Ranking with Uncertain Scores

Mohamed A. Soliman Ihab F. Ilyas
School of Computer Science
University of Waterloo
{m2ali,ilyas}@cs.uwaterloo.ca

***Abstract*— Large databases with uncertain information are becoming more common in many applications including data integration, location tracking, and Web search. In these applications, ranking records with uncertain attributes needs to handle new problems that are fundamentally different from conventional ranking. Specifically, uncertainty in records' scores induces a partial order over records, as opposed to the total order that is assumed in the conventional ranking settings.**

In this paper, we present a new probabilistic model, based on partial orders, to encapsulate the space of possible rankings originating from score uncertainty. Under this model, we formulate several ranking query types with different semantics. We describe and analyze a set of efficient query evaluation algorithms. We show that our techniques can be used to solve the problem of rank aggregation in partial orders. In addition, we design novel sampling techniques to compute approximate query answers. Our experimental evaluation uses both real and synthetic data. The experimental study demonstrates the efficiency and effectiveness of our techniques in different settings.

I. Introduction

Uncertain data are becoming more common in many applications. Examples include managing sensor data, consolidating information sources, and opinion polls. Uncertainty impacts the quality of query answers in these environments. Dealing with data uncertainty by removing records with uncertain information is not desirable in many settings. For example, there could be too many uncertain values in the database (e.g., readings of sensing devices that become frequently unreliable under high temperature). Alternatively, there could be only few uncertain values in the database but they affect records that closely match query requirements. Dropping such records leads to inaccurate or incomplete query results. For these reasons, modeling and processing uncertain data have been the focus of many recent studies [1], [2], [3].

Top-k (ranking) queries report the k records with the highest scores in query output, based on a scoring function defined on one or more scoring predicates (e.g., columns of database tables, or functions defined on one or more columns). A scoring function induces a *total order* over records with different scores (ties are usually resolved using a deterministic tie-breaker such as unique record IDs [4]). A survey on the subject can be found in [5].

In this paper, we study ranking queries for records with uncertain scores. In contrast to the conventional ranking settings, score uncertainty induces a *partial order* over the underlying records, where multiple possible rankings are valid. Studying the formulation and processing of top-k queries in this context is lacking in the current proposals.

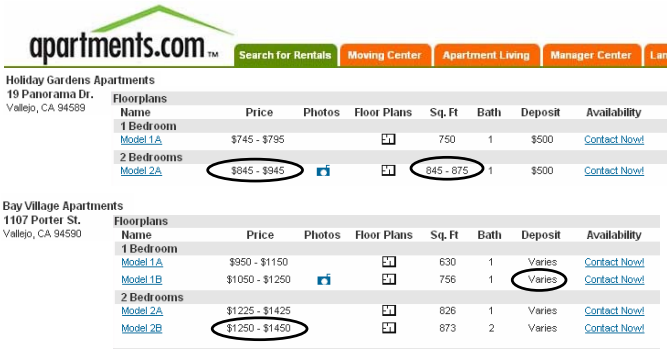


Fig. 1. Uncertain Data in Search Results

A. Motivation and Challenges

Consider Figure 1 which shows a snapshot of actual search results reported by $apartments.com$ for a simple search for available apartments to rent. The shown search results include several uncertain pieces of information. For example, some apartment listings do not explicitly specify the deposit amount. Other listings mention apartment rent and area as ranges rather than single values.

The obscure data in Figure 1 may originate from different sources including: (1) *data entry errors*, for example, an apartment listing is missing the number of rooms by mistake, (2) *integrating heterogeneous data sources*, for example, listings are obtained from sources with different schemas, (3) *privacy concerns*, for example, zip codes are anonymized, (4) *marketing policies*, for example, areas of small-size apartments are expressed as ranges rather than precise values, and (5) *presentation style*, for example, search results are aggregated to group similar apartments.

In a sample of search results we scrapped from $apartments.com$ and $carpages.ca$, the percentage of apartment records with uncertain rent was 65%, and the percentage of car records with uncertain price was 10%.

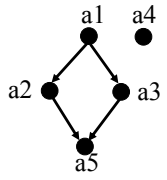
Uncertainty introduces new challenges regarding both the semantics and processing of ranking queries. We illustrate such challenges by giving the following simple example for the apartment search scenario in Figure 1.

Example 1: Assume an apartment database. Figure 2(a) gives a snapshot of the results of some user query posed against such database. Assume that the user would like to rank

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.102

AptID	Rent	Score
a1	$600	9
a2	[$650-$1100]	[5-8]
a3	$800	7
a4	*negotiable*	[0-10]
a5	$1200	4

(a)



(b)

Linear Extensions	
l_1	⟨a1,a4,a2,a3,a5⟩
l_2	⟨a1,a2,a3,a5,a4⟩
l_3	⟨a1,a3,a2,a5,a4⟩
l_4	⟨a1,a4,a3,a2,a5⟩
l_5	⟨a1,a2,a3,a4,a5⟩
l_6	⟨a1,a2,a4,a3,a5⟩
l_7	⟨a1,a3,a2,a4,a5⟩
l_8	⟨a1,a3,a4,a2,a5⟩
l_9	⟨a4,a1,a2,a3,a5⟩
l_{10}	⟨a4,a1,a3,a2,a5⟩

(c)

Fig. 2. Partial Order for Records with Uncertain Scores

the results using a function that scores apartments based on rent (the cheaper the apartment, the higher the score). Since the rent of apartment $a2$ is specified as a range, and the rent of apartment $a4$ is unknown, the scoring function assigns a range of possible scores to $a2$, while the full score range [1] $[0-10]$ is assigned to $a4$. □

Figure 2(b) depicts a diagram for the partial order induced by apartment scores (we formally define partial orders in Section II-A). Disconnected nodes in the diagram indicate the incomparability of their corresponding records. Due to the intersection of score ranges, $a4$ is incomparable to all other records, and $a2$ is incomparable to $a3$.

