rank $r'(t_i)$ turns out to be an excellent surrogate for $r(t_i)$ for $i \in [1, n]$ in $\mathcal{D}$ (when the pruning algorithm terminates after processing n tuples). Hence, we return the top-k of these as the result of the query. We omit a detailed analysis of the quality of this approach, and instead show an empirical evaluation in our experimental study.

A straightforward implementation of *A-ERrank-Prune* requires $O(n^2)$ time. After seeing t_n, the bounds in both (7) and (8) can be updated in constant time, by retaining $\sum_{\ell=1}^{s_j} \frac{p_{i,\ell}}{v_{i,\ell}}$ for each seen tuple. The challenge is to update the first term in (7) for all $i \leq n$. A basic approach requires linear time, for adding $\Pr[X_n > X_i]$ to the already computed $\sum_{j \leq n-1, j \neq i} \Pr[X_j > X_i]$ for all i's as well as computing $\sum_{i \leq n-1} \Pr[X_i > X_n]$). This leads to a total running time of $O(n^2)$ for algorithm A-ERrank-Prune. Using a similar idea in designing algorithm A-ERank, we could utilize the value universe U' of all the seen tuples and maintain prefix sums of the $q(v)$ values, which would drive down the cost of this step to $O(n \log n)$. We omit full details for space reasons.

V. Expected Ranks in the Tuple-Level Uncertainty Model

We now consider ranking an uncertain database $\mathcal{D}$ in the *tuple-level uncertainty model* . For $\mathcal{D}$ with N tuples and M rules, the aim is to retrieve the k tuples with the smallest expected ranks. Recall that each rule τ_j is a set of tuples, where $\sum_{t_i \in \tau_j} p(t_i) \leq 1$. Without loss of generality we assume the tuples $t_1, \ldots, t_n$ are already sorted by the ranking attribute and t_1 is the tuple with the highest score. We use $t_i \diamond t_j$ to denote that t_i and t_j are in the same exclusion rule and $t_i \neq t_j$; we use $t_i \bar{\diamond} t_j$ to denote that t_i and t_j are not in the same exclusion rule. We first give an exact algorithm with $O(N \log N)$ complexity that accesses every tuple. Secondly, we show a pruning algorithm with $O(n \log n)$ complexity, that only reads the first n tuples, assuming that the *expected* number of tuples in $\mathcal{D}$ is known to the algorithm.

A. Exact computation

From Definition 6, in particular (2), given tuples that are sorted by their score attribute, we have:

$$r(t_i) = p(t_i) \cdot \sum_{t_j \bar{\diamond} t_i, j<i} p(t_j) + (1 - p(t_i)) \cdot \left(\frac{\sum_{t_j \diamond t_i} p(t_j)}{1 - p(t_i)} + \sum_{t_j \bar{\diamond} t_i} p(t_j) \right).$$

The first term computes t_i's expected rank for random worlds when it appears, and the second term computes the expected size of a random world W when t_i does not appear in W. The term $\frac{\sum_{t_j \diamond t_i} p(t_j)}{1 - p(t_i)}$ is the expected number of appearing tuples in the same rule as t_i, conditioned on t_i not appearing, while $\sum_{t_j \bar{\diamond} t_i} p(t_j)$ accounts for the rest of the tuples. Rewriting,

$$r(t_i) = p(t_i) \cdot \sum_{t_j \bar{\diamond} t_i, j<i} p(t_j) + \sum_{t_j \diamond t_i} p(t_j) + (1 - p(t_i)) \cdot \sum_{t_j \bar{\diamond} t_i} p(t_j). \quad (9)$$

Let $q_i = \sum_{j<i} p(t_j)$. We first compute q_i in $O(N)$ time. At the same time, we find the expected number of tuples, $\mathrm{E}[|W|] = \sum_{j=1}^{N} p(t_j)$. Now (9) can be rewritten as:

$$r(t_i) = p(t_i) \cdot (q_i - \sum_{t_j \diamond t_i, j<i} p(t_j)) + \sum_{t_j \diamond t_i} p(t_j) + (1 - p(t_i))(\mathrm{E}[|W|] - p(t_i) - \sum_{t_j \diamond t_i} p(t_j)). \quad (10)$$

