Semantics of Ranking Queries for Probabilistic Data

Jeffrey Jestes, Graham Cormode, Feifei Li, and Ke Yi

Abstract—Recently, there have been several attempts to propose definitions and algorithms for ranking queries on probabilistic data. However, these lack many intuitive properties of a top-k over deterministic data. We define numerous fundamental properties, including *exact-*k, *containment*, *unique-rank*, *value-invariance*, and *stability*, which are satisfied by ranking queries on certain data. We argue these properties should also be carefully studied in defining ranking queries in probabilistic data, and fulfilled by definition for ranking uncertain data for most applications. We propose an intuitive new ranking definition based on the observation that the ranks of a tuple across all possible worlds represent a well-founded rank distribution. We studied the ranking definitions based on the expectation, the median and other statistics of this rank distribution for a tuple and derived the *expected rank, median rank and quantile rank* correspondingly. We are able to prove that the expected rank, median rank and quantile rank satisfy all these properties for a ranking query. We provide efficient solutions to compute such rankings across the major models of uncertain data, such as attribute-level and tuple-level uncertainty. Finally, a comprehensive experimental study confirms the effectiveness of our approach.

Index Terms—probabilistic data, ranking queries, top-k queries, uncertain database.

1 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.* [15], such queries have received considerable attention in traditional relational databases, including [28], [24], [45] and many others. See the excellent survey by Ilyas *et al.* [25] 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 data is typically inherently fuzzy or uncertain. For instance, multimedia and unstructured web data frequently require data integration or schema mapping [19], [9], [20]. 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 [11], [13]. 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. Some notable examples of such systems include MystiQ [12], Trio [1], Orion [39] and MayBMS [2].

With a probabilistic database, it is possible to compactly 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 have been several recent works outlining possible definitions, and associated algorithms. Ré *et al.* [34] base their ranking on the confidence associated with each query result. Soliman *et al.* [42] 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 [48], [43], [46], [23] (discussed in detail in Section 4.2).

For certain data with a single score value, there is a clear total ordering based on their scores from which the top-k is derived, which leads to 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 which have been proposed in the literature, such as U-Top$k$ [42], U-kRanks [42], Global-Top$k$ [48] and PT-k [23]. In choosing a definition, we must ask, what conditions do we want the resulting query answer to satisfy. We address this following a principled approach, returning to ranking query properties on certain data. We provide the following properties which are desirable on the output of a ranking query as a minimum:

- *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

Jeffrey Jestes and Feifei Li are with the School of Computing, University of Utah. ({jestes,lifeifei}@cs.utah.edu)
Graham Cormode is with AT&T Labs-Research. (graham@research.att.com)
Ke Yi is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. (yike@cse.ust.hk)

should not be listed multiple times within the top-k.
- *Stability:* Making an item in the top-k list more likely or more important should not remove it from the list.
- *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;

We define these properties more formally in Section 4.1. These properties are satisfied for certain data, and capture much of our intuition on how a "ranking" query should behave. 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 best extent possible [12], [4]. But, as we demonstrate, none of the prior works on ranking queries for probabilistic data has systematically examined these properties and studied whether a ranking definition satisfies them. It should be noted these five ranking properties are by no means a complete characterization for ranking uncertain data. Nevertheless, it is an interesting and important problem to search for meaningful definitions that satisfy at least these properties.

Lastly, we note prior work stated results primarily in the *tuple-level* model [1], [12]; here, we show results for both *tuple-level* and *attribute-level* models [11], [44].

Our contributions. To remedy the shortcomings we identify, we propose an intuitive new approach for ranking based on the rank distribution of a tuple's ranks across all possible worlds. Using this well-founded rank distribution as the basis of the ranking, we study ranking definitions based on typical statistical values over a distribution. Specifically,

- We formalize some important semantics of ranking queries on certain data and migrate them to probabilistic data (Section 4.1), and systematically examine the characteristics of existing approaches for this problem with respect to these properties (Section 4.2).
- We propose a new approach based on the distribution of each tuple's ranks across all possible worlds. By leveraging statistical properties on such a rank distribution, such as the expectation, the median and the quantile, we derive the expected rank, the median rank and quantile rank. We are able to show that the new definitions provably satisfy these requirements. These new definitions work seamlessly with both the *attribute-level* and *tuple-level* uncertainty models (Section 4.3).
- We provide efficient algorithms for expected ranks in both models. For an uncertain relation of N constant-sized tuples, the processing cost of expected ranks is $O(N \log N)$ for both models. 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 5 and 6).
- We study additional properties guaranteed by median and quantile ranks and present dynamic programs for computing them. The formulations are different in the *attribute-level* and the *tuple-level* models, however, they are similar for the median and different quantile values. For an uncertain relation of N tuples, the processing cost of our algorithm is $O(N^3)$ in the *attribute-level* model, and $O(NM^2)$ in the *tuple-level* model where M is the number of rules in the database (Section 7).
- We present a comprehensive experimental study that confirms the effectiveness of our approach for various ranking definitions (Section 8).
- We discuss other issues related to this work in Appendix A, e.g., continuous functions, further properties of a ranking and the interesting relationship between our study and [29] which proposes a general framework for imposing different ranking definitions in probabilistic data.

2 Background

Much effort has been devoted to modeling and processing uncertain data, so we survey only the most related work. TRIO [1], [4], [35], MayBMS [2], Orion [39], [38] and MystiQ [12] are promising systems currently being developed. General query processing techniques have been extensively studied under the possible worlds semantics [11], [12], [17], [26], and important query types with specific semantics are explored in more depth, skyline queries [33] and heavy hitters [47]. Indexing and nearest neighbor queries under the attribute-level model have also been explored [31], [40], [44], [6], [11], [32].

Section 4.2 discusses the most closely related works on answering top-k queries on uncertain databases [23], [42], [48], [46]. Techniques have included the Monte Carlo approach of sampling possible worlds [34], AI-style branch-and-bound search of the probability state space [42], dynamic programming approaches [46], [48], [22], and applying tail (Chernoff) bounds to determine when to prune [23]. There is ongoing work to understand semantics of top-k queries in a variety of contexts. For example, the work of Lian and Chen [30] deals with ranking objects based on spatial uncertainty, and ranking based on linear functions. Ge *et al.* [18] presented a detailed study on finding the typical vectors that effectively sample the score distribution from the top-k query results in uncertain databases. Recently, Soliman *et al.* [43] extended their study on top-k queries [42] to Group-By aggregate queries, and to the case when scores give a partial order, instead of a total order [41]. A general framework for imposing various ranking definitions in probabilistic data was recently proposed by Li *et al.* [29]. We discussed the relationship between this study and that work in Appendix A.

Our study on the tuple-level model limits us to considering correlations in the form of mutual exclusions. More advanced rules and processing may be needed for complex correlations. Recent works based on graphical probabilistic models and Bayesian networks have shown promising results in both offline [36] and streaming data [27]. In these situations, initial approaches are based on Monte-Carlo simulations [26], [34].

3 Uncertain Data Models

Many models for describing uncertain data have been presented in the literature. The work by Das Sarma *et al.* [35] 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 [35].

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 (finite) 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 [27], [13], spatial objects with fuzzy locations [44], [11], [6], [32], [31], etc. More important, it is very easy to represent this model using the traditional relational database model, 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 bounded size s_i. This is a realistic assumption for many practical applications, including movie ratings [12], and string matching [9]. The general, continuous pdf case is discussed in Appendix A. 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 $1 \leq j \leq s_i$.

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.

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

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.

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 [12], or other forms.

All previous work concerned with ranking queries in uncertain data has focused on the tuple-level uncertainty model with *exclusion rules* [23], [42], [48], [46] where each tuple appears in a single rule τ. Each rule τ lists a set of tuples that are mutually exclusive so that at most one of these can appear in any possible world. Arbitrary generation rules have been discussed in [42], [43], but they have been shown to require exponential processing complexity [23], [46]. Hence, as with many other works in the literature [42], [23], [46], [47], 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 to 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 appear in (exactly) one of the rules. This is essentially equivalent to the popular x-relations model [1]. This tuple-level model is a good fit for applications where it is important to capture correlations between tuples; this model has been used to fit a large number of real-life examples [4], [12], [42], [23], [47]. Examples of a relation in this model are shown in Figures 3 and 4. 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 $p(t_2) + p(t_4) \leq 1$.

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.

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 rule τ that applies to $\mathcal{D}$, $p_W(\tau)$ 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 emphasize that there is a significant difference for the two models in the context of *ranking tuples*. More specifically, the goal 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. In particular, although there are mappings between relations in the attribute-level and tuple-level models, these have different sets of tuples to rank (often, with different cardinalities). As such, this means that there is no simple reduction between the two cases, and different algorithmic solutions are needed for each. It remains a tantalizing prospect to make further use of structural similarities between the two models to design a unified approach for ranking in both. We hope this can be addressed in future work.

4 Ranking Query Semantics

4.1 Properties of Ranking Queries

We now define a set of properties for ranking tuples. These are chosen to describe key properties of ranking certain data, and hence give properties which a user would naturally expect of a ranking over uncertain data. These properties should be seen largely as *desirable* but by no means *sufficient* for a ranking. Our main purpose in introducing them is to make them explicit, demonstrate prior definitions do not adhere to them all, and provoke discussion about which properties should hold in general for proposed ranking methods.

The first property is very natural, and is also used in [48].

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 $\forall i \neq j, r_k(i) \neq r_k(j)$. ■

Recently, Zhang and Chomicki [48] proposed the *stability* condition in the tuple-level uncertainty model. We adopt this property and generalize it to the attribute-level model:

Definition 4 (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 $\mathcal{D}$.

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* [37] X_i, denoted as $X_i^{\uparrow} \succeq X_i$, meaning $\Pr(X_i^{\uparrow} \geq x) \geq \Pr(X_i \geq x)$ for all $x \in (-\infty, \infty)$. ■

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 tuple probabilistically smaller should not bring it into the top-k.

The final 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 5 (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. ■

Discussion of Value Invariance. The value-invariance property is defined as it (trivially) holds in the deterministic setting. It is more debatable if it should always be enforced over uncertain data. The argument against value-invariance notably arises when the score may have an intuitive linear interpretation (e.g. when measuring financial profits, twice the profit is considered twice as good). In such scenarios, value

invariance can ignore the "common sense" meaning of the scores and lead to counter-intuitive results. For these cases, it is clearly preferable to choose a method which does *not* obey this property, and instead define an appropriate requirement which captures the given semantics of the score value.

Nevertheless, we argue this property is important to consider for a number of reasons. There are many cases when the score has no such linear interpretation. For example, consider the general case where there is no explicit score value revealed to the algorithm; instead, for any pair of (deterministic) tuples, there is an "oracle" which reports which ranks above the other. This encodes a total order. Then any method which can operate in this setting must necessarily obey value invariance, whereas methods which rely on being given a score value will be unable to operate. Other examples arise when scores arise from outputs from the sum of classification algorithms, and so have no linear property. Instead, we only have that a larger total score is preferable. Here (as in the deterministic ranking case), the ranking should be invariant under different score values which give the same total ordering. 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.