A simple approach to deal with the above partial order is to reduce it to a total order by replacing score ranges with their expected values. The problem with such approach, however, is that for score intervals with large variance, arbitrary rankings that are independent from how the ranges intersect may be produced. For example, assume 3 apartments, $a1$, $a2$, and $a3$ with uniform score intervals $[0, 100]$, $[40, 60]$, and $[30, 70]$, respectively. The expected score of each apartment is 50, and hence all apartment permutations are equally likely rankings. However, based on how the score intervals intersect, we show in Section IV that we can compute the probabilities of different rankings of these apartments as follows: $\Pr(\langle a1, a2, a3\rangle) = 0.25$, $\Pr(\langle a1, a3, a2\rangle) = 0.2$, $\Pr(\langle a2, a1, a3\rangle) = 0.05$, $\Pr(\langle a2, a3, a1\rangle) = 0.2$, $\Pr(\langle a3, a1, a2\rangle) = 0.05$, and $\Pr(\langle a3, a2, a1\rangle) = 0.25$. That is, the rankings have a non-uniform distribution even though the score intervals are uniform with equal expectations. Similar examples exist with non-uniform/skewed data.

Another possible ranking query on partial orders is finding the skyline (i.e., the non-dominated objects [8]). An object is non-dominated if, in the partial order diagram, the object's node has no incoming edges. In Example 1, the skyline objects are $\{a1, a4\}$. The number of skyline objects can vary from a small number (e.g., Example 1) to the whole database. Furthermore, skyline objects may not be equally good and, similarly, dominated objects may not be equally bad. A user may want to compare objects' relative orders in different data exploration scenarios. Current proposals [9], [10] have demonstrated that there is no unique way to distinguish or rank the skyline objects.

A different approach to rank the objects involved in a partial order is inspecting the space of possible rankings that conform to the relative order of objects. These rankings (or permutations) are called the *linear extensions* of the partial order. Figure 2(c) shows all linear extensions of the partial order in Figure 2(b). Inspecting the space of linear extensions allows ranking the objects in a way consistent with the partial order. For example, $a1$ may be preferred to $a4$ since $a1$ has rank 1 in 8 out of 10 linear extensions, even though both $a1$ and $a4$ are skyline objects. A crucial challenge for such approach is that the space of linear extensions grows exponentially in the number of objects [11].

Furthermore, in many scenarios, uncertainty is quantified probabilistically. For example, a moving object's location can be described using a probability distribution defined on some region based on location history [12]. Similarly, a missing attribute can be filled in with a probability distribution of multiple imputations, using machine learning methods [6], [7]. Augmenting uncertain scores with such probabilistic quantifications generates a (possibly non-uniform) probability distribution of linear extensions that cannot be captured using a standard partial order or dominance relationship.

In this paper, we address the challenges associated with dealing with uncertain scores and incorporating probabilistic score quantifications in both the semantics and processing of ranking queries. We summarize such challenges as follows:

- *Ranking Model:* The conventional total order model cannot capture score uncertainty. While partial orders can represent incomparable objects, incorporating probabilistic score information in such model requires new probabilistic modeling of partial orders.
- *Query Semantics:* Conventional ranking semantics assume that each record has a single score and a distinct rank (by resolving ties using a deterministic tie breaker). Query semantics allowing a score range, and hence different possible ranks per record needs to be adopted.
- *Query Processing:* Adopting a probabilistic partial order model yields a probability distribution over a huge space of possible rankings that is exponential in the database size. Hence, we need efficient algorithms to process such space in order to compute query answers.

B. Contributions

We present an integrated solution to compute ranking queries of different semantics under a general score uncertainty model. We tackle the problem through the following key contributions:

[1]Imputation methods [6], [7] can give better guesses for missing values. However, imputation is not the main focus of this paper.

tID	Score Interval	Score Density
t_1	[6 , 6]	$f_1 = 1$
t_2	[4 , 8]	$f_2 = 1/4$
t_3	[3 , 5]	$f_3 = 1/2$
t_4	[2 , 3.5]	$f_4 = 2/3$
t_5	[7 , 7]	$f_5 = 1$
t_6	[1 , 1]	$f_6 = 1$

Fig. 3. Modeling Score Uncertainty

- We introduce a novel probabilistic ranking model based on partial orders (Section II-A).
- We formulate the problem of ranking with score uncertainty by introducing multiple different semantics of ranking queries under our model (Section II-B).
- We introduce a space pruning algorithm to cut down the answer space allowing efficient query evaluation (Sections VI-A).
- We introduce a set of efficient query evaluation techniques. We show that exact query evaluation is expensive for some of our proposed queries (Section VI-C). We thus design novel sampling techniques based on a Markov Chain Monte-Carlo (MCMC) method to compute approximate answers (Section VI-D).
- We study the novel problem of optimal rank aggregation in partial orders. We give a polynomial time algorithm to solve the problem (Section VI-E).
- We conduct an extensive experimental study using real and synthetic data to examine the robustness and efficiency of our techniques in various settings (Section VII).

II. Data Model and Problem Definition

In this section, we describe the data model we adopt in this paper (Section II-A), followed by our problem definition (Section II-B).

A. Data Model

We adopt a general representation of uncertain scores, where the score of record t_i is modeled as a probability density function f_i defined on a score interval $[lo_i, up_i]$. The density f_i can be obtained directly from uncertain attributes (e.g., a uniform distribution on possible apartment's rent values as in Figure 1). Alternatively, the score density can be computed from the predictions of missing/incomplete attribute values that affect records' scores [6], or constructed from histories and value correlations as in sensor readings [13]. A deterministic (certain) score is modeled as an interval with equal bounds, and a probability of 1. For two records t_i and t_j with deterministic equal scores (i.e., $lo_i = up_i = lo_j = up_j$), we assume a tie-breaker $\tau(t_i, t_j)$ that gives a deterministic records' relative order. The tie-breaker τ is transitive over records with identical deterministic scores.

Figure 3 shows a set of records with uniform score densities, where $f_i = 1/(up_i - lo_i)$ (e.g., $f_2 = 1/4$). For records with deterministic scores (e.g., t_1), the density $f_i = 1$.