By keeping the auxiliary information $\sum_{t_j \diamond t_i, j<i} p(t_j)$ (i.e., the sum of probabilities of tuples that have score values higher than t_i in the same rule as t_i) and $\sum_{t_j \diamond t_i} p(t_j)$ (i.e., the sum of probabilities of tuples that are in the same rule as t_i) for each tuple t_i in $\mathcal{D}$, $r(t_i)$ can be computed in $O(1)$ time. By maintaining a priority queue of size k that keeps the k tuples with the smallest $r(t_i)$'s, we can select the top-k tuples in $O(N \log k)$ time. Note that both $\sum_{t_j \diamond t_i, j<i} p(t_j)$ and $\sum_{t_j \diamond t_i} p(t_j)$ are cheap to calculate initially given all the rules in a single scan of the relation (time $O(N)$). When $\mathcal{D}$ is not presorted by t_i's score attribute, the running time of this algorithm is dominated by the sorting step, $O(N \log N)$. Algorithm 2 gives pseudo-code for this algorithm, T-ERrank.

Algorithm 2: T-ERank($\mathcal{D}, k$)

```
Sort D by score attribute s.t. if t_i.v_i ≥ t_j.v_j, then i ≤ j;
Compute q_i ∀i ∈ [1, N] and E[|W|] by one pass over D;
Initialize a priority queue A sorted by expected rank;
for i = 1, ..., N do
    Compute r(t_i) using (10);
    if |A| > k then drop element with largest expected rank
    from A;
return A;
```

B. Pruning

Provided that the expected number of tuples $\mathrm{E}[|W|]$ is known, we can answer top-k queries more efficiently using pruning techniques without accessing all tuples. Note that $\mathrm{E}[|W|]$ can be efficiently maintained in $O(1)$ time when $\mathcal{D}$ is updated with deletion or insertion of tuples. As $\mathrm{E}[|W|]$ is simply the sum of all the probabilities (note that it does not depend on the rules), it is reasonable to assume that it is always available. Similar to the attribute-level uncertainty case, we assume that $\mathcal{D}$ provides an interface to retrieve tuples in order of their score attribute from the highest to the lowest.

The pruning algorithm scans the tuples in order. After seeing t_n, it can compute $r(t_n)$ exactly using $\mathrm{E}[|W|]$ and q_n in $O(1)$ time based on (10). It also maintains $r^{(k)}$, the k-th smallest $r(t_i)$ among all the tuples that have been retrieved. This can

be done with a priority queue in $O(\log k)$ time per tuple. A lower bound on $r(t_\ell)$ for any $\ell > n$ is computed as follows:

$$\begin{aligned} r(t_\ell) =& p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell, j<\ell} p(t_j) \\ &+ \sum_{t_j \diamond t_\ell} p(t_j) + (1 - p(t_\ell)) \cdot \sum_{t_j \bar{\diamond} t_\ell} p(t_j) \quad \text{(from (9))} \\ =& p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell, j<\ell} p(t_j) + \mathrm{E}[|W|] - p(t_\ell) - p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell} p(t_j) \\ =& \mathrm{E}[|W|] - p(t_\ell) - p(t_\ell) \cdot \left(\sum_{t_j \bar{\diamond} t_\ell} p(t_j) - \sum_{t_j \bar{\diamond} t_\ell, j<\ell} p(t_j) \right) \\ =& \mathrm{E}[|W|] - p(t_\ell) - p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell, j>\ell} p(t_j). \end{aligned} \tag{11}$$

In the second step, we used the fact that
$\sum_{t_j \diamond t_\ell} p(t_j) + \sum_{t_j \bar{\diamond} t_\ell} p(t_j) = \mathrm{E}[|W|] - p(t_\ell)$.
Now, since $q_\ell = \sum_{j<\ell} p(t_j)$, we observe that

$$\mathrm{E}[|W|] - q_\ell = \sum_{j>\ell} p(t_j) + p(t_\ell) \geq \sum_{t_j \bar{\diamond} t_\ell, j>\ell} p(t_j).$$

Continuing with (11), we have:

$$\begin{aligned} r(t_\ell) &\geq \mathrm{E}[|W|] - p(t_\ell) - p(t_\ell) \cdot (\mathrm{E}[|W|] - q_\ell) \\ &\geq q_\ell - 1 \geq q_n - 1. \end{aligned} \tag{12}$$

The last step uses the monotonicity of q_i—by definition, $q_n \leq q_\ell$ if $n \leq \ell$. Since tuples are scanned in order, obviously $\ell > n$.