Whether or not value invariance is considered desirable in a given ranking situation, it is important to know if a proposed ranking method will guarantee the property or not. It is perhaps surprising to note that *all* existing ranking definitions [42], [23], [30], [48] for probabilistic data have this property.

Properties and Probabilities. Observe that 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. It should nevertheless be noted that these properties are not meant to be a complete characterization of ranking queries in probabilistic data. Indeed, in some cases, a specific application may only require a subset of these conditions. However, in order to choose a ranking definition to work with for different domain requirements, it is imperative to examine the semantics of ranking queries for probabilistic data, especially in the context of real-world applications, and to understand which ranking definitions provide which properties. The intricate interplay between the score and the probability attributes indicates that no single definition will be a universal best choice for all applications. Nevertheless, we believe that many natural situations will require all these five simple properties to hold.

4.2 Top-k Queries on Probabilistic Data

We now consider how to extend ranking queries to uncertain data. The two uncertainty models require different approaches: 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 meaningful 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.

Ignore one dimension. A baseline approach is to ignore one dimension (score or likelihood). If we ignore likelihood it becomes an instance of ranking certain data. The work of Ré *et al.* [34] studies the case where there is no score, and instead ranks the results of a query solely by their probability (across all possible worlds). However, when there is both score and probability information available, ignoring one dimension is insufficient for most purposes. Such simple methods may trivially satisfy the above five basic properties, but they fail to meaningfully combine information in the input. They are easily shown to lead to undesirable features, such as ranking very low probability tuples above much more probable ones.

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 [14], [16]. Likewise, skyline-based approaches extract points which do not dominate each other, and are not themselves dominated, under multiple ordered dimensions [7]. 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-Top$k$* approach [42] reports the most likely top-k as the answer to the ranking query (that is, the top-k set with the highest total probability across all worlds). This method has the advantage that it more 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 [48]. 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 [42]; essentially the same definition is proposed as *PRank* in [30] 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 [23], [48]. In addition, it fails on stability, as shown in [48], 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 [23]. 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 the user has to specify the threshold p which greatly affects the result.

Similarly, the *Global-Top$k$* method ranks the tuples by their top-k probability, and then takes the top-k of these [48] 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 diminishes, so these two methods reduce to simply ranking the reported top-k items 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. This method may be desirable when the score has a strong linear interpretation (e.g. it represents a financial profit), but it does not apply in the "oracle model" where only the relative ordering of each pair of tuples is given. It is easy to check that the expected score 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.

4.3 The Rank Distribution and Expected Ranks

Motivated by deficiencies of existing definitions, we propose a new ranking framework that depends on the ranks of a tuple across all possible worlds and we refer to these ranks (for a given tuple t), together with the corresponding possible worlds' probabilities, as t's rank distribution. Our 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 (i.e. it does not provide value invariance), a natural candidate is to consider the orderings based on the *ranks* of the tuple over the possible worlds. More formally,

Definition 6 (Ranks of a tuple in all possible worlds): The rank of tuple t_i in a possible world W is defined to be the number of tuples whose score is higher than t_i (the top tuple has rank 0), i.e., $\text{rank}_W(t_i) = |\{t_j \in W | v_j > v_i\}|$. In the tuple-level model, for a world W where t_i does not appear, we define $\text{rank}_W(t_i) = |W|$, i.e. it follows after all appearing tuples. ■

The ranks of a tuple t_i in all possible worlds and the probabilities of all worlds constitute a proper probability distribution function (pdf): $\text{rank}(t_i)$, i.e., $\text{rank}(t_i) = \{(\text{rank}_W(t_i), \Pr[W])\}$ for $\forall W \in \mathcal{W}$, since $\sum_{W \in \mathcal{W}} \Pr[W] = 1$. Note to form a well-defined pdf we need to combine (sum up the corresponding probabilities) the ranks from different possible worlds that have the same value (i.e., $\text{rank}_W(t_i)$) from the above set. Formally, let $R(t_i)$ be a random variable for the rank of tuple t_i in a random selected possible world,

Definition 7 (Rank Distribution): The rank distribution of a tuple t_i, $\text{rank}(t_i)$, is a proper probability distribution function (pdf) for the random variable $R(t_i)$ defined as:

$$\text{rank}(t_i) : \Pr[R(t_i) = V] = \sum_{W \in \mathcal{W} | \text{rank}_W(t_i) = V} \Pr[W], \quad \forall V \in [0, N] \blacksquare$$

The rank distribution for a tuple captures important information on how a tuple behaves in terms of ranking across

Ranking method	Exact-k	Containment	Unique-Rank	Value-Invariant	Stability
U-top$k$ [42]	×	×	✓	✓	✓
U-kRanks [42], [30]	✓	✓	×	✓	×
PT-k [23]	×	weak	✓	✓	✓
Global-top$k$ [48]	✓	×	✓	✓	✓
Expected score	✓	✓	✓	×	✓
Expected rank	✓	✓	✓	✓	✓

Fig. 5. Summary of Ranking Methods for Uncertain Data

all possible worlds. Applying statistical operators to each distribution to generate a single statistic is a natural way to summarize the rank distribution, and can be used as the basis for ranking. We first study the *expectation*, which leads to a new ranking method, which we call the *expected rank*.

Definition 8 (Expected Rank): The expected rank of tuple t_i is the expectation of $\mathrm{rank}(t_i)$. The smaller $\mathrm{rank}(t_i)$'s expectation, the smaller t_i's final rank, denoted as $r(t_i)$.

In the attribute-level model, the expected rank $r(t_i)$ can be computed as the expectation on $\mathrm{rank}_W(t_i)$'s, then the top-k tuples with the lowest $r(t_i)$ can be returned. More precisely,

$$r(t_i) = \mathrm{E}[R(t_i)] = \sum_{W \in \mathcal{W}, t_i \in W} \Pr[W] \cdot \mathrm{rank}_W(t_i) \qquad (1)$$

In the tuple-level model, in a world W where t_i does not appear, $\mathrm{rank}_W(t_i) = |W|$, i.e. we imagine it follows after all the tuples which do appear (as per Definition 6 above). So,

$$r(t_i) \quad = \quad \sum_{W \in \mathcal{W}} \Pr[W]\, \mathrm{rank}_W(t_i), \qquad (2)$$

where the definition of $\mathrm{rank}_W(t_i)$ is extended so that $\mathrm{rank}_W(t_i) = |W|$ if $t_i \notin W$. (*Proof in Appendix B*). ■

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 $r(t_1) = 1.2$, $r(t_3) = 1$, and 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 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 prove expected ranks satisfies all five fundamental properties in Appendix B. Specifically,

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

5 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 uncertainty 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 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.

5.1 Exact Computation

By Definition 8 and the linearity of expectation, we have

$$r(t_i) = \sum_{j \neq i} \Pr[X_j > X_i]. \qquad (3)$$

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 ranks of all tuples. The quadratic dependence on N is prohibitive for large N. Below we present an improved algorithm requiring $O(N \log N)$ time. We observe that (3) can be written as:

$$r(t_i) = \sum_{\ell=1}^{s_i} p_{i,\ell} \big(q(v_{i,\ell}) - \Pr[X_i > v_{i,\ell}] \big), \qquad (4)$$

where we define $q(v) = \sum_j \Pr[X_j > v]$ (*Proof in Appendix B*). Let U be the universe of all possible values of all X_i, $i = 1, \ldots, N$. Because we assume each pdf has 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 (4), 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 Appendix C.

5.2 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 $r^+(t_i)$'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 (*Proof in Appendix B*):

$$r(t_i) \leq \sum_{j \leq n, j \neq i} \Pr[X_j > X_i] + (N - n) \sum_{\ell=1}^{s_i} p_{i,\ell} \frac{\mathrm{E}[X_n]}{v_{i,\ell}}. \qquad (5)$$

The first term in (5) can be computed using only seen tuples $t_1, \ldots, t_n$. The second term could be computed using X_i and X_n. Hence, from 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 (5) for $i = 1, \ldots, n$. $r^+(t_i)$'s second term is updated for every new 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 (*Proof in Appendix B*):

$$r(t_u) \quad \geq \quad n - \sum_{j \leq n} \sum_{\ell=1}^{s_j} p_{j,\ell} \frac{\mathrm{E}[X_n]}{v_{j,\ell}}. \tag{6}$$

This holds for any unseen tuple. Hence, we set r^- to be (6). Note that (6) 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 the k tuples with the 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 is difficult as it is not possible to obtain a precise order on their final ranks without inspecting all N tuples in $\mathcal{D}$. Instead, we use the curtailed database $\mathcal{D}' = \{t_1, \ldots, t_n\}$, and compute exact expected rank $r'(t_i)$ for 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 show an evaluation of the quality of this approach in our experimental study.

A straightforward implementation of *A-ERrank-Prune* requires $O(n^2)$ time. After seeing t_n, the bounds in both (5) and (6) 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 updating the first term in (5) 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 complexity of $O(n^2)$ for algorithm A-ERrank-Prune. Using a similar idea in designing algorithm A-ERank, it is possible to utilize value universe U' of all seen tuples and maintain prefix sums of the $q(v)$ values, which would drive down the cost of this step to $O(n \log n)$.

6 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 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 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 t_i and t_j are not in the same exclusion rule. We first give an exact $O(N \log N)$ algorithm which accesses every tuple. Secondly, we show an $O(n \log n)$ pruning algorithm, which only reads the first n tuples, assuming the *expected* number of tuples in $\mathcal{D}$ is known to the algorithm.

6.1 Exact computation

From Definition 8, in particular (2), given tuples that are sorted by their score attribute, we have:

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

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,

$$\begin{aligned} r(t_i) =& p(t_i) \cdot \sum_{t_j \bar{\diamond} t_i, j<i} p(t_j) \\ & + \sum_{t_j \diamond t_i} p(t_j) + (1 - p(t_i)) \cdot \sum_{t_j \bar{\diamond} t_i} p(t_j). \end{aligned} \tag{7}$$

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 (7) can be rewritten as:

$$\begin{aligned} r(t_i) =& p(t_i) \cdot (q_i - \sum_{t_j \diamond t_i, j<i} p(t_j)) + \sum_{t_j \diamond t_i} p(t_j) \\ & + (1 - p(t_i))(\mathrm{E}[|W|] - p(t_i) - \sum_{t_j \diamond t_i} p(t_j)). \end{aligned} \tag{8}$$

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 (taking time $O(N)$, since each tuple appears in exactly one rule). When $\mathcal{D}$ is not presorted by t_i's score attribute, the running time of this algorithm is dominated by the sorting step, $O(N \log N)$. The pseudo-code for T-ERank is presented in Appendix C.