Our interval-based score representation induces a *partial order* over database records, which extends the following definition of *strict partial orders*:

Definition 1: **Strict Partial Order.** A strict partial order $\mathbb{P}$ is a 2-tuple $(\mathcal{R}, \mathcal{O})$, where $\mathcal{R}$ is a finite set of elements, and $\mathcal{O} \subset \mathcal{R} \times \mathcal{R}$ is a binary relation with the following properties:
(1) Non-reflexivity: $\forall i \in \mathcal{R} : (i, i) \notin \mathcal{O}$.
(2) Asymmetry: If $(i, j) \in \mathcal{O}$, then $(j, i) \notin \mathcal{O}$.
(3) Transitivity: If $\{(i, j), (j, k)\} \subset \mathcal{O}$, then $(j, k) \in \mathcal{O}$. □

Strict partial orders allow the relative order of some elements to be left undefined. A widely-used depiction of partial orders is Hasse diagram (e.g., Figure 2(b)), which is a directed acyclic graph whose nodes are the elements of $\mathcal{R}$, and edges are the binary relationships in $\mathcal{O}$, except the transitive relationships (relationships derived by transitivity). An edge (i, j) indicates that i is ranked higher than j according to $\mathbb{P}$. The *linear extensions* of a partial order are all possible topological sorts of the partial order graph (i.e., the relative order of any two elements in any linear extension does not violate the set of binary relationships $\mathcal{O}$).

Typically, a strict partial order $\mathbb{P}$ induces a uniform distribution over its linear extensions. For example, for $\mathbb{P} = (\{a, b, c\}, \{(a, b)\})$, the 3 possible linear extensions $\langle a, b, c\rangle$, $\langle a, c, b\rangle$, and $\langle c, a, b\rangle$ are equally likely.

We extend strict partial orders to encode score uncertainty based on the following definitions.

Definition 2: **Record Dominance.** A record t_i *dominates* another record t_j iff $lo_i \geq up_j$. □

The deterministic tie-breaker τ eliminates cycles when applying Definition 2 to records with deterministic equal scores.

Based on Definitions 2, Record Dominance is a *non-reflexive*, *asymmetric*, and *transitive* relation.

We assume the independence of score densities of individual records. Hence, the probability that record t_i is ranked higher than record t_j, denoted $\Pr(t_i > t_j)$, is given by the following 2-dimensional integral:

$$\Pr(t_i > t_j) = \int_{lo_i}^{up_i} \int_{lo_j}^{x} f_i(x) \cdot f_j(y) dy \; dx \qquad (1)$$

When neither t_i nor t_j dominates the other record, $[lo_i, up_i]$ and $[lo_j, up_j]$ are intersecting intervals, and so $\Pr(t_i > t_j)$ belongs to the open interval $(0, 1)$, where $\Pr(t_j > t_i) = 1 - \Pr(t_i > t_j)$. On the other hand, if t_i dominates t_j, then we have $\Pr(t_i > t_j) = 1$ and $\mathrm{P}(t_j > t_i) = 0$.

We say that a record pair (t_i, t_j) belongs to a *probabilistic dominance relation* iff $\Pr(t_i > t_j) \in (0, 1)$.

We next give the formal definition of our ranking model:

Definition 3: **Probabilistic Partial Order (PPO).** Let $\mathcal{R} = \{t_1, \ldots, t_n\}$ be a set of real intervals, where each interval $t_i = [lo_i, up_i]$ is associated with a density function f_i such that $\int_{lo_i}^{up_i} f_i(x)dx = 1$. The set $\mathcal{R}$ induces a probabilistic partial order $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$, where $(\mathcal{R}, \mathcal{O})$ is a strict partial order with $(t_i, t_j) \in \mathcal{O}$ iff t_i dominates t_j, and $\mathcal{P}$ is the *probabilistic dominance relation* of intervals in $\mathcal{R}$. □

Definition 3 states that if t_i dominates t_j, then $(t_i, t_j) \in \mathcal{O}$. That is, we can deterministically rank t_i on top of t_j. On the other hand, if neither t_i nor t_j dominates the other record, then $(t_i, t_j) \in \mathcal{P}$. That is, the uncertainty in the relative order of t_i and t_j is quantified by $\Pr(t_i > t_j)$.

Figure 4 shows the Hasse diagram and the probabilistic dominance relation of the PPO of records in Figure 3. We also show the set of linear extensions of the PPO.

The linear extensions of $\text{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$ can be viewed as tree where each root-to-leaf path is one linear extension. The root node is a dummy node since there can be multiple elements in $\mathcal{R}$ that may be ranked first. Each occurrence of an element $t \in \mathcal{R}$ in the tree represents a possible ranking of t, and each level i in the tree contains all elements that occur at rank i in any linear extension. We explain how to construct the linear extensions tree in Section V.

Due to probabilistic dominance, the space of possible linear extensions is a probability space generated by a probabilistic process that draws, for each record t_i, a random score $s_i \in [lo_i, up_i]$ based on the density f_i. Ranking the drawn scores gives a total order on the database records, where the probability of such order is the joint probability of the drawn scores. For example, we show in Figure 4, the probability value associated with each linear extension. We show how to compute these probabilities in Section IV.

B. Problem Definition

Based on the data model in Section II-A, we consider three classes of ranking queries:

(1) RECORD-RANK QUERIES: queries that produce records that appear in a given range of ranks, defined as follows:

Definition 4: **Uncertain Top Rank (UTop-Rank).** A UTop-Rank(i, j) query reports the most probable record to appear at any rank $i \dots j$ (i.e., from i to j inclusive) in possible linear extensions. That is, for a linear extensions space Ω of a PPO, UTop-Rank(i, j) query, for $i \leq j$, reports $argmax_t(\sum_{\omega \in \Omega_{(t,i,j)}} \Pr(\omega))$, where $\Omega_{(t,i,j)} \subseteq \Omega$ is the set of linear extensions with the record t at any rank $i \dots j$. □

For example, in Figure 4, a UTop-Rank$(1, 2)$ query reports t_5 with probability $\Pr(\omega_1) + \dots + \Pr(\omega_7) = 1.0$, since t_5 appears at all linear extensions at either rank 1 or rank 2.

(2) TOP-k-QUERIES: queries that produce a set of top-ranked records. We give two different semantics for TOP-k-QUERIES:

Definition 5: **Uncertain Top Prefix (UTop-Prefix).** A UTop-Prefix(k) query reports the most probable linear extension prefix of k records. That is, for a linear extensions space Ω of a PPO, UTop-Prefix(k) query reports $argmax_p(\sum_{\omega \in \Omega_{(p,k)}} \Pr(\omega))$, where $\Omega_{(p,k)} \subseteq \Omega$ is the set of linear extensions sharing the same k-length prefix p. □

For example, in Figure 4, a UTop-Prefix(3) query reports $\langle t_5, t_1, t_2 \rangle$ with probability $\Pr(\omega_1) + \Pr(\omega_2) = 0.438$.