Thus, when $r^{(k)} \leq q_n - 1$, we know for sure there are at least k tuples amongst the first n with expected ranks smaller than all unseen tuples. At this point, we can safely terminate the search. In addition, recall that for all the scanned tuples, their expected ranks are calculated *exactly* by (10). Hence this algorithm—which we dub *T-ERank-Prune*—can simply return the current top-k tuples. From the above analysis, its time cost is $O(n \log k)$ where n is potentially much smaller than N.

VI. Extensions

Scoring functions. Our analysis has assumed that the score is a fixed value. In general, the score can be specified at query time by a user defined function. Note that our offline algorithms also work under this setting, as long as the scores can be computed. If the system has some interface that allows us to retrieve tuples in the score order (for the tuple-level order) or in the expected score order (for the attribute-level model), our pruning algorithms are applicable as well.

A main application of a query-dependent scoring function is k-nearest-neighbor queries, which is the top-k query instantiated in spatial databases. Here, the score is implicitly the distance of a data point to a query point. When the data points are uncertain, the distance to the query is a random variable, which can be modeled as an attribute-level uncertainty relation. Existing works [8], [24] essentially adopt U-kRanks semantics to define k-nearest-neighbor queries in spatial databases. We believe that the expected rank definition makes a lot of sense in this context, and may have similar benefits over previous definitions of uncertain nearest neighbors.

When a relation has multiple (certain and uncertain) attributes on which a ranking query is to be performed, the user typically will give some function that combines this multiple attributes together and then rank on the output of the function. When at least one of the attributes is uncertain, the output of the function is also uncertain. This gives us another instance where our ranking semantics and algorithms could be applied.

Continuous distributions. When the input data in the attribute-level uncertainty model is specified by a continuous distribution (e.g. a Gaussian or Poisson), it is often hard compute the probability that one variable exceeds another. However, by discretizing the distributions to an appropriate level of granularity (i.e., represented by a histogram), we can reduce to an instance of the discrete pdf problem. The error in this approach is directly related to the granularity of the discretization. Moreover, observe that our pruning-based methods initially require only information about expected values of the distributions. Since continuous distributions are typically described by their expected value (e.g., a Gaussian distribution is specified by its mean and variance), we can run the pruning algorithm on these parameters directly.

Further properties of a ranking. The ranking properties we define and study in Section III-A are by no means a complete characterization; rather, we argue that they are a minimum requirement for a ranking. Further properties can be defined and analyzed, although care is needed in their formulation. For example, Zhang and Chomicki [39] define the "faithfulness" property, which demands that (in the tuple-level model), given two tuples $t_1 = (v_1, p(t_1))$ and $t_2 = (v_2, p(t_2))$ with $v_1 < v_2$ and $p(t_1) < p(t_2)$, then $t_1 \in R_k \Rightarrow t_2 \in R_k$. This intuitive property implies that if t_2 "dominates" t_1, then t_2 should always be ranked higher than t_1. However, there are examples where all existing definitions fail to guarantee faithfulness. Consider the relation:

t_i	t_1	t_2	t_3	t_4	t_5
v_i	1	2	3	4	5
$p(t_i)$	0.4	0.45	0.2	0.2	0.2

with rules $\tau_1 = \{t_1, t_3, t_4, t_5\}, \tau_2 = \{t_2\}$. Here, t_2 "dominates" t_1, but all prior definitions (U-top$k$, U-$k$ranks, Global-top$k$, and PT-k) select t_1 is as the top-1. On this example, the expected rank definition will rank t_2 as the top-1, but unfortunately there are other examples where expected rank will also rank a dominating tuple lower than a dominated tuple. Our interpretation is that "faithfulness" defined this way may not be an achievable property, and one has to somehow take rules into consideration in order to make it a viable property.