6.2 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 (8). 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 (*Proof in Appendix B*):

$$r(t_\ell) \quad \geq \quad q_\ell - 1 \geq q_n - 1. \tag{9}$$

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 (8). 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.

7 Median and Quantile 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 (or equivalently, the expectation) is statistically sensitive to the distribution of the underlying values (in our case, the absolute ranks of the tuple from all possible worlds), especially when there are outliers in the distribution. It is natural to consider alternate statistical operators as the basis of the ranking, such as the median of the rank distribution instead of the mean. This can be further generalized to any quantile for the distribution of absolute ranks for a tuple across all possible worlds. We can then derive the final ranking based on such quantiles. Furthermore, the rank distribution $\mathrm{rank}(t_i)$ for a tuple t_i reflects important characteristics of t_i's rank in any random possible world. Studying these critical statistics (median and general quantiles) for this rank distribution is of independent interest. This section formalizes these ideas and presents efficient algorithms to compute the median and quantile ranks for uncertain databases in both the attribute-level and tuple-level uncertainty models. Similarly to the approach taken for the expected rank, we first present the definitions for these ranks and discuss the efficient, polynomial-time algorithms that compute such ranks; then we improve the efficiency of our algorithms by designing necessary pruning techniques.

7.1 Definitions and Properties

We formally define the median and quantile rank as,

Definition 9: For tuple $t_i \in \mathcal{D}$, its median rank $r_m(t_i)$ is the median value from t_i's rank distribution $\mathrm{rank}(t_i)$, i.e., it is the value in the cumulative distributive function (cdf) of $\mathrm{rank}(t_i)$, denoted as $\mathrm{cdf}(\mathrm{rank}(t_i))$, that has a cumulative probability of 0.5; For any user-defined ϕ-quantile where $\phi \in (0, 1)$, the ϕ-quantile rank of t_i is the value in the $\mathrm{cdf}(\mathrm{rank}(t_i))$ that has a cumulative probability of ϕ, denoted as $r_\phi(t_i)$. ■

For the attribute-level uncertainty model example in Figure 2, t_1's rank distribution $\mathrm{rank}(t_1)$ is $\{(0, 0.4), (1, 0), (2, 0.6)\}$. Therefore, t_1's median rank $r_m(t_1)$ is 2. Similarly, $r_m(t_2) = 1$ and $r_m(t_3) = 1$. Hence, the final ranking is (t_2, t_3, t_1), which is identical to the final ranking obtained from the expected ranks. For the tuple-level uncertainty model example in Figure 4, t_4's rank distribution $\mathrm{rank}(t_4)$ is $\{(0, 0), (1, 0.3), (2, 0.5), (3, 0.2)\}$. t_4's median rank $r_m(t_4)$ is 2. Similarly, $r_m(t_1) = 2$, $r_m(t_2) = 1$, and $r_m(t_3) = 1$. The final ranking is (t_2, t_3, t_1, t_4) whereas the final ranking from the expected ranks was (t_3, t_1, t_2, t_4).

Clearly, the median rank is a special case of ϕ-quantile rank where $\phi = 0.5$. Ranking by the median or the ϕ-quantile ranks of all tuples is straightforward. We first derive a total ordering of all tuples in the *ascending* order of their $r_m(t_i)$'s (or $r_\phi(t_i)$'s); and the k tuples with the smallest values of $r_m(t_i)$'s (or $r_\phi(t_i)$'s) are returned as the top-k. We can show that ranking by the median rank (or the general quantile rank) offers all properties satisfied by the expected rank. Formally,

Theorem 2: Ranking by median and quantile ranks satisfies all properties in Figure 5. The proof of is quite similar to the proof of Theorem 1, so we omit it.

The next important problem is whether we can calculate $r_m(t_i)$ or $r_\phi(t_i)$ efficiently for a tuple t_i. Note, besides for ranking purposes, these values themselves are important statistics to characterize the rank distribution of t_i in all possible worlds, $\mathrm{rank}(t_i)$. In the sequel, in a random possible world if t_i and t_j are tied ranking by their scores, t_i ranks before t_j if $i < j$. In general, other tie-breaking mechanisms could be easily substituted. Also, our algorithms work the same way for median and quantile ranks for any quantile values. Hence, for brevity, we focus on discussing median ranks and extend it to quantiles ranks at the end of each case. Recall that for any tuple t, $R(t)$ is a random variable that denotes t's rank in a random possible world. $R(t)$'s distribution is represented by t's rank distribution $\mathrm{rank}(t)$.

7.2 Attribute-Level Uncertainty Model

In the attribute-level uncertainty model, given a tuple t_i, its uncertain score attribute is represented by the random variable $X_i = \{(v_{i,1}, p_{i,1}), \ldots, (v_{i,s_i}, p_{i,s_i})\}$ for some constant s_i. When X_i takes the value $v_{i,1}$ (denote this as a special tuple t_i^1), tuple t_i's rank in all possible worlds could be described by a probability distribution $\mathrm{rank}(t_i^1) = \mathrm{rank}(t_i | t_i = v_{i,1})$ where '|' means "given that". We concentrate on calculating $\mathrm{rank}(t_i^1)$ first. Note that $\mathrm{rank}(t_i^1) \equiv \Pr[R(t_i^1) = \ell]$ for $\ell = 0, \ldots, N-1$ (that is, the distribution gives the probability of the random variable $R(t_i^l)$ taking on each of the N possible rank values). For any other tuple $t_j \in \mathcal{D} \wedge j \neq i$, we calculate $\Pr[t_j > t_i^1] = \sum_{\ell=1}^{s_j} p_{j,\ell} | v_{j,\ell} > v_{i,1}$, and $\Pr[t_j < t_i^1]$, $\Pr[t_j = t_i^1]$ similarly. Then when $j < i$,

$$\Pr[R(t_j) < R(t_i^1)] = \Pr[t_j > t_i^1] + \Pr[t_j = t_i^1] \text{ and}$$
$$\Pr[R(t_j) > R(t_i^1)] = \Pr[t_j < t_i^1]$$

and when $j > i$,

$$\Pr[R(t_j) < R(t_i^1)] = \Pr[t_j > t_i^1] \text{ and}$$

$$\Pr[R(t_j) > R(t_i^1)] = \Pr[t_j < t_i^1] + \Pr[t_j = t_i^1].$$

In short, for any t_j, we can calculate both $\Pr[R(t_j) < R(t_i^1)]$, denoted as $p_j^{\uparrow}$, and $\Pr[R(t_j) > R(t_i^1)]$, denoted as $p_j^{\downarrow}$, efficiently. Now, for each tuple t_j there are two possible outcomes, either it ranks higher than t_i^1 with probability $p_j^{\uparrow}$ or it ranks lower than t_i^1 with probability $p_j^{\downarrow}$. There are $(N-1)$ such independent events (one for each t_j for $j \in \{1, \ldots, N\} - \{i\}$). This could be viewed as a *generalized binomial distribution*. In the jth trial, the head probability is $p_j^{\uparrow}$ and the tail probability is $p_j^{\downarrow}$. The possible ranks of t_i^1 (essentially it is $(t_i | t_i = v_{i,1})$) are simply the possible number of heads from this distribution. Then, $\mathrm{rank}(t_i^1)$ is the probability distribution on the number of heads in this generalized binomial distribution, i.e., $\mathrm{rank}(t_i^1) = \Pr[\text{number of heads} = \ell]$ for $\ell = 1, \ldots, N-1$. For the binomial distribution, there is a compact closed-form formula to calculate $\Pr[\text{number of heads} = \ell]$ for any ℓ. This no longer holds for the generalized binomial distribution. However, one can efficiently compute $\Pr[\text{number of heads} = \ell]$ for all ℓ in this case as follows. Let $E_{\gamma,j}$ be the probability that in the first j trials there are γ number of heads. Then,

$$E_{\gamma,j} = E_{\gamma-1,j-1} \times p_j^{\uparrow} + E_{\gamma,j-1} \times p_j^{\downarrow} \tag{10}$$

From equation 10, the rank distribution $\mathrm{rank}(t_i^1)$ is defined by $\Pr[\text{number of heads} = \ell]$ for $\ell \in \{0, 1, \ldots, N-1\}$, which is given by $E_{\ell,N-1}$'s for $\ell \in \{0, 1, \ldots, N-1\}$. This observation and Equation 10 immediately give us a dynamic programming formulation to calculate the rank distribution $\mathrm{rank}(t_i^1)$.

We can carry out the similar procedure to obtain the distributions $\mathrm{rank}(t_i^1), \ldots, \mathrm{rank}(t_i^{s_i})$, one for each choice of t_i. Finally, $\mathrm{rank}(t_i) \equiv \Pr[R(t_i) = \ell]$ for $\ell \in \{0, 1, \ldots, N-1\}$, where $\Pr[R(t_i) = \ell] = \sum_{\kappa=1}^{s_i} p_{i,\kappa} \times \Pr[R(t_i^{\kappa}) = \ell]$ and $\Pr[R(t_i^{\kappa}) = \ell]$ is given by $\mathrm{rank}(t_i^{\kappa})$, i.e, the rank distribution of t_i, $\mathrm{rank}(t_i)$, is simply the weighted sum of the distributions $\mathrm{rank}(t_i^1), \ldots, \mathrm{rank}(t_i^{s_i})$. With $\mathrm{rank}(t_i)$, one can easily get t_i's median rank $r_m(t_i)$ or any ϕ-quantile rank $r_\phi(t_i)$. Given a query parameter k, the final step is to simply retrieve the k tuples with the smallest median (or quantile) rank values. We denote this algorithm as *A-MQRank*.

Example 1: For the attribute-level uncertainty model example in Figure 2, t_2^1's rank distribution $\mathrm{rank}(t_2^1) = \mathrm{rank}(t_2 | t_2 = 92)$ is $\{(0, 0.6), (1, 0.4), (2, 0)\}$ and t_2^2's rank distribution $\mathrm{rank}(t_2^2) = \mathrm{rank}(t_2 | t_2 = 80)$ is $\{(0, 0), (1, 0.6), (2, 0.4)\}$. Therefore, t_2's rank distribution $\mathrm{rank}(t_2)$ is $\{(0, 0.36), (1, 0.48), (2, 0.16)\}$. □

We observe that the same principle could be applied for the case of continuous distributions. Essentially, for a tuple t_i, we need to compute $p_j^{\uparrow} = \Pr[X_j > X_i]$ and $p_j^{\downarrow} = \Pr[X_j < X_i]$ for any other tuple $t_j \in \mathcal{D}$, where X_j and X_i are two random variables with continuous distributions. Once $p_j^{\uparrow}$ and $p_j^{\downarrow}$ are available, the rest is the same as discussed above.