Definition 6: **Uncertain Top Set (UTop-Set).** A UTop-Set(k) query reports the most probable set of top-k records of linear extensions. That is, for a linear extensions space Ω of a PPO, UTop-Set(k) query reports $argmax_s(\sum_{\omega \in \Omega_{(s,k)}} \Pr(\omega))$, where $\Omega_{(s,k)} \subseteq \Omega$ is the set of linear extensions sharing the same set of top-k records s. □

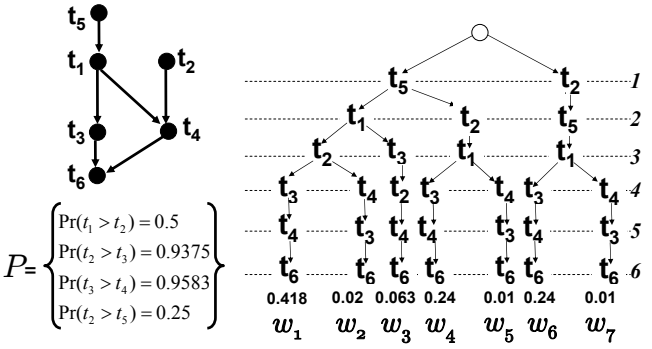


Fig. 4. Probabilistic Partial Order and Linear Extensions

For example, in Figure 4, UTop-Set(3) query reports the set $\{t_1, t_2, t_5\}$ with probability $\Pr(\omega_1) + \Pr(\omega_2) + \Pr(\omega_4) + \Pr(\omega_5) + \Pr(\omega_6) + \Pr(\omega_7) = 0.937$

Note that $\{t_1, t_2, t_5\}$ appears as Prefix $\langle t_5, t_1, t_2 \rangle$ in ω_1 and ω_2, appears as Prefix $\langle t_5, t_2, t_1 \rangle$ in ω_4 and ω_5, and appears as Prefix $\langle t_2, t_5, t_1 \rangle$ in ω_6 and ω_7. However, unlike the UTop-Prefix query, the UTop-Set query ignores the order of records within the query answer. This allows finding query answers with a relaxed within-answer ranking.

The above query definitions can be extended to rank different answers on probability. We define the answer of l-UTop-Rank(i, j) query as the l most probable records to appear at a rank $i \dots j$, the answer of l-UTop-Prefix(k) query as the l most probable linear extension prefixes of length k, and the answer of l-UTop-Set(k) query as the l most probable top-k sets. We assume a tie-breaker that deterministically orders answers with equal probabilities.

(3) RANK-AGGREGATION-QUERIES: queries that produce a ranking with the minimum average distance to all linear extensions, formally defined as follows:

Definition 7: **Rank Aggregation Query (Rank-Agg).** For a linear extensions space Ω, a Rank-Agg query reports a ranking ω^* that minimizes $\frac{1}{|\Omega|} \sum_{\omega \in \Omega} d(\omega^*, \omega)$, where $d(.)$ is a measure of the distance between two rankings. □

We give examples for Rank-Agg query in Section VI-E. We also show that this query can be mapped to a UTop-Rank query.

The answer space of the above queries is a projection on the linear extensions space. That is, the probability of an answer is the summation of the probabilities of linear extensions that support that answer. These semantics are analogous to possible worlds semantics in probabilistic databases [14], [3], where a database is viewed as a set of possible instances, and the probability of a query answer is the summation of the probabilities of database instances containing this answer.

UTop-Set and UTop-Prefix query answers are related. The top-k set probability of a set s is the summation of the top-k

prefix probabilities of all prefixes p that consist of the same records of s. Similarly, the top-rank$(1, k)$ probability of a record t is the summation of the top-rank(i, i) probabilities of t for $i = 1 \dots k$.

Similar query definitions are used in [15], [16], [17], under the membership uncertainty model where records belong to database with possibly less than absolute confidence, and scores are single values. However, our score uncertainty model (Section II-A) is fundamentally different, which entails different query processing techniques. Furthermore, to the best of our knowledge, the UTop-Set query has not been proposed before.

Applications. Example applications of our query types include the following:

- A UTop-Rank(i, j) query can be used to find the most probable athlete to end up in a range of ranks in some competition given a partial order of competitors.
- A UTop-Rank$(1, k)$ query can be used to find the most-likely location to be in the top-k hottest locations based on uncertain sensor readings represented as intervals.
- A UTop-Prefix query can be used in market analysis to find the most-likely product ranking based on fuzzy evaluations in users' reviews. Similarly, a UTop-Set query can be used to find a set of products that are most-likely to be ranked higher than all other products.

Naïve computation of the above queries requires materializing and aggregating the space of linear extensions, which is very expensive. We analyze the cost of such naïve aggregation in Section V. Our goal is to design efficient algorithms that overcome such prohibitive computational barrier.

III. Background

In this section, we give necessary background material on Monte-Carlo integration, which is used to construct our probability space, and Markov chains, which are used in our sampling-based techniques.

• **Monte-Carlo Integration.** Monte-Carlo integration [18] computes accurate estimate of the integral $\int_{\acute{\Gamma}} f(x)dx$, where $\acute{\Gamma}$ is an arbitrary volume, by sampling from another volume $\Gamma \supseteq \acute{\Gamma}$ in which uniform sampling and volume computation are easy. The volume $\acute{\Gamma}$ is estimated as the proportion of samples from Γ that are inside $\acute{\Gamma}$ multiplied by the volume of Γ. The average $f(x)$ over such samples is used to compute the integral. Specifically, let v be the volume of Γ, s be the total number of samples, and $x_1 \dots x_m$ be the samples that are inside $\acute{\Gamma}$. Then,

$$\int_{\acute{\Gamma}} f(x)dx \approx \frac{m}{s} \cdot v \cdot \frac{1}{m} \sum_{i=1}^{m} f(x_i) \tag{2}$$

The expected value of the above approximation is the true integral value with an $O(\frac{1}{\sqrt{s}})$ approximation error.