Limitation of expected ranks. Our expected rank definition uses the expectation as the basis of ranking, i.e., the absolute ranks of each tuple from all possible worlds are represented by their mean. It is well known that the mean is statistically sensitive to the distribution of the underlying values (in our case, the absolute ranks of the tuple from all possible worlds). Hence, a more general and statistically more stable approach might be to use the median instead of the mean. This can be

generalized to any quantile of the collection of absolute ranks for a tuple and derive the final ranking based on such quantiles. It remains an open problem to efficiently compute both the median-rank and the quantile-rank (for any quantile value). Likewise, it will also be important to study the semantics of these definitions, and how they compare to expected rank.

VII. Experiments

We implemented our algorithms in GNU C++. All experiments were executed on a Linux machine with a 2GHz CPU and 2GB main memory. In order to study the impact of data sets with different characteristics on both the score value distribution and the probability distribution, we focused on synthetic data sets. We additionally tested our algorithms on real data sets from the MystiQ project, and the trends there were similar to those reported here on synthetic data. We developed several data generators for both attribute-level and tuple-level uncertain models. Each generator controls the distribution on the score value as well as the probability. For both models, these distributions refer to the *universe of score values and probabilities* when we take the union of all tuples in $\mathcal{D}$.[2] The distributions used include *uniform, Zipfian* and *correlated bivariate*. They are abbreviated as *u, zipf* and *cor*. For each tuple, we draw a score and probability value independently from the score distribution and probability distribution respectively. We refer to the result of drawing from these two distributions by the concatenation of the short names for each distribution for score then probability. For example, *uu* indicates a data set with uniform distributions for both score values and probabilities; *zipfu* indicates a Zipfian distribution of score values and uniform distribution on the probabilities. The default the skewness parameter for the Zipfian distribution is 1.2, and the default value of $k = 100$.

A. Attribute-level Uncertainty Model

We first studied the performance of the exact algorithm A-ERank by comparing it to the basic brute-force search (BFS) approach. The distribution on the probability universe does not affect the performance of both algorithms, since both algorithms calculate the expected ranks of all tuples. The score value distribution has no impact on BFS, but does affect A-ERank: the uniform score distribution results in the worst performance given a fixed number of tuples, as it leads to a large set of possible values. So we used uu data sets for this experiment, to give the toughest test for this algorithm.

The score of each tuple is given by a pdf with 5 unique choices (i.e., $s = 5$). Figure 6(a) shows the total running time of these two algorithms as the size of $\mathcal{D}$ (i.e. the number of tuples, N) is varied, up to $100,000$ tuples. A-ERank outperforms BFS by up to six orders of magnitude. This gap grows steadily as N gets larger. A-ERank has very low query cost: it takes only about 10ms to find all tuples expected ranks for $N = 100,000$, while the brute force approach takes ten minutes. Results are similar for other values of s.

[2]For the attribute-level model, this includes all the value and probability pairs that a tuple's pdf has on its uncertain attribute.

Figure 6(b) shows the pruning power of A-ERank-Prune. In this experiment N is set to $100,000$, with $s = 5$ and k is varied from 10 to 100. It shows that we often only need to materialize a small number of tuples of $\mathcal{D}$ (ordered by expected score) before we can be sure that we have found the top-k, across a variety of data sets. Intuitively, a more skewed distribution on either dimension should increase the algorithm's pruning power. This intuition is confirmed by the results in Figure 6(b). When both distributions are skewed, A-ERank-Prune could halt the scan after seeing less than 20% of the relation. Overall, this shows that expected scores hold enough information to prune, even for more uniform distributions.