The Complexity of *A-MQRank*. For an uncertain database $\mathcal{D}$ with N tuples and assuming that each tuple takes on at most s possible choices, the cost of one dynamic program for one choice of a tuple (i.e. applying (10)) is $O(N^2)$. We have to do this for each choice of every tuple. Hence, the cost of the algorithm *A-MQRank* is $O(sN^3)$. When s is a constant, the complexity of *A-MQRank* is $O(N^3)$.

Pruning Techniques. In practice, the number of choices of each tuple could be large. The contribution by the factor of s in the overall computation cost for the algorithm $O(sN^3)$ may be non-negligible. To remedy this problem, an important observation is that if we rank the score values of all choices for a tuple in decreasing order, then the median (or any quantile) rank value for a tuple after seeing more choices will only increase. This intuition gives us a way to lower-bound the median (or any quantile) rank value in any intermediate steps after seeing any number of choices for a tuple. Specifically, after calculating the rank distributions $\mathrm{rank}(t_i^{\ell})$'s for the first x number of choices for a tuple t_i, we denote the lower-bound on $r_m(t_i)$ as $r_m^x(t_i)$ (or $r_\phi^x(t_i)$ for a quantile rank $r_\phi(t_i)$ with a quantile value ϕ). We would like to maintain $r_m^x(t_i)$ such that (a) for $\forall \ell_1, \ell_2 \in [1, s_i]$, if $\ell_1 < \ell_2$, then $r_m^{\ell_1}(t_i) \leq r_m^{\ell_2}(t_i)$; and (b) $r_m^{s_i}(t_i) = r_m(t_i)$. The same holds for the quantile ranks.

Assume we can achieve the above, an immediate pruning technique is to maintain a priority queue of size k for the first ℓ ($\ell \in [1, N)$) tuples whose exact median ranks (or quantile ranks) have already been calculated. The priority queue is ordered by increasing order of the tuple's exact median rank (or quantile rank) and we denote the kth tuple's rank as r_k. When processing the $(\ell+1)$-th tuple, after calculating $\mathrm{rank}(t_{\ell+1}^i)$'s for $i = 1, \ldots, x$-th choices of $t_{\ell+1}$, if $r_m^x(t_i) \geq r_k$ (or $r_\phi^x(t_i) \geq r_k$), we can stop processing the remaining choices of $t_{\ell+1}$ and safely claim $t_{\ell+1}$ has no chance to be in the final top-k answer. If one has to exhaust all choices for $t_{\ell+1}$, then the exact rank $r_m(t_{\ell+1})$ (or $r_\phi(t_{\ell+1})$) is obtained and the priority queue is updated if necessary: if $t_{\ell+1}$'s rank is smaller than r_k, it will be inserted into the queue with its rank and the last (the kth) element of the queue will be deleted.

The remaining challenge is how to calculate the lower-bound $r_m^x(t_i)$ for a tuple t_i after processing its first x choices. A key observation is each tuple's choices are sorted by descending order of their score values, i.e., for $\forall t_i$ and $\forall \ell_1, \ell_2 \in [1, s_i]$, if $\ell_1 < \ell_2$, then $v_{i,\ell_1} > v_{i,\ell_2}$. After processing the first x ($x < s_i$) choices of t_i, we have obtained x rank distributions, $\mathrm{rank}(t_i^1), \ldots, \mathrm{rank}(t_i^x)$. At this point, we can construct a (notional) tuple $t_{i,x}$ with $(x+1)$ choices as follows: $\{(v_{i,1}, p_{i,1}), \ldots, (v_{i,x}, p_{i,x}), (v_{i,x}, 1 - \sum_{\ell=1}^{x} p_{i,\ell})\}$. The first x choices of $t_{i,x}$ are identical to the first x choices of t_i; and the last choice of $t_{i,x}$ has the same score value, $v_{i,x}$ with probability as $1 - \sum_{\ell=1}^{x} p_{i,\ell}$ — we aggregate the probability of all remaining choices (of t_i) into one choice and make its score value the same as the last processed choice (the xth one) from t_i. We can write the rank distribution for constructed tuple $t_{i,x}$ as follows, which follows immediately from its construction:

Lemma 1: We have $\forall \ell \in [1, x]$, $\mathrm{rank}(t_{i,x}^{\ell}) = \mathrm{rank}(t_i^{\ell})$; $\mathrm{rank}(t_{i,x}^{x+1}) = \mathrm{rank}(t_{i,x}^{x})$; and $\mathrm{rank}(t_{i,x}) = \sum_{\ell=1}^{x} p_{i,\ell} \times \mathrm{rank}(t_i^{\ell}) + (1 - \sum_{\ell=1}^{x} p_{i,\ell}) \times \mathrm{rank}(t_i^{x})$.

The next result is the median (quantile) value from distribution $\mathrm{rank}(t_{i,x})$ will not be larger than the median (or corresponding quantile) value in distribution $\mathrm{rank}(t_i)$, i.e.,

Lemma 2: $\forall x \in [1, s_i)$ and $\forall \phi \in [0, 1]$, $r_\phi(t_{i,x}) \leq r_\phi(t_i)$. A special case is that $r_m(t_{i,x}) \leq r_m(t_i)$.

This follows immediately from Lemma 1 and the definition of $r_\phi(t)$ and $r_m(t)$ for any tuple t. Lemma 2 indicates that by constructing $t_{i,x}$ after processing the first x choices of the

tuple t_i, we can efficiently obtain a lower bound on $r_\phi(t_i)$ or $r_m(t_i)$. This can be done after processing each choice of t_i and it perfectly satisfies our pruning framework. Note Lemma 1 shows that after processing the first x choices of the tuple t_i, $\mathrm{rank}(t_{i,x})$ can be obtained immediately (and hence its $r_\phi(t_{i,x})$ or $r_m(t_{i,x})$ is immediate). We denote this pruning technique as the *A-MQRank-Prune* algorithm.

7.3 Tuple-level Uncertainty Model

In the tuple-level model, we have additional challenges due to the presence of exclusion rules. We leverage the dynamic programming formulation that is similar in spirit to some of the existing work in the tuple-level model [46], [5].

Our approach begins by sorting all tuples by the *descending* order of their score values. Without loss of generality, we assume that for any $t_i, t_j \in \mathcal{D}, i \neq j$, if $i < j$, then t_i's score value is greater or equal than t_j's score value. Ties are broken arbitrarily or can be otherwise specified.

Let $\mathcal{D}_i$ be the database when $\mathcal{D}$ is *restricted* (both the tuples and the rules) on the first i tuples $\{t_1, \ldots, t_i\}$ for $i = 1, \ldots, N$, i.e., for each $\tau \in \mathcal{D}$, $\tau' = \tau \bigcap \{t_1, \ldots, t_i\}$ is included in $\mathcal{D}_i$. We first discuss the simplified tuple-level model where each rule contains just one tuple, i.e., every tuple is independent from all other tuples in the database. In this case, our idea is 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 tuples from the first $i-1$ tuples appear, no matter *which* tuples appear. Now, let $R_{i,j}$ be the probability that a randomly generated world from $\mathcal{D}_i$ has exactly j tuples, i.e., $R_{i,j} = \sum_{|W|=j} \Pr[W|\mathcal{D}_i]$, and $R_{0,0} = 1$, $R_{i,j} = 0$ if $j > i$. Also, let $R^{-i}_{N-1,j}$ be the probability that a randomly generated world from $\mathcal{D} - \{t_i\}$ has exactly j tuples. Then, based on Definition 6, it is clear that the probability that t_i's rank equals to j in a randomly generated world from $\mathcal{D}$ is:

$$\Pr[R(t_i) = j] = p(t_i) \cdot R_{i-1,j} + (1 - p(t_i)) \cdot R^{-i}_{N-1,j}, \quad (11)$$

recalling $R(t_i)$ is a random variable denoting the rank for t_i in a random possible world. The final rank distribution $\mathrm{rank}(t_i)$ is simply represented by the pairs, $(j, \Pr[R(t_i) = j])$, for $j = 0, \ldots, N-1$. For any tuple t_i, it is straightforward to calculate $R^{-i}_{N-1,j}$ for all j's in a similar fashion as we compute $R_{i,j}$'s. Our job is then to compute $R_{i,j}$'s, which in this case is:

$$R_{i,j} = p(t_i) R_{i-1,j-1} + (1 - p(t_i)) R_{i-1,j}. \quad (12)$$

This gives us a dynamic programming formulation to calculate the $R_{i,j}$'s. We can then effectively compute the rank distribution $\mathrm{rank}(t_i)$ for the tuple t_i. Consequently, both the median rank and the ϕ-quantile rank for the tuple t_i can be easily obtained. We do this for each tuple in the database and return the k tuples with the smallest median ranks or the ϕ-quantile ranks as the answer to the top-k query.

The general case when there are multiple tuples in one rule is more complex. Nevertheless, in this case, the probability that t_i's rank is equal to j in a randomly generated world from $\mathcal{D}$ still follows the principle outlined in (11). However, $R_{i,j}$ can no longer be calculated as in (12), because if t_i has some preceding tuples from the same rule, the event that t_i appears is no longer independent of the event exactly $j-1$ tuples in $\mathcal{D}_{i-1}$ appear. Similarly, for the second term in (11), when t_i does not appear in a random world with j tuples, we may not simply multiply $(1 - p(t_i))$ by the probability of this event: it could be one tuple from the same rule containing t_i has already appeared, which asserts t_i cannot appear at all.

To overcome this difficulty, we first convert $\mathcal{D}_i$ to a database $\bar{\mathcal{D}}_i$ where all rules contain only one tuple and apply the above algorithm on $\bar{\mathcal{D}}_i$ to compute $R_{i,j}$'s. We construct $\bar{\mathcal{D}}_i$ as follows: For each rule $\tau \in \mathcal{D}_i$ and tuples in τ, we create one tuple $\overline{t}$ and one rule $\overline{\tau} = \{\overline{t}\} \in \bar{\mathcal{D}}_i$, where $p(\overline{t}) = \sum_{t \in \tau} p(t)$, with all of $\overline{t}$'s other attributes set to null. Essentially, $\overline{t}$ represents all tuples in a rule $\tau \in \mathcal{D}_i$. Now the $R_{i,j}$ computed from $\bar{\mathcal{D}}_i$ is the same as the probability that exactly j tuples in $\mathcal{D}_i$ appear, because for $R_{i,j}$ we only care about the number of tuples appearing, merging does not affect anything since the probability that $\overline{t}$ appears is the same as the probability that one of the tuples in τ appears.