• **Markov Chains.** We give a brief description for the theory of Markov chains. We refer the reader to [19] for more detailed coverage of the subject. Let X be a random variable, where X_t denotes the value of X at time t. Let $S = \{s_1, \dots, s_n\}$ be the set of possible X values, denoted the *state space* of X. We say that X follows a Markov process if X moves from the current state to a next state based only on its current state. That is, $\Pr(X_{t+1} = s_i | X_0 = s_m, \dots, X_t = s_j) = \Pr(X_{t+1} = s_i | X_t = s_j)$. A Markov chain is a state sequence generated by a Markov process. The transition probability between a pair of states s_i and s_j, denoted $\Pr(s_i \rightarrow s_j)$, is the probability that the process at state s_i moves to state s_j in one step.

A Markov chain may reach a stationary distribution π over its state space, where the probability of being at a particular state is independent from the initial state of the chain. The conditions of reaching a stationary distribution are *irreducibility* (i.e., any state is reachable from any other state), and *aperiodicity* (i.e., the chain does not cycle between states in a deterministic number of steps). A unique stationary distribution is reachable if the following balance equation holds for every pair of states s_i and s_j:

$$\Pr(s_i \rightarrow s_j)\pi(s_i) = \Pr(s_j \rightarrow s_i)\pi(s_j) \tag{3}$$

• **Markov Chain Monte-Carlo (MCMC) Method.** The concepts of Monte-Carlo method and Markov chains are combined in the MCMC method [19] to simulate a complex distribution using a Markovian sampling process, where each sample depends only on the previous sample.

A standard MCMC algorithm is the Metropolis-Hastings (M-H) sampling algorithm [20]. Suppose that we are interested in drawing samples from a target distribution $\pi(x)$. The (M-H) algorithm generates a sequence of random draws of samples that follow $\pi(x)$ as follows:

1) Start from an initial sample x_0.
2) Generate a candidate sample x_1 from an *arbitrary* proposal distribution $q(x_1|x_0)$.
3) Accept the new sample x_1 with probability $\alpha = min(\frac{\pi(x_1).q(x_0|x_1)}{\pi(x_0).q(x_1|x_0)}, 1)$.
4) If x_1 is accepted, then set $x_0 = x_1$.
5) Repeat from step (2).

The (M-H) algorithm draws samples biased by their probabilities. At each step, a candidate sample x_1 is generated given the current sample x_0. The ratio α compares $\pi(x_1)$ and $\pi(x_0)$ to decide on accepting x_1. The (M-H) algorithm satisfies the balance condition (Equation 3) with arbitrary proposal distributions [20]. Hence, the algorithm converges to the target distribution π. The number of times a sample is visited is proportional to its probability, and hence the relative frequency of visiting a sample x is an estimate of $\pi(x)$. The (M-H) algorithm is typically used to compute distribution summaries or estimate a function of interest on π.

IV. Probability Space

In this section, we formulate and compute the probabilities of the linear extensions of a PPO.

The probability of a linear extension is computed as a nested integral over records' score densities in the order given by the linear extension. Let $\omega = \langle t_1, t_2, \dots t_n \rangle$ be a linear extension. Then, $\Pr(\omega) = \Pr((t_1 > t_2), (t_2 > t_3), \dots, (t_{n-1} > t_n))$. The individual events $(t_i > t_j)$ in the previous formulation

are not independent, since any two consecutive events share a record. Hence, For $\omega = \langle t_1, t_2, \dots t_n \rangle$, $\Pr(\omega)$ is given by the following n-dimensional integral with dependent limits:

$$\Pr(\omega) = \int_{lo_1}^{up_1} \int_{lo_2}^{x_1} \dots \int_{lo_n}^{x_{n-1}} f_1(x_1) \dots f_n(x_n) dx_n \dots \, dx_1 \tag{4}$$

Monte-Carlo integration (Section III) can be used to compute complex nested integrals such as Equation 4. For example, the probabilities of linear extensions $\omega_1, \dots, \omega_7$ in Figure 4 are computed using Monte-Carlo integration.

In the next theorem, we prove that the space of linear extensions of a PPO induces a probability distribution.

Theorem 1: Let Ω be the set of linear extensions of $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$. Then, (1) Ω is equivalent to the set of all possible rankings of $\mathcal{R}$, and (2) Equation 4 defines a probability distribution on Ω. □

Proof: We prove (1) by contradiction. Assume that $\omega \in \Omega$ is an invalid ranking of $\mathcal{R}$. That is, there exist at least two records t_i and t_j whose relative order in ω is $t_i > t_j$, while $lo_j \geq up_i$. However, this contradicts the definition of $\mathcal{O}$ in $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$. Similarly, we can prove that any valid ranking of $\mathcal{R}$ corresponds to only one linear extension in Ω.

We prove (2) as follows. First, map each linear extension $\omega = \langle t_1, \dots, t_n \rangle$ to its corresponding event $e = ((t_1 > t_2) \wedge \dots \wedge (t_{n-1} > t_n))$. Equation 4 computes $\Pr(e)$ or equivalently $\Pr(\omega)$. Second, let ω_1 and ω_2 be two linear extensions in Ω whose events are e_1 and e_2, respectively. By definition, ω_1 and ω_2 must be different in the relative order of at least one pair of records. It follows that $\Pr(e_1 \wedge e_2) = 0$ (i.e., any two linear extensions map to mutually exclusive events). Third, since Ω is equivalent to all possible rankings of $\mathcal{R}$ (as proved in (1)), the events corresponding to elements of Ω must completely cover a probability space of 1 (i.e., $\Pr(e_1 \vee e_2 \dots \vee e_m) = 1$, where $m = |\Omega|$). Since all e_i's are mutually exclusive, it follows that $\Pr(e_1 \vee e_2 \dots \vee e_m) = \Pr(e_1) + \dots + \Pr(e_m) = \sum_{\omega \in \Omega} \Pr(\omega) = 1$, and hence Equation 4 defines a probability distribution on Ω. ■

V. A Baseline Exact Algorithm

We describe a baseline algorithm that computes the queries in Section II-B by materializing the space. Algorithm 1 gives a simple recursive technique to build the linear extensions tree (Section II-A). The first call to Procedure BUILD_TREE is passed the parameters $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$, and a dummy root node. A record $t \in \mathcal{R}$ is a *source* if no other record $\acute{t} \in \mathcal{R}$ dominates t. The children of the tree root will be the initial sources in $\mathcal{R}$, so we can add a source t as a child of the root, remove it from $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$, and then recurse by finding new sources in $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$ after removing t.