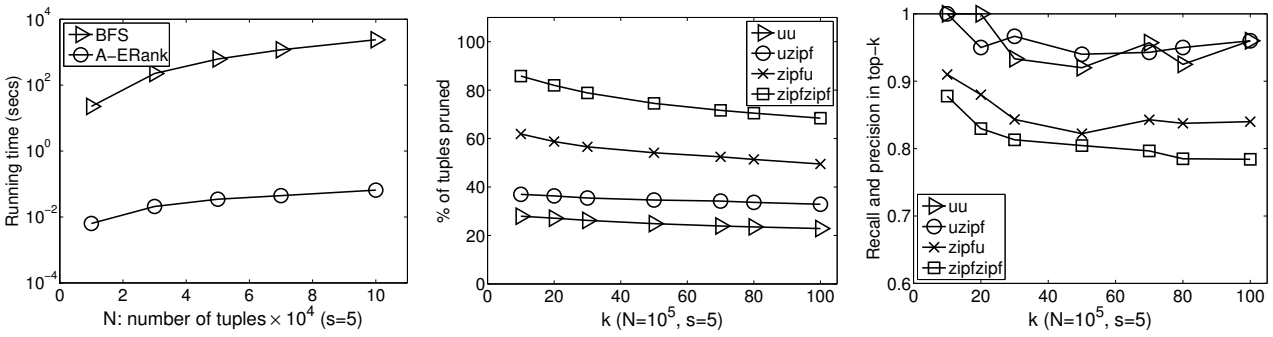
As discussed in Section IV-B, A-ERank-Prune is an approximate algorithm, in that it may not find the exact top-k. Figure 6 reports its approximation quality on various data sets using the standard *precision* and *recall* metrics. Since A-ERank-Prune always returns k tuples, its recall and precision are always the same. Figure 6 shows that it achieves high approximation quality: recall and precision are both in the 90th percentile when the score is distributed uniformly. The worst case occurs when the data is skewed on both dimensions, where the potential for pruning is greatest. The reason for this is that as more tuples are pruned, these unseen tuples have a greater chance to affect the expected ranks of the observed tuples. Even though the pruned tuples all have low expected scores, they could still have values with high probability to be ranked above some seen tuples, because of the heavy tail of their distribution. Even in this worst case, the recall and precision of T-ERank-Prune is about 80%.

B. Tuple-level Uncertainty Model

For our experiments in the tuple-level uncertainty model, where rules determine exclusions between tuples, we show results on data sets where 30% of tuples are involved in rules with other tuples. Experiments with a greater or lesser degree of correlation gave similar results. We first investigate the performance of our algorithms. As before, there is also a brute-force search based approach, and it is also much more expensive than our algorithms, so we do not show these results.

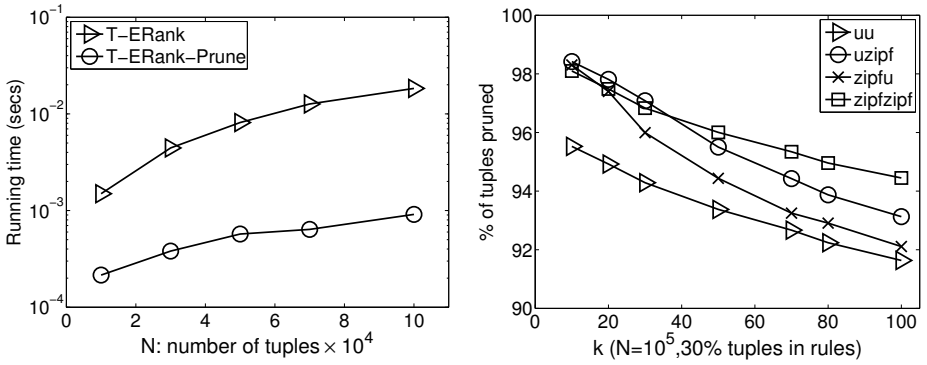
A notable difference in this model is that the pruning algorithm is able to output the exact top-k, provided that $\mathrm{E}[|W|]$, the expected number of tuples of $\mathcal{D}$, is known. Figure 7(a) shows the total running time for the T-ERank and T-ERank-Prune algorithms using *uu* data. Both algorithms are extremely efficient. For $100,000$ tuples, the T-ERank algorithm takes 10 milliseconds to compute the expected ranks of all tuples; applying pruning, T-ERank-Prune finds the same k smallest ranks in just 1 millisecond. However, T-ERank is still highly efficient, and is the best solution when $\mathrm{E}[|W|]$ is unavailable.