Now, we can compute all the $R_{i,j}$'s, but another difficulty is the probability t_i's rank is equal to j if it appears is no longer simply $p(t_i) \cdot R_{i-1,j}$. This happens if t_i has some preceding tuples from the same rule in $\mathcal{D}_{i-1}$. Then the existence of t_i has to exclude all these tuples, while $R_{i-1,j}$ includes the probability of the possible worlds that contain one of them. To handle this case, one can define $\mathcal{D}^-_{i-1} = \mathcal{D}_{i-1} - \{t | t \in \mathcal{D}_{i-1}$ and $t \in \tau$ and $t_i \in \tau\}$, i.e., $\mathcal{D}^-_{i-1}$ is the version of $\mathcal{D}_{i-1}$ that excludes all tuples from the same rule τ which contains t_i . Just as $R_{i-1,j}$ is defined with respect to $\mathcal{D}_{i-1}$, let $R^-_{i-1,j}$ be the probability exactly j tuples from $\mathcal{D}^-_{i-1}$ have appeared in a random possible world. We construct $\bar{\mathcal{D}}^-_{i-1}$ as follows: For each rule $\tau \in \mathcal{D}^-_{i-1}$ and tuples in τ, we create one tuple $\overline{t}$ and one rule $\overline{\tau} = \{\overline{t}\} \in \bar{\mathcal{D}}^-_{i-1}$, where $p(\overline{t}) = \sum_{t \in \tau} p(t)$, with all of $\overline{t}$'s other attributes set to null. Now, computing $R^-_{i,j}$ from $\bar{\mathcal{D}}^-_{i-1}$ is done in the same fashion as (12).

For the second term in (11), to cater for the case when a random world generated from database $\mathcal{D} - \{t_i\}$ has exactly j tuples and t_i does not appear, there are two cases. In case one, none of the j tuples in this random world is from the same rule that contains t_i, we denote this event's probability as $R^{-i-}_{N-1,j}$; In the second case, one of the j tuples in this random world comes from the same rule which contains t_i, and this event's probability is $R^{-i+}_{N-1,j}$. We can compute both $R^{-i-}_{N-1,j}$ and $R^{-i+}_{N-1,j}$ similarly as we compute the $R_{i,j}$'s.

The probability t_i's rank in a random world equals j is:

$$\Pr[R(t_i) = j] = p(t_i) \cdot R^-_{i-1,j} + (1 - p(t_i)) \cdot R^{-i-}_{N-1,j} + R^{-i+}_{N-1,j}, \quad (13)$$

since $R^-_{i-1,j}$ already excludes all tuples from the same rule containing t_i. We calculate this probability for all j's where $j = 0, \ldots, N-1$; then $(j, \Pr[R(t_i) = j])$'s for $j = 0, \ldots, N-1$ is the rank distribution $\mathrm{rank}(t_i)$ for tuple t_i. Both the median rank and the ϕ-quantile rank can be easily obtained thereafter. The top-k answer could be easily obtained after calculating the median (or the quantile) rank for each tuple. We denote this algorithm as *T-MQRank*.

Example 2: We consider tuple t_4 for the tuple-level uncertainty model example in Figure 4. For t_4 the $R^-_{i-1,j}$'s, where $i = 4$ and $j = 0, 1, 2, 3$ in this

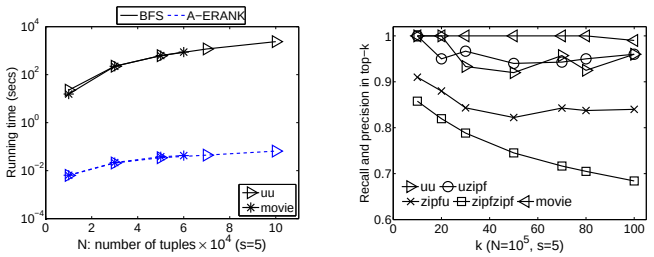
case, are $\{(0,0),(1,0.6),(2,0.4),(3,0)\}$, the $R^{-i+}_{N-1,j}$'s are $\{(0,0),(1,0),(2,0.3),(3,0.2)\}$, and the $R^{-i-}_{N-1,j}$'s are all zero as the probability that neither t_2 or t_4 appear is zero. Then, $\mathrm{rank}(t_4)$ is $\{(0,0),(1,0.3),(2,0.5),(3,0.2)\}$. □

Complexity of *T-MQRank*. In the basic case where each rule contains only one tuple, the cost of the dynamic program to compute the $R_{i,j}$'s is $O(N^2)$, and one can compute $R^{-i}_{N-1,j}$'s for all t_i's similarly in $O(N^2)$ time. After which, for each t_i, obtaining the entire distribution $\mathrm{rank}(t_i)$ given $R_{i,j}$'s and $R^{-i}_{N-1,j}$ is linear. Hence, the overall cost is $O(N^2)$.

In the second case when each rule may contain multiple tuples, we have to invoke the dynamic programming formulation for $R^{-}_{i-1,j}$ based on $\bar{D}^{-}_{i-1}$, which is different for each t_i. However, the size of the table for the dynamic programming formulation is only $O(M^2)$, where $M \leq N$ is the number of rules in the database. The cost of calculating $R^{-i-}_{N-1,j}$'s and $R^{-i+}_{N-1,j}$'s is also $O(M^2)$ for every tuple t_i. Hence, the overall cost of this algorithm is $O(NM^2)$.

8 Experiments

We implemented our algorithms in GNU C++. All experiments were executed on a Linux machine with a 2GHz CPU and 2GB main memory. We utilized synthetic data sets to study the impact of data sets with different characteristics on both the score and the probability distribution. We additionally tested our algorithms on a real data set, *movie*, from the MystiQ project. *movie* consists of data integrated from IMDB and Amazon. Each record in *movie* consists of an IMDB id (*imdbid*), IMDB string, Amazon id (*aid*), Amazon string, and a probability, where the pair (*imdbid,aid*) is unique. We converted *movie* to the attribute-level and tuple-level models by grouping records by the *imdbid* attribute. To obtain an attribute-level representation we created a single tuple for each *imdbid* group where the pdf consists of all of the *aid* in the group. To obtain a tuple-level representation we created a tuple for each *aid* in an *imdbid* group and we also created an exclusion rule (τ) for each *imdbid* group consisting of all *aid* in the group. For both models we rank tuples by the *aid* attribute. The *movie* data set consists of 246,816 unique (*imdbid, aid*) pairs and there are 56,769 unique *imdbid* groups with an average of 4.35 *aid* per group. For the experiments we vary only N and k for *movie*, where N is varied by uniformly and randomly selecting tuples. To generate synthetic data sets, we developed several data generators for both 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 $\mathcal{D}$. 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, i.e. *zipfu* indicates a Zipfian distribution of scores and uniform distribution of probabilities. The default skewness for the Zipfian distribution is 1.2, and other default values are $k = 100$, $N = 10,000$, $s = 5$, $\psi = 5$, and $\zeta = 30\%$.



(a) Running time of exact algorithms (b) A-ERank-Prune's Precision/Recall.

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

8.1 Expected Ranks

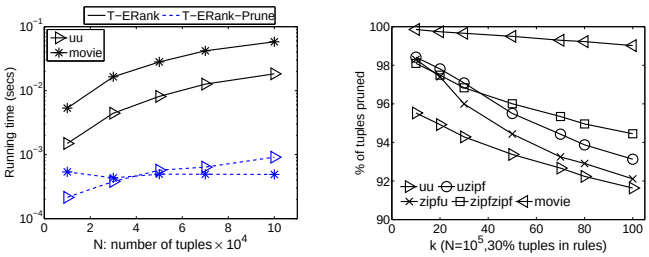
8.1.1 Attribute-level Uncertainty Model

We first studied the performance of the exact algorithm A-ERank by comparing it to the basic brute-force search (BFS) approach on uu and $movie$. The distribution on the probability universe does not affect the performance of either algorithm, 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, amongst the synthetic data sets we only consider uu for this experiment, to give the toughest test for this algorithm.

Figure 6(a) shows the total running time of these algorithms as the size of $\mathcal{D}$ (the number of tuples, N) is varied, up to $100,000$ tuples. Note we can only vary $movie$ up to $N = 56,769$ for this and the following attribute-level experiments. 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 the expected ranks of all tuples for $N = 100,000$, while the brute force approach takes ten minutes. Results are similar for other values of s.

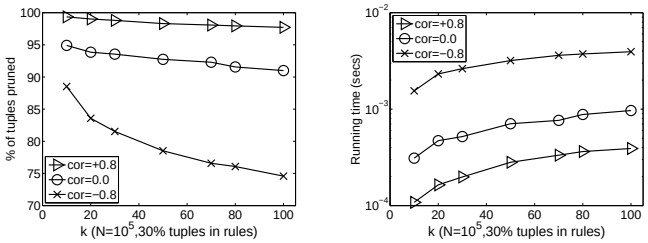
As discussed in Section 5.2, A-ERank-Prune is an approximate algorithm, in that it may not find the exact top-k. Figure 6(b) reports its approximation quality on various data sets using the standard *precision* and *recall* metrics. Since A-ERank-Prune always returns k tuples, its recall and precision are always the same. Figure 6(b) shows it achieves high approximation quality: recall and precision are near the 100th percentile for $movie$ and in the 90th percentile when the score is distributed uniformly for the synthetic data sets. 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%.

We also evaluate the pruning power of A-ERank-Prune in Appendix D.1.1. The results show A-ERank-Prune is more ef-



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

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



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

Fig. 8. Impact of different correlations on T-ERank-Prune.

ficient for skewed distributions. However, there is still enough information from expected scores to prune even for more uniform distributions.

8.1.2 Tuple-level Uncertainty Model

For our experiments in the tuple-level model, we first investigate the performance of our algorithms. As before, there is a brute-force search based approach which is much more expensive than our algorithms, so we do not show these results.

A notable difference in this model is the pruning algorithm is able to output the exact top-k, provided $\mathrm{E}[|W|]$, 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 on *uu* and $movie$. Both algorithms are extremely efficient. For $100,000$ tuples, the T-ERank algorithm takes less than 100 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 k. T-ERank-Prune prunes more than 98% of tuples for all k for $movie$ which shows T-ERank-Prune may be very effective at pruning real data for certain applications. We also see a skewed distribution on either dimension increases the pruning capability of T-ERank-Prune. Even in the worst case of processing *uu*, T-ERank-Prune is able to prune more than 90% of tuples.

Our next experiments study the impact of correlations between a tuple's score value and probability. We say the two are positively correlated when a tuple with a higher score value also has a higher probability; a negative correlation means a tuple with a higher score value has a 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 the 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 $100,000$ tuples.