The space of all linear extensions of $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$ grows exponentially in $|\mathcal{R}|$. As a simple example, suppose that $\mathcal{R}$ contains m elements, none of which is dominated by any other element. A counting argument shows that there are $\Sigma_{i=1}^{m} \frac{m!}{(m-i)!}$ nodes in the linear extensions tree.

Algorithm 1 Build Linear Extension Tree

```
BUILD_TREE (PPO(R, O, P) : PPO, n : Tree node)
1  for each source t ∈ R
2      do
3          child ← create a tree node for t
4          Add child to n's children
5          PPÓ ← PPO(R, O, P) after removing t
6          BUILD_TREE( PPÓ, child)
```

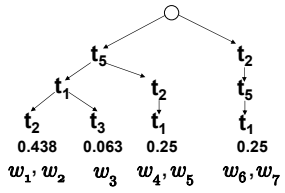


Fig. 5. Prefixes of Linear Extensions at Depth 3

When we are only interested in records occupying the top ranks, we can terminate the recursive construction algorithm at level k, which means that our space is reduced from complete linear extensions to linear extensions' prefixes of length k. Under our probability space, the probability of each prefix is the summation of the probabilities of linear extensions sharing that prefix. We can compute prefix probabilities more efficiently as follows. Let $\omega^{(k)} = \langle t_1, t_2, \dots, t_k \rangle$ be a linear extension prefix of length k. Let $T(\omega^{(k)})$ be the set of records not included in $\omega^{(k)}$. Let $\Pr(t_k > T(\omega^{(k)}))$ be the probability that t_k is ranked higher than all records in $T(\omega^{(k)})$. Let $F_i(x) = \int_{lo_i}^{x} f_i(y) dy$ be the cumulative density function (CDF) of f_i. Hence, $\Pr(\omega^{(k)}) = \Pr((t_1 > t_2), \dots, (t_{k-1} > t_k), (t_k > T(\omega^{(k)})))$, where

$$\Pr(t_k > T(\omega^{(k)})) = \int_{lo_k}^{up_k} f_k(x) \cdot \Big(\prod_{t_i \in T(\omega^{(k)})} F_i(x) \Big) dx \tag{5}$$

Hence, we have

$$\Pr(\omega^{(k)}) = \int_{lo_1}^{up_1} \int_{lo_2}^{x_1} \dots \int_{lo_k}^{x_{k-1}} f_1(x_1) \dots f_k(x_k) \cdot \Big(\prod_{t_i \in T(\omega^{(k)})} F_i(x_k) \Big) \, dx_k \dots dx_1 \tag{6}$$

Figure 5 shows the prefixes of length 3 and their probabilities for the linear extensions tree in Figure 4. We annotate the leaves with the linear extensions that share each prefix. Unfortunately, prefix enumeration is still infeasible for all but the smallest sets of elements, and, in addition, finding the probabilities of nodes in the prefix tree requires computing an l dimensional integral, where l is the node's level.

• **Query Evaluation.** The algorithm computes UTop-Prefix query by scanning the nodes in the prefix tree in depth-first search order, computing integrals only for the nodes at depth k (Equation 6), and reporting the prefixes with the highest probabilities. We can use these probabilities to answer UTop-Rank query for ranks $1 \dots k$, since the probability of

a node t at level $l < k$ can be found by summing the probabilities of its children. Once the nodes in the tree have been labeled with their probabilities, the answer of UTop-Rank(i, j), $\forall i, j \in [1, k]$ and $i \leq j$, can be constructed by summing up the probabilities of all occurrences of a record t at levels $i \ldots j$. This is easily done in time linear to the number of tree nodes using a breadth-first traversal of the tree. Here, we compute $\frac{m!}{(m-k)!}$ k-dimensional integrals to answer both queries. However, the algorithm still grows exponentially in m. Answering UTop-Set query can be done using the relationship among query answers discussed in Section II-B.

VI. Query Evaluation

The Baseline algorithm described in Section V exposes two fundamental challenges for efficient query evaluation:

1) Database size: The naïve algorithm is exponential in database size. How to make use of special indexes and other data structures to access a small proportion of database records while computing query answers?
2) Query evaluation cost: Computing probabilities by naïve simple aggregation is prohibitive. How to exploit query semantics for faster computation?

In Section VI-A, we answer the first question by using indexes to prune records that do not contribute to query answers, while in Sections VI-C and VI-D, we answer the second question by exploiting query semantics for faster computation.

A. k-Dominance: Shrinking the Database

Given a database D conforming to our data model, we call a record $t \in D$ "k-dominated" if at least k other records in D dominate t. For example in Figure 4, the records t_4 and t_6 are 3-dominated. Our main insight to shrink the database D used in query evaluation is based on Lemma 1.

Lemma 1: Any k-dominated record in D can be ignored while computing UTop-Rank(i, k) and Top-k queries. □

Lemma 1 follows from the fact that k-dominated records do not occupy ranks $\leq k$ in any linear extension, and so they do not affect the probability of any k-length prefix. Hence, k-dominated records can be safely pruned from D.

In the following, we describe a simple and efficient technique to shrink the database D by removing all k-dominated records. Our technique assumes a list U ordering records in D in descending score upper-bound (up_i) order, and that $t_{(k)}$, the record with the k^{th} largest score lower-bound (lo_i), is known (e.g., by using an index maintained over score lower-bounds). Ties among records are resolved using our deterministic tie breaker τ (Section II-A).

Algorithm 2 gives the details of our technique. The central idea is to conduct a binary search on U to find the record t^*, such that t^* is dominated by $t_{(k)}$, and t^* is located at the highest possible position in U. Based on Lemma 1, t^* is k-dominated. Moreover, let pos^* be the position of t^* in U, then all records located at positions $\geq pos^*$ in U are also k-dominated.