Figure 7(b) shows the pruning power of T-ERank-Prune for different data sets. We fix $N = 100,000$ and vary the k values. Clearly, a skewed distribution on either dimension increases the pruning capability of T-ERank-Prune. More importantly, even in the worst case of processing the *uu* data set, T-ERank-Prune is able to prune more than 90% of tuples.



(a) Running time of exact algorithms (b) Pruning of A-ERank-Prune. (c) Precision and Recall of A-ERank-Prune.

Fig. 6. Attribute-level uncertain model: performance analysis.



(a) Running times of T-ERank and T-ERank-Prune (b) Pruning of T-ERank-Prune.

Fig. 7. Tuple-level uncertain model: performance analysis.

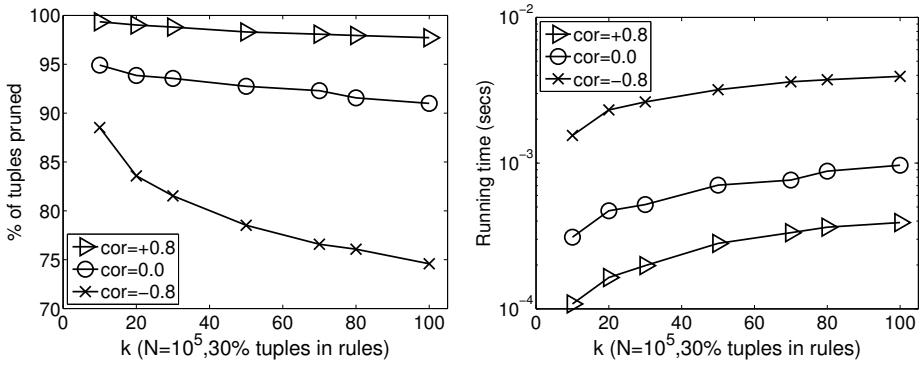
Our final set of experiments studies the impact of correlations between tuple's score value and probability. We say that the two are positively correlated when a tuple with higher score value also has a higher probability; a negative correlation means that higher score means lower probability. Such correlations have no impact on the performance of T-ERank as it computes the expected ranks for all tuples. However, correlation does have an interesting effect on the pruning capability of T-ERrank-Prune. Using correlated bivariate data sets of different correlation degrees, Figure 8(a) repeats the pruning experiment for T-ERank-Prune with $N = 100,000$. The strongly positively correlated data set with a $+0.8$ correlation degree allows the highest amount of pruning, whereas the strongly negatively correlated data set with a -0.8 correlation degree results in the worst pruning power. But even in that worst case, T-ERank-Prune still pruned more than 75% of tuples. Figure 8(b) reflects the running time of the same experiment. T-ERank-Prune consumes between 0.1 and 5 milliseconds to process $100,000$ uncertain tuples.

VIII. Background on Querying Uncertain Data

Much effort has been devoted to modeling and processing uncertain data, so we survey only the most related work. TRIO [1], [4], [29], MayBMS [2] and MystiQ [10] are promising systems that are currently being developed. General query processing techniques have been extensively studied under the possible worlds semantics [9], [10], [14], [21], and important query types with specific query semantics are explored in more depth, skyline queries [27] and heavy hitters [38]. Indexing and nearest neighbor queries under the attribute-level uncertain model have also been explored [25], [32], [35], [5], [9], [26].

Section III-B discusses the most closely related works on answering top-k queries on uncertain databases [18], [33], [39], [37]. Techniques used have included the Monte Carlo approach of sampling possible worlds [28], AI-style branch-and-bound search of the probability state space [33], dynamic programming approaches [37], [39], [17], and applying tail (Chernoff) bounds to determine when to prune [18]. There is ongoing work to understand semantics of top-k queries in a variety of contexts. For example, the work of Lian and Chen [24] deals with ranking objects based on spatial uncertainty, and ranking based on linear functions. Recently, Soliman *et al.* [34] have extended their study on top-k queries [33] to Group-By aggregate queries.