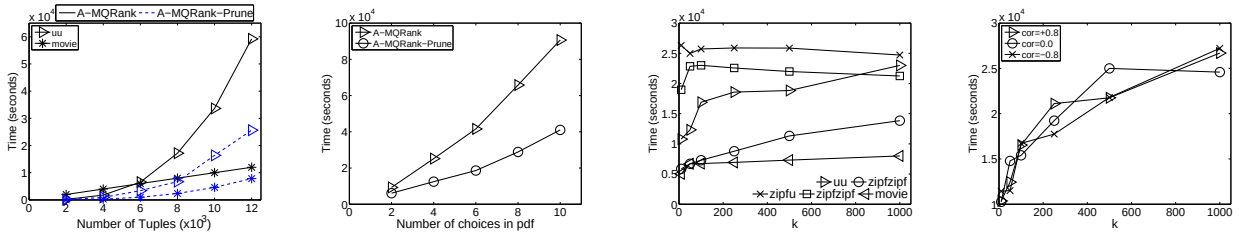
8.2 Median and Quantile Ranks

Our primary concern when evaluating A-MQRank, A-MQRank-Prune, and T-MQRank was the time necessary to retrieve the top-k from an uncertain database. To evaluate the query time for A-MQRank and T-MQRank we utilize uu and $movie$. The results for other data sets are similar as the computational effort imposed by A-MQRank and T-MQRank are the same regardless of the distribution as both algorithms compute the median or quantile rank for every tuple in the database. We utilize $movie$, uu, $uzipf$, $zipfzipf$, and $zipfu$ to show the effect different distributions have on the pruning power of A-MQRank-Prune. We also analyze the effect positively correlated and negatively correlated data sets have on the processing time of A-MQRank-Prune. For all experiments we present the results from calculating the median ranks of all tuples, as other general quantiles perform similarly.

8.2.1 Attribute-Level Uncertainty Model

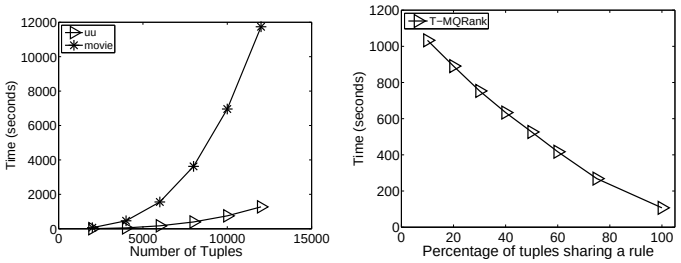
We first analyze the effects of varying the number of tuples N with respect to the processing time for A-MQRank and A-MQRank-Prune in Figure 9(a). Recall A-MQRank and A-MQRank-Prune are both $O(sN^3)$. However, the pruning techniques utilized in A-MQRank-Prune could allow the algorithm to avoid performing the $O(N^2)$ dynamic program for all the s unique entries in the pdf of a tuple. The effects of this pruning are apparent in Figure 9(a). We utilize $movie$ and uu and vary N from 2,000 to 12,000. The processing time for $movie$ is less than that of uu for both algorithms as s is on average 4 in $movie$ whereas in uu $s = 5$ for all tuples. In all cases A-MQRank-Prune requires less processing time. In Figure 9(b) we analyze the effect of varying s for A-MQRank and A-MQRank-Prune. Here we use only uu as we cannot vary s in $movie$. In this experiment, s is varied from 2 to 10. Again we see the pruning of A-MQRank-Prune performs very well effectively reducing the constant in the $O(sN^3)$ complexity of A-MQRank.

We next analyze the effect different distributions have on the pruning power of A-MQRank-Prune. In Figure 9(c) we study the effects of varying k over $movie$, uu, $zipfzipf$, $uzipf$, and $zipfu$ on A-MQRank-Prune. We see in general the processing time grows almost linearly with respect to k. It is apparent the pruning utilized by A-MQRank-Prune works best for scores which are uniformly distributed and probabilities which follow a zipfian distribution. Regardless of the distribution A-MQRank-Prune shows excellent scalability with respect to k. In Figure 9(d) we analyze the effect of



(a) Effect of N. (b) Effect of s. (c) Effect of k: vary distributions for A-MQRank-Prune. (d) Effect of k: vary correlated distributions for A-MQRank-Prune.

Fig. 9. Attribute-level model: median and quantile ranks performance analysis.



(a) Effect of N. (b) Effect of ζ.

Fig. 10. Tuple-level model: median and quantile ranks performance analysis.

varying k over the positively and negatively correlated data sets. In general we see correlations of the data set do not affect the pruning of A-MQRank-Prune.

8.2.2 Tuple-Level Uncertainty Model

We evaluate the effect of varying the number of tuples N in the database and the percentage of tuples from the database which share a rule with another tuple ζ in Figure 10. Note the value selected for k is irrelevant as the quantile rank for every tuple is computed. In Figure 10(a) we see the amount of time required to determine the median ranks increases quadratically with respect to the number of exclusion rules M in the database, since M clearly increases as N increases. This quadratic relationship makes sense as the running time for T-MQRank is $O(NM^2)$. Also, note $movie$ has about 3 times as many rules as uu which explains why it is about 9 times more expensive. We also evaluate the effect of varying the percentage of tuples which share a rule with another tuple in Figure 10(b). As expected, the amount of time required to calculate the median ranks drops quadratically as the percentage increases. This is clear since as the percentage of tuples which share a rule increases, there are fewer tuples in rules by themselves, and therefore M decreases. We also analyze the effect of varying the maximum number of tuples in Appendix D.1.2. In general, as ψ increases while $\zeta = 30\%$ the amount of time necessary to compute the median ranks decreases.

8.3 Comparison of Expected and Median Ranks

We also studied the similarity of top-k lists for the expected and median ranks on different data sets as k varies. We extend the *averaging Kendall distance* from [16] and define the *normalized averaging Kendall distance* which we use in our comparisons. In general, our results show the top-k lists for both median and expected ranks are rather different which shows median and expected ranks enable us to emphasize different characteristics of a rank distribution. We also study the similarity of the top-k lists of the median ranks and different quantile ranks as k varies. We observe this similarity is very stable, with quantiles closer to the median having more similar top-k lists. The complete details of our results appear in Appendix D.2.

9 Conclusion

We have studied semantics of ranking queries in probabilistic data. We adapt important properties that guide the definition of ranking queries in deterministic databases and analyze characteristics of existing top-k ranking queries for probabilistic data. These properties naturally lead to the ranking approach that is based on the rank distribution for a tuple across all possible worlds in an uncertain domain. Efficient algorithms for two major uncertainty models ensure the practicality of our approach. Our experiments demonstrate that ranking by expected ranks, median ranks and quantile ranks is efficient in both attribute-level and tuple-level uncertainty models.

10 Acknowledgment

Jeffrey Jestes and Feifei Li were supported in part by the US National Science Foundation (NSF) Grants IIS-0916488 and IIS-1053979. Ke Yi was supported in part by a DAG and an RPC grant from HKUST.

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] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Zuefle, "Probabilistic frequent itemset mining in uncertain databases," in *KDD*, 2009.

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

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

[8] L. L. Cam, "An approximation theorem for the poisson binomial distribution," *Pacific Journal of Mathematics*, vol. 10, no. 4, pp. 1181–1197, 1960.

[9] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and efficient fuzzy match for online data cleaning," in *SIGMOD*, 2003.
[10] R. Cheng, J. Chen, M. Mokbel, and C.-Y. Chow, "Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data," in *ICDE*, 2008.
[11] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *SIGMOD*, 2003.
[12] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.
[13] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004.
[14] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *WWW Conference*, 2001.
[15] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *PODS*, 2001.
[16] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing top k lists," in *ACM-SIAM Symposium on Discrete Algorithms*, 2003.
[17] A. Fuxman, E. Fazli, and R. J. Miller, "ConQuer: efficient management of inconsistent databases," in *SIGMOD*, 2005.
[18] T. Ge, S. Zdonik, and S. Madden, "Top-k queries on uncertain data: on score distribution and typical answers," in *SIGMOD*, 2009.
[19] A. Halevy, A. Rajaraman, and J. Ordille, "Data integration: the teenage year," in *VLDB*, 2006.
[20] 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.
[21] W. Hoeffding, "On the distribution of the number of successes in independent trials," *Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 713–721, 1956.
[22] M. Hua, J. Pei, W. Zhang, and X. Lin, "Efficiently answering probabilistic threshold top-k queries on uncertain data," in *ICDE*, 2008.
[23] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *SIGMOD*, 2008.
[24] 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.
[25] I. F. Ilyas, G. Beskales, and M. A. Soliman, "Survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, 2008.
[26] 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.
[27] B. Kanagal and A. Deshpande, "Online filtering, smoothing and probabilistic modeling of streaming data," in *ICDE*, 2008.
[28] C. Li, K. C.-C. Chang, I. Ilyas, and S. Song, "RankSQL: Query algebra and optimization for relational top-k queries," in *SIGMOD*, 2005.
[29] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," *PVLDB*, vol. 2, no. 1, pp. 502–513, 2009.
[30] X. Lian and L. Chen, "Probabilistic ranked queries in uncertain databases," in *EDBT*, 2008.
[31] V. Ljosa and A. Singh, "APLA: Indexing arbitrary probability distributions," in *ICDE*, 2007.
[32] V. Ljosa and A. K. Singh, "Top-k spatial joins of probabilistic objects," in *ICDE*, 2008.
[33] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *VLDB*, 2007.
[34] C. Re, N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic databases," in *ICDE*, 2007.
[35] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working models for uncertain data," in *ICDE*, 2006.
[36] P. Sen and A. Deshpande, "Representing and querying correlated tuples in probabilistic databases," in *ICDE*, 2007.
[37] J. G. Shanthikumar and M. Shaked, *Stochastic Orders and Their Applications*. Academic Press, 1994.
[38] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah, "Orion 2.0: native support for uncertain data," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1239–1242.
[39] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah, "The orion uncertain data management system," in *COMAD*, 2008.
[40] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch, "Indexing uncertain categorical data," in *ICDE*, 2007.
[41] M. A. Soliman and I. F. Ilyas, "Ranking with uncertain scores," in *ICDE*, 2009.
[42] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *ICDE*, 2007.
[43] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Probabilistic top-k and ranking-aggregate queries," *TODS*, vol. 33, no. 3, 2008.
[44] 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.
[45] D. Xin, J. Han, and K. C.-C. Chang, "Progressive and selective merge: Computing top-k with ad-hoc ranking functions," in *SIGMOD*, 2007.
[46] 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.
[47] Q. Zhang, F. Li, and K. Yi, "Finding frequent items in probabilistic data," in *SIGMOD*, 2008.
[48] X. Zhang and J. Chomicki, "On the semantics and evaluation of top-k queries in probabilistic databases," in *DBRank*, 2008.

PLACE PHOTO HERE

Jeffrey Jestes received the BS degree in computer science from Florida State University in 2008. He was a PhD student in the Computer Science Department, Florida State University between August 2008 and July 2011. He is a PhD student at the School of Computing, University of Utah, Since August 2011. His research interests include databases and data management, probabilistic data, parallel and distributed query processing, and string similarity.