Complexity Analysis. Since Algorithm 2 conducts a binary search on U, its worst case complexity is in $O(log(m))$, where

Algorithm 2 Remove k-Dominated Records

SHRINK_DB (D: database, k: dominance level, U: score upper-bound list)

```
1  start ← 1; end ← |D|
2  pos* ← |D| + 1
3  t_(k) ← the record with the k^th largest lo_i
4  while (start ≤ end) {binary search}
5       do
6            mid ← (start+end)/2
7            t_i ← record at position mid in U
8            if (t_(k) dominates t_i)
9              then
10                   pos* ← mid
11                   end ← mid − 1
12             else  {t_(k) does not dominate records above t_i}
13                   start ← mid + 1
14 return D\ {t: t is located at position ≥ pos* in U }
```

$m = |D|$. The list U and the record $t_{(k)}$ can be pre-computed for heavily-used scoring functions with typical values of k (e.g., sensor reading in a sensor database, or the rent attribute in an apartment database). Otherwise, U is constructed by sorting D on up_i in $O(m\ log(m))$, while $t_{(k)}$ is found in $O(m\ log(k))$ by scanning D while maintaining a k-length priority queue for the top-k records with respect to lo_i's. The overall complexity in this case is $O(m\ log(m))$, which is the same complexity of sorting D.

In the remainder of this paper, we use $\acute{D}$ to refer to the database D after removing all k-dominated records.

B. Overview of Query Processing

There are two main factors impacting query evaluation cost: the size of answer space, and the cost of answer computation.

The size of the answer space of Record-Rank Queries is bounded by $|\acute{D}|$ (the number of records in $\acute{D}$), while for UTop-Set and UTop-Prefix queries, it is exponential in $|\acute{D}|$ (the number of record subsets of size k in $\acute{D}$). Hence, materializing the answer space for UTop-Rank queries is feasible, while materializing the answer space of UTop-Set and UTop-Prefix queries is very expensive (in general, it is intractable).

The computation cost of each answer can be heavily reduced by replacing the naïve probability aggregation algorithm (Section V) with simpler Monte-Carlo integration exploiting the query semantics to avoid enumerating the probability space.

Our goal is to design exact algorithms when the space size is manageable (Record-Rank Queries), and approximate algorithms when the space size is intractable (Top-k-Queries).

In the following, let $\acute{D} = \{t_1, t_2, \ldots, t_n\}$, where $n = |\acute{D}|$. Let Γ be the n-dimensional hypercube that consists of all possible combinations of records' scores. That is, $\Gamma = ([lo_1, up_1] \times [lo_2, up_2] \times \cdots \times [lo_n, up_n])$. A vector $\gamma = (x_1, x_2, \ldots, x_n)$ of n real values, where $x_i \in [lo_i, up_i]$, represents one point in Γ. Let $\Pi_{\acute{D}}(\gamma) = \prod_{i=1}^{n} f_i(x_i)$, where f_i is the score density of record t_i.

C. Computing RECORD-RANK QUERIES

We start by defining records' rank intervals.

Definition 8: **Rank Interval.** The rank interval of a record $t \in \acute{D}$ is the range of all possible ranks of t in the linear extensions of the PPO induced by $\acute{D}$. □

For a record $t \in \acute{D}$, let $\overline{\acute{D}}(t) \subseteq \acute{D}$ and $\underline{\acute{D}}(t) \subseteq \acute{D}$ be the record subsets dominating t and dominated by t, respectively. Then, based on the semantics of partial orders, the rank interval of t is given by $[|\overline{\acute{D}}(t)| + 1, n - |\underline{\acute{D}}(t)|]$.

For example, in Figure 4, for $\acute{D} = \{t_1, t_2, t_3, t_5\}$, we have $\overline{\acute{D}}(t_5) = \phi$, and $\underline{\acute{D}}(t_5) = \{t_1, t_3\}$, and thus the rank interval of t_5 is $[1, 2]$.

The shrinking algorithm in Section VI-A does not affect record ranks smaller than k, since any k-dominated record appears only at ranks $> k$.

Hence, given a range of ranks $i \dots j$, we know that a record t has non-zero probability to be in the answer of UTop-Rank(i, j) query only if its rank interval intersects $[i, j]$.

We compute UTop-Rank(i, j) query using Monte-Carlo integration. The main insight is transforming the complex space of linear extensions, that have to be aggregated to compute query answer, to the simpler space of all possible score combinations Γ. Such space can be sampled uniformly and independently to find the probability of query answer without enumerating the linear extensions. The accuracy of the result depends only on the number of drawn samples s (cf. Section III). We assume that the number of samples is chosen such that the error (which is in $O(\frac{1}{\sqrt{s}})$) is tolerated. We experimentally verify in Section VII that we obtain query answers with high accuracy and a considerably small cost using such strategy.

For a record t_k, we draw a sample $\gamma \in \Gamma$ as follows:

1) generate the value x_k in γ
2) generate $n-1$ independent values for other components in γ one by one.
3) If at any point there are j values in γ greater than x_k, reject γ.
4) Eventually, if the rank of x_k in γ is in $i \dots j$, accept γ.

Let $\lambda_{(i,j)}(t_k)$ be the probability of t_k to appear at rank $i \dots j$. The above procedure is formalized by the following integral:

$$\lambda_{(i,j)}(t_k) = \int_{\Gamma_{(i,j,t_k)}} \Pi_{\acute{D}}(\gamma)\ d\gamma \tag{7}$$

where $\Gamma_{(i,j,t_k)} \subseteq \Gamma$ is the volume defined by the points $\gamma = (x_1, \dots, x_n)$, with x_k's rank is in $i \dots j$. The integral in Equation 7 is evaluated as discussed in Section III.

Complexity Analysis. Let s be the total number of samples drawn from Γ to evaluate Equation 7. In order to compute the l most probable records to appear at a rank in $i \dots j$, we need to apply Equation 7 to each record in $\acute{D}$ whose rank interval intersects $[i, j]$, and use a heap of size l to maintain the l most probable records. Hence, computing l-UTop-Rank(i, j) query has a complexity of $O(s \cdot n_{(i,j)} \cdot log(l))$, where $n_{(i,j)}$ is the number of records in $\acute{D}$ whose rank intervals intersect $[i, j]$. In the worst case, $n_{(i,j)} = n$.