Our study on the tuple-level uncertainty model limits us to considering correlations in the form of mutual exclusions. More advanced rules and processing techniques may be needed for complex correlations. Recent works based on graphical probabilistic models and Bayesian networks have shown promising results in both offline [30] and streaming data [22]. In these situations, initial approaches are based on Monte-Carlo simulations [21], [28].



(a) Pruning power. (b) Running time.

Fig. 8. Impact of tuple's core and probability correlations on T-ERank-Prune.

IX. Conclusion

We have studied the semantics of ranking queries in probabilistic data. We adapt important properties that guide the definition of ranking queries in traditional, relational databases and analyze the limitations of existing top-k ranking queries for probabilistic data. These properties naturally lead to the expected rank approach in uncertain domain. Efficient algorithms for two major models of uncertain data ensure the practicality of the expected rank. Our experiments convincingly demonstrate that ranking by expected ranks is very efficient in both attribute-level and tuple-level uncertainty models.

References

[1] 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.

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

[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *ICDE*, 2008.

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

[5] G. Beskales, M. A. Soliman, and I. F. Ilyas, "Efficient search for the top-k probable nearest neighbors in uncertain databases," in *VLDB*, 2008.

[6] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001.

[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, J. Chen, M. Mokbel, and C.-Y. Chow, "Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data," in *ICDE*, 2008.

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

[10] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.

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

[12] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *WWW Conference*, 2001.

[13] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *PODS*, 2001.

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

[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*, vol. 2, no. 1, pp. 9–37, 1998.

[17] M. Hua, J. Pei, W. Zhang, and X. Lin, "Efficiently answering probabilistic threshold top-k queries on uncertain data," in *ICDE*, 2008.

[18] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *SIGMOD*, 2008.

[19] I. F. Ilyas, W. G. Aref, A. K. Elmagarmid, H. Elmongui, R. Shah, and J. S. Vitter, "Adaptive rank-aware query optimization in relational databases," *TODS*, vol. 31, 2006.

[20] I. F. Ilyas, G. Beskales, and M. A. Soliman, "Survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, 2008.

[21] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas, "MCDB: a monte carlo approach to managing uncertain data," in *SIGMOD*, 2008.

[22] B. Kanagal and A. Deshpande, "Online filtering, smoothing and probabilistic modeling of streaming data," in *ICDE*, 2008.

[23] C. Li, K. C.-C. Chang, I. Ilyas, and S. Song, "RankSQL: Query algebra and optimization for relational top-k queries," in *SIGMOD*, 2005.

[24] X. Lian and L. Chen, "Probabilistic ranked queries in uncertain databases," in *EDBT*, 2008.

[25] V. Ljosa and A. Singh, "APLA: Indexing arbitrary probability distributions," in *ICDE*, 2007.

[26] V. Ljosa and A. K. Singh, "Top-k spatial joins of probabilistic objects," in *ICDE*, 2008.

[27] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *VLDB*, 2007.

[28] C. Re, N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic databases," in *ICDE*, 2007.

[29] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working models for uncertain data," in *ICDE*, 2006.

[30] P. Sen and A. Deshpande, "Representing and querying correlated tuples in probabilistic databases," in *ICDE*, 2007.

[31] J. G. Shanthikumar and M. Shaked, *Stochastic Orders and Their Applications*. Academic Press, 1994.

[32] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch, "Indexing uncertain categorical data," in *ICDE*, 2007.

[33] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *ICDE*, 2007.

[34] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Probabilistic top-k and ranking-aggregate queries," *TODS*, vol. 33, no. 3, 2008.

[35] 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.

[36] D. Xin, J. Han, and K. C.-C. Chang, "Progressive and selective merge: Computing top-k with ad-hoc ranking functions," in *SIGMOD*, 2007.

[37] K. Yi, F. Li, D. Srivastava, and G. Kollios, "Efficient processing of top-k queries in uncertain databases with x-relations," *IEEE TKDE*, vol. 20, no. 12, pp. 1669–1682, 2008.

[38] Q. Zhang, F. Li, and K. Yi, "Finding frequent items in probabilistic data," in *SIGMOD*, 2008.

[39] X. Zhang and J. Chomicki, "On the semantics and evaluation of top-k queries in probabilistic databases," in *DBRank*, 2008.