PLACE PHOTO HERE

Graham Cormode is a Principal Member of Technical Staff in the Database Management Group at AT&T Shannon Laboratories. Previously, he was a researcher at Bell Labs, after postdoctoral study at the DIMACS center in Rutgers University from 2002-2004. His PhD was granted by the University of Warwick in 2002. He works on data stream algorithms, large-scale data mining, and applied algorithms, with applications to databases, networks, and fundamentals of communications and computation.

PLACE PHOTO HERE

Feifei Li received the BS degree in computer engineering from Nanyang Technological University in 2002 and the PhD degree in computer science from Boston University in 2007. He was an assistant professor at the Computer Science Department, Florida State University, between August 2007 and July 2011. Since August 2011 He is an Assistant Professor at the School of Computing, University of Utah. His research interests include data management, data structures, and databases, as well as security issues in data management. He is a member of the IEEE.

PLACE PHOTO HERE

Ke Yi received his B.E. from Tsinghua University and Ph.D. from Duke University, in 2001 and 2006 respectively, both in computer science. After spending one year at AT&T Labs as a researcher, he has been an Assistant Professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology since 2007. His research interests include algorithms, data structures, and databases.

APPENDIX A
OTHER ISSUES

Variance of the rank distribution: connection of expected ranks, median ranks and quantile ranks. Implicit in all our discussions so far is that for any uncertain tuple t, there is a well-defined probability distribution $\mathrm{rank}(t)$ over its possible ranks within the uncertain relation. Within this setting, the expected rank, median rank and quantile rank of t are the expectation, median, and quantile of this distribution respectively. It is certainly meaningful to study other properties of these distributions. In particular, we next discuss the variance of the rank distribution $\mathrm{rank}(t)$, which we denote as $\mathrm{var}(t)$. A first observation is that $\mathrm{var}(t)$ is a good indicator to gauge the difference among $r(t)$ (expected rank), $r_m(t)$ (median rank) and $r_\phi(t)$ (quantile rank) for any quantile value ϕ. Intuitively, a small $\mathrm{var}(t)$ suggests that $r(t)$ and $r_m(t)$ will be similar (in the extreme case, an extremely small $\mathrm{var}(t)$ suggests that even $r_\phi(t)$ for any ϕ value will be similar to $r(t)$). If all tuples in the database have small $\mathrm{var}(t)$'s, then ranking by expected ranks probably is a good choice. On the other hand, if a large number of tuples have large $\mathrm{var}(t)$ values, then ranking by expected ranks does not provide a good reflection on the ranks of various tuples. Rather, ranking by median ranks or some quantile ranks should be used.

It remains to show how to compute $\mathrm{var}(t)$ given an uncertain tuple t. Clearly, this task is trivial if one is provided with the rank distribution $\mathrm{rank}(t)$. Recall that our algorithms for ranking by median ranks or quantile ranks in Section 7, in both uncertain models, work by firstly computing the rank distribution for any given tuple in the database, then deriving the median (or the quantile) from the computed rank distribution. So, as a straightforward extension, these algorithms can easily support the calculation of $\mathrm{var}(t)$ for any tuple t as well. It is an open problem to find more direct ways to compute $\mathrm{var}(t)$, but the existence of correlations between tuples suggests that this may not be more efficient than simply computing the rank distribution.

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 all of our offline algorithms (for expected ranks, median and quantile ranks) 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 for expected ranks 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 [10], [30] essentially adopt U-kRanks semantics to define k-nearest-neighbor queries in spatial databases. We believe that our ranking definition makes a lot of sense in this context, and may have similar benefits over previous definitions of uncertain nearest neighbors.

0.18	
t_2	2
t_1	1
0.22	
t_1	1

0.09	
t_3	3
t_2	2
0.11	
t_3	3

0.09	
t_4	4
t_2	2
0.11	
t_4	4

0.09	
t_5	5
t_2	2
0.11	
t_5	5

Fig. 11. Possible worlds for the example where all previous definitions fail to satisfy the faithfulness.

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 these 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 to 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. It is certainly reasonable to define further properties and analyze when they hold. However, formulating the right properties can be tricky. For example, Zhang and Chomicki [48] defined a property of "faithfulness", 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 basically says that if t_2 "dominates" t_1, then t_2 should always be ranked at least as high as t_1. It was claimed that the Global-Top$k$ definition satisfies this property, however this only holds in the absence of exclusion rules. There are cases with exclusions where all existing definitions fail on faithfulness. Consider the following example:

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 $\tau_1 = \{t_1, t_3, t_4, t_5\}, \tau_2 = \{t_2\}$. The possible worlds of this relation is shown in Figure 11. Here, t_2 "dominates" t_1, but all of the previous definitions (U-Top$k$, U-kRanks, Global-Top$k$, and PT-k) will select t_1 as the top-1. On this example, ranking by expected ranks will rank t_2 as the top-1, hence, satisfying the "faithfulness" requirement. But it is easy to construct other examples where the expected rank will also rank a dominating tuple lower than a dominated tuple. Consider the example shown in Figure 12, the expected rank of t_1 is $0.105 \times 1 + 0.245 \times 1 + 0.455 \times 1 = 0.805$. But the expected rank of t_2 is $0.195 \times 2 + 0.455 \times 1 = 0.845$.

tuples	score	$p(t)$
t_1	1	0.3
t_2	2	0.35
t_3	0.5	0.65

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

0.105

t_2	2
t_1	1

0.195

t_1	1
t_3	0.5

0.245

t_2	2

0.455

t_3	0.5

Fig. 12. A database $\mathcal{D}$ and its possible worlds where the expected rank does not satisfy the faithfulness.

Hence, t_2 ranks after t_1 even though t_2 dominates t_1. Our initial study suggests that "faithfulness" defined this way may not be achievable, and one has to somehow take rules (i.e., correlations) into consideration in order to make it a viable.

Rank of missing tuples. In the tuple-level uncertainty model, we chose to determine the rank of tuples not present in a world W as $|W|$. This is an intuitive choice (the missing tuples were all ranked "equal last"), but other choices are possible. A different approach is to compute the rank of a tuple only over those worlds W where it does appear, and then to scale this by the probability that it appears. This can be understood in terms of conditional probabilities: for the expected rank definition, we compute the rank of a tuple t by summing the probabilities that other tuples appear which score more highly than t. This can be viewed as the probability that each tuple t' outranks t and t appears; dividing this by $p(t)$ gives the conditional probability t' outranks t given that t appears. Accordingly, we can call this the "conditional expected rank" of t_i, $e(t_i)$. Formally, adopting the convention that the tuples are indexed in decreasing order of their score, $e(t_i) = \frac{1}{p(t_i)}(1 + \sum_{j<i, t_j \bar{\diamond} t_i} p(t_j))$, which can be computed in constant time for each t_i after computing prefix sums on the $p(t_i)$'s and on the probabilities associated with each rule. On the example in Figure 4, we obtain $e(t_1) = 1/0.4 = 2.5$, $e(t_2) = (1 + 0.4)/0.5 = 2.8$, $e(t_3) = (1 + 0.4 + 0.5)/1 = 1.9$ and $e(t_4) = (1 + 0.4 + 1)/0.5 = 4.8$. This yields the ranking (t_3, t_1, t_2, t_4), the same as under the expected rank semantics. Likewise, the properties of exact-k, containment, unique-rank and value-invariance follow immediately. Stability also follows easily: increasing the score of a tuple t cannot increase the sum of probabilities of tuples with lower scores, while increasing its probability drives down $1/p(t)$, so either way $e(t)$ cannot increase, ensuring that if it was in the top-k before it will remain so after. We leave further study of this alternate semantics to future work.

Parametrized Ranking Function. In parallel to this work, a new ranking framework for probabilistic data was proposed [29], namely, the parametrized ranking function (PRF). It is interesting to note that PRF adopts a similar basis to that shown in this work for ranking probabilistic data. Essentially, the basis of ranking in PRF is also the rank distributions for different tuples. However, instead of using the expectations, medians or quantiles of these ranking distributions to derive a total ordering of tuples, Li et al. proposed that any parametrized function may be defined over the rank distributions of tuples, i.e., the final rank value of a tuple $t \in \mathcal{D}$, where $|\mathcal{D}| = N$, could be defined as $\sum_{i=0}^{N-1} \omega(t, i) \Pr[R(t) = i]$, where $\{\omega(t, 0), \ldots, \omega(t, N-1)\}$ is a set of $(N-1)$ user-defined parametrized functions. Clearly, the basis for the above ranking definition is $\Pr[R(t) = i]$ for $i = \{0, \ldots, N-1\}$, which is nothing else but the rank distribution of t, $\mathrm{rank}(t)$. Note that the PRF is a framework for ranking, but not a ranking definition by itself. Many ranking definitions are possible to be defined in the PRF framework, for example, it is indeed possible to define the expected rank in this paper under the PRF framework. However, it is not feasible to directly define the median and quantile ranks using the PRF mechanism. Nevertheless, one may extend the PRF framework to support the median and quantile ranks when its ranking definitions are no longer constrained by using only parametrized functions.

Since the ranking basis is the rank distributions for tuples, when $\omega(t, i)$ is independent from t for all $i \in \{0, \ldots, N-1\}$ (which is prevalent as also noted in [29]), all such rankings within the PRF framework necessarily follow value-invariance. Different instantiations of the parametrized function $\omega(t, i)$ in the PRF framework introduces quite different definitions. It is an intriguing and challenging open problem to further study the rich semantics and properties the PRF framework imposes on various ranking definitions extended from it.

Approximate expected ranks. The rank distribution of an uncertain tuple follows the Poisson Binomial distribution. It follows from standard results in statistics [8], [21] the expected rank of a tuple t is approximated by the sum of probabilities of the tuples ranked before t in the tuple-level model. Since the focus of this work is to rank the uncertain tuples based on their *exact* expected ranks (or median or quantile ranks for that purpose), i.e., we are interested at computing the *exact* top-k results in each framework (be it expected ranks, median ranks or quantile ranks), we do not pursue this observation further in this presentation. We leave further study of approximate top-k ranks of uncertain data to future work. We observe the key challenges to address are to design approximation schemes for broader models of uncertain data, and to quantify the quality of the approximation over all possible uncertain relations, or else to describe particular cases of uncertain relations which are guaranteed to be well-approximated. The hope is an approximation which compromises on finding the exact answer can compensate with computational efficiency.

APPENDIX B
PROOFS

Proof of Equation 2

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

Proof of Theorem 1 For simplicity, we assume 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 identifier. The same tie-breaking issues affect the ranking of certain data as well.

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 $t_i^{\uparrow}$ (as in Definition 4), 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 8 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 $\mathrm{rank}_W(t_i)$ 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, $\mathrm{rank}_W(t_i)$ 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). Since $r(t_i)$ is nothing but a weighted average of the ranks in all the possible worlds, where the weight of W is $\Pr[W]$, it is sufficient to consider the changes in the contribution of the possible worlds in categories (a) and (c). Observe that there is a one-to-one mapping between the possible worlds in category (c) and (a): $W \leftrightarrow W \cup \{t_i\}$. For each such pair, its contribution to $r(t_i)$ is

$$\Pr[W] \cdot |W| + \Pr[W \cup \{t_i\}] \cdot \mathrm{rank}_{W\cup\{t_i\}}(t_i). \qquad (14)$$

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 $\Pr[W] = \pi\,(1 - \sum_{\ell} p(t_{i,\ell}) - p(t_i))$ 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 (14) is

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

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

$$\Pr[W] \cdot \mathrm{rank}_W(t_{i'}) + \Pr[W \cup \{t_i\}] \cdot \mathrm{rank}_{W\cup\{t_i\}}(t_{i'}). \quad (15)$$

When $p(t_i)$ increases to $p(t_i^{\uparrow})$, the increase in (15) is

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

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

Proof of Equation 4

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

Proof of Equation 5

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

Proof of Equation 6

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

Proof of Equation 9

$$\begin{aligned} r(t_\ell) =& p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell, j<\ell} p(t_j) \\ &+ \sum_{t_j \diamond t_\ell} p(t_j) + (1 - p(t_\ell)) \cdot \sum_{t_j \bar{\diamond} t_\ell} p(t_j) \quad \text{(from (7))} \\ =& p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell, j<\ell} p(t_j) + \mathrm{E}[|W|] - p(t_\ell) - p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell} p(t_j) \\ =& \mathrm{E}[|W|] - p(t_\ell) - p(t_\ell) \cdot \left(\sum_{t_j \bar{\diamond} t_\ell} p(t_j) - \sum_{t_j \bar{\diamond} t_\ell, j<\ell} p(t_j) \right) \end{aligned}$$

$$= \mathrm{E}[|W|] - p(t_\ell) - p(t_\ell) \cdot \sum_{t_j \bar{\diamond} t_\ell, j>\ell} p(t_j). \quad (16)$$

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 (16), we have:

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

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 $\ell > n$.

APPENDIX C
ALGORITHM PSEUDO CODE

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. (4);
    Insert (t_i, r(t_i)) into A;
    if |A| > k then Drop element with largest expected rank from A;
return A;
```

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 (8);
    if |A| > k then drop element with largest expected rank from A;
return A;
```

APPENDIX D
FURTHER EXPERIMENTS

D.1 Expected Ranks

D.1.1 Expected Ranks in the Attribute-level Model

Figure 13 shows the pruning power of A-ERank-Prune. In this experiment $N = 100,000$, $s = 5$, and k is varied from 10 to 100. It shows we often only need to materialize a small number of tuples of $\mathcal{D}$ (ordered by expected score) before we can be sure 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 results in Figure 13. When both distributions are skewed, A-ERank-Prune could halt the scan after seeing less than 20% of the relation. For the $movie$ data set A-ERank-Prune is also effective and prunes almost 50% of the tuples. Even for more uniform distributions such as uu expected scores hold enough information to prune.

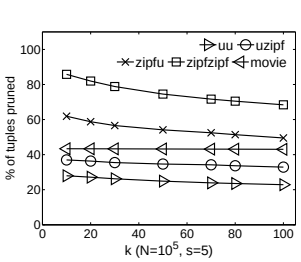


Fig. 13. Attribute-level: Pruning of A-ERank-Prune.

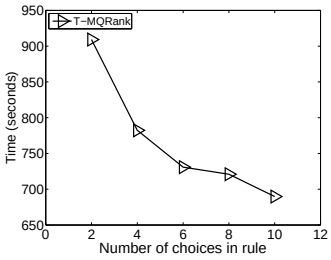


Fig. 14. Tuple-level: Effect of ψ for median and quantile ranks.

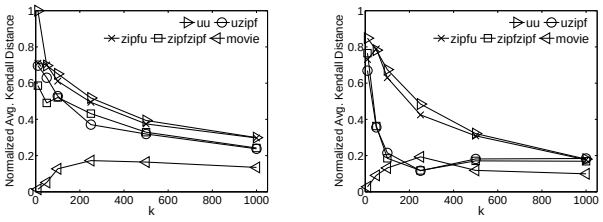
D.1.2 Expected Ranks in the Tuple-Level Model

Figure 14 shows by increasing the number of tuples in a rule, while still requiring $\zeta = 30\%$, also reduces the amount of time to compute median ranks. This is explainable by the fact that increasing the number of tuples in any rule τ will clearly cause the total number of exclusion rules in the database to decrease. It follows that the processing time for T-MQRank will be very dependent on N, ζ, and ψ. It is likely M will only be a constant factor smaller than N for any uncertain database. However, as N increases this reduction factor will become increasingly significant in terms of the computational effort required by T-MQRank. In any case T-MQRank still has a low polynomial time cost with respect to N and M.

D.2 Comparison of Expected and Median Ranks

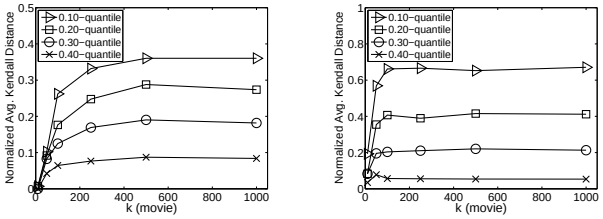
We have shown retrieving the top-k from an uncertain database following either the attribute-level or tuple-level model may be computed efficiently in $O(N \log N)$ time when ranking by expected ranks. We have also presented algorithms to compute median and quantile ranks for a database following the attribute-level and tuple-level model in $O(sN^3)$ and (NM^2) time respectively. It is evident not only from our experiments but also from the corresponding complexities for the expected rank and median and quantile rank algorithms that retrieving the top-k tuples from an uncertain database using expected ranks may require much less computational effort than median or quantile ranks. It is clear from Figure 9 and Figure 10 that determining the top-k from both attribute-level and tuple-level uncertain databases using median ranks requires on the order of 10^4 seconds. In comparison, from Figure 6 and Figure 7 we can see we need less than one second to determine the top-k for both attribute-level and tuple-level uncertain databases utilizing the expected ranks. It is not surprising ranking by median ranks requires more computational effort than ranking by expected ranks since we must compute the rank distribution $\mathrm{rank}(t_i)$ for every $t_i \in \mathcal{D}$ in order to determine the median ranks. To do this we rely on dynamic programs with quadratic complexities. However, it has been commonly observed that calculating the median or quantile values for a distribution is more expensive than computing the expectation of the distribution.

We also compared the similarity between the top-k lists returned by ranking with the median ranks and ranking with the expected ranks. We adopted the techniques from [16] for



(a) Attr.-level uncertainty model. (b) Tuple-level uncertainty model.

Fig. 15. Normalized averaging Kendall distance for expected and median ranks with different k values.



(a) Attr.-level uncertainty model. (b) Tuple-level uncertainty model.

Fig. 16. Normalized averaging Kendall distance for median and quantile ranks with different k values.

this purpose. Specifically, for two top-k lists τ_1 and τ_2, we use the *averaging Kendall distance* to measure their similarity, denoted as $K_{avg}(\tau_1, \tau_2)$. $K_{avg}(\tau_1, \tau_2)$ is computed as

$$K_{avg}(\tau_1, \tau_2) = \sum_{\{i,j\} \in P(\tau_1, \tau_2)} \bar{K}_{i,j}^{(p)}(\tau_1, \tau_2), \text{for } p = 0.5 \quad (17)$$

where $\bar{K}_{i,j}^{(p)}(\tau_1, \tau_2)$ is defined as a penalty over the pairs in the set $P(\tau_1, \tau_2) = \{\{i, j\} | i \neq j \text{ and } i, j \in \tau_1 \cup \tau_2\}$, i.e. $P(\tau_1, \tau_2)$ is the set of unordered pairs of distinct elements in $\tau_1 \cup \tau_2$. The exact details of how the penalty $\bar{K}_{i,j}^{(p)}(\tau_1, \tau_2)$ is assigned for different pairs in the set P are found in [16]. A larger $K_{avg}(\tau_1, \tau_2)$ value indicates a higher *dissimilarity* between two top-k lists τ_1 and τ_2. By examining the assignment of the penalty to possible pairs in P, we can show that $K_{avg}(\tau_1, \tau_2) \in [0, k^2 + \binom{k}{2}]$ for any two top-k lists τ_1 and τ_2. The smallest value for $K_{avg}(\tau_1, \tau_2)$ happens when τ_1 and τ_2 are identical as two ordered sets; and the largest value for $K_{avg}(\tau_1, \tau_2)$ happens when τ_1 and τ_2 are completely disjoint. Hence, a meaningful way to represent the similarity between any two top-k lists, τ_1 and τ_2, is to use the *normalized averaging Kendall distance*, which is defined as:

$$K_{navg}(\tau_1, \tau_2) = \frac{K_{avg}(\tau_1, \tau_2)}{k^2 + \binom{k}{2}} \quad (18)$$

Clearly, $K_{navg}(\tau_1, \tau_2) \in [0, 1]$. Smaller $K_{navg}(\tau_1, \tau_2)$ values indicate higher similarity between τ_1 and τ_2, and larger values indicate lower similarity.

In Figures 15(a) and 15(b) we compare the similarity between the top-k results returned from the median ranks and expected ranks for both the attribute-level and tuple-level uncertainty models, using the normalized averaging Kendall distance. It is clear from the results in Figure 15(a) and 15(b) that the top-k lists produced by the median ranks and the expected ranks are rather different for both the attribute-level and the tuple-level uncertainty models, especially when k is small for the synthetic data sets. In general, the similarity between their top-k lists increases while k increases, but still maintains a clear difference. This shows that ranking by median and quantile ranks or by expected ranks will arrive at a different view of the top-k. This result is quite natural since median (quantile) ranks and expected ranks characterize different characteristics of the rank distributions $\mathrm{rank}(\mathrm{t_i})$ for all $t_i \in \mathcal{D}$, i.e. the 0.50-quantile (or other quantile values) and expectation of $\mathrm{rank}(\mathrm{t_i})$.

In Figures 16(a) and 16(b) we compare the similarity between the top-k results returned from the median ranks and different quantile ranks for the $movie$ data set for both the attribute-level and tuple-level uncertainty models, again using the normalized averaging Kendall distance. From these results we see that the similarity of the top-k lists produced by different quantile ranks and median ranks behaves in a very stable manner. As k increases from 0 to about 200 the similarity decreases quadratically between the different quantile ranks and the median ranks and when $k > 200$ we see that the similarity between the median ranks and the different quantile ranks remains roughly the same. Also notice that for all values of k, as the quantile approaches the median the normalized averaging Kendall distance approaches 0.