D. Computing TOP-k-QUERIES

Let v be a linear extension prefix of k records, and u be a set of k records. Let $\theta(v)$ be the top-k prefix probability of v, and $\Theta(u)$ be the top-k set probability of u.

Similar to our discussion of UTop-Rank queries in Section VI-C, $\theta(v)$ can be computed using Monte-Carlo integration on the volume $\Gamma_{(v)} \subseteq \Gamma$ which consists of the points $\gamma = (x_1, \dots, x_n)$ such that the values in γ that correspond to records in v have the same ranking as the ranking of records in v, and any other value in γ is smaller than the value corresponding to the last record in v. On the other hand, $\Theta(u)$ is computed by integrating on the volume $\Gamma_{(u)} \subseteq \Gamma$ which consists of the points $\gamma = (x_1, \dots, x_n)$ such that any value in γ that does not correspond to a record in u is smaller than the minimum value that corresponds to a record in u.

The cost of the previous integration procedure can be further improved using the CDF product of remaining records in $\acute{D}$, as described in Equation 6.

The above integrals have comparable cost to Equation 7. However, the number of integrals we need to evaluate here is exponential (one integral per each top-k prefix/set), while it is linear for UTop-Rank queries (one integral per each record). We thus design sampling techniques, based on the (M-H) algorithm (cf. Section III), to derive approximate query answers.

• **Sampling Space.** A state in our space is a linear extension ω of the PPO induced by $\acute{D}$. Let $\pi(\omega)$ be the probability of the top-k prefix, or the top-k set in ω, depending on whether we simulate θ or Θ, respectively. The main intuition of our sample generator is to *propose* states with high probabilities in a light-weight fashion. This is done by shuffling the ranking of records in ω biased by the weights of pairwise rankings (Equation 1). This approach guarantees sampling valid linear extensions since ranks are shuffled only when records probabilistically dominate each other.

Given a state ω_i, a candidate state ω_{i+1} is generated as follows:

1) Generate a random number $z \in [1, k]$.
2) For $j = 1 \dots z$ do the following:
   a) Randomly pick a rank r_j in ω_i. Let $t_{(r_j)}$ be the record at rank r_j in ω_i.
   b) If $r_j \in [1, k]$, move $t_{(r_j)}$ downward in ω_i, otherwise move $t_{(r_j)}$ upward. This is done by swapping $t_{(r_j)}$ with lower records in ω_i if $r_j \in [1, k]$, or with upper records if $r_j \notin [1, k]$. Swaps are conducted one by one, where swapping records $t_{(r_j)}$ and $t_{(m)}$ is committed with probability $P_{(r_j,m)} = \Pr(t_{(r_j)} > t_{(m)})$ if $r_j > m$, or with probability $P_{(m,r_j)} = \Pr(t_{(m)} > t_{(r_j)})$ otherwise. Record swapping stops at the first uncommitted swap.

The (M-H) algorithm is proven to converge with arbitrary proposal distributions [20]. Our proposal distribution $q(\omega_{i+1}|\omega_i)$ is defined as follows. In the above sample generator, at each step j, assume that $t_{(r_j)}$ has moved to a rank $r < r_j$. Let $R_{(r_j,r)} = \{r_j - 1, r_j - 2, \dots, r\}$. Let $P_j =$

$\prod_{m \in R_{(r_j,r)}} P_{(r_j,m)}$. Similarly, P_j can be defined for $r > r_j$. Then, the proposal distribution $q(\omega_{i+1}|\omega_i) = \prod_{j=1}^{z} P_j$, due to independence of steps. Based on the (M-H) algorithm, ω_{i+1} is accepted with probability $\alpha = min(\frac{\pi(\omega_{i+1}).q(\omega_i|\omega_{i+1})}{\pi(\omega_i).q(\omega_{i+1}|\omega_i)}, 1)$.

• **Computing Query Answers.** The (M-H) sampler simulates the top-k prefixes/sets distribution using a Markov chain (a random walk) that visits states biased by probability. Gelman and Rubin [21] argued that it is not generally possible to use a single simulation to infer distribution characteristics. The main problem is that the initial state may trap the random walk for many iterations in some region in the target distribution. The problem is solved by taking dispersed starting states and running multiple iterative simulations that independently explore the underlying distribution.

We thus run multiple independent Markov chains, where each chain starts from an independently selected initial state, and each chain simulates the space independently of all other chains. The initial state of each chain is obtained by independently selecting a random score value from each score interval, and ranking the records based on the drawn scores, resulting in a valid linear extension.

A crucial point is determining whether the chains have mixed with the target distribution (i.e., whether the current status of the simulation closely approximates the target distribution). At mixing time, the Markov chains produce samples that closely follow the target distribution and hence can be used to infer distribution characteristics. In order to judge chains mixing, we used the Gelman-Rubin diagnostic [21], a widely-used statistic in evaluating the convergence of multiple independent Markov chains [22]. The statistic is based on the idea that if a model has converged, then the behavior of all chains simulating the same distribution should be the same. This is evaluated by comparing the within-chain distribution variance to the across-chains variance. As the chains mix with the target distribution, the value of the Gelman-Rubin statistic approaches 1.0.

At mixing time, which is determined by the value of convergence diagnostic, each chain approximates the distribution's mode as the most probable visited state (similar to simulated annealing). The l most probable visited states across all chains approximate the l-UTop-Prefix (or l-UTop-Set) query answers. Such approximation improves as the simulation runs for longer times. The question is, at any point during simulation, how far is the approximation from the exact query answer?

We derive an upper-bound on the probability of any possible top-k prefix/set as follows. The top-k prefix probability of a prefix $\langle t_{(1)}, \ldots, t_{(k)} \rangle$ is equal to the probability of the event $e = ((t_{(1)} \text{ ranked } 1^{st}) \wedge \cdots \wedge (t_{(k)} \text{ ranked } k^{th}))$. Let $\lambda_i(t)$ be the probability of record t to be at rank i. Based on the principles of probability theory, we have $\Pr(e) \leq \min_{i=1}^{k} \lambda_i(t_{(i)})$. Hence, the top-$k$ prefix probability of any k-length prefix cannot exceed $\min_{i=1}^{k}(\max_{j=1}^{n} \lambda_i(t_j))$. Similarly, Let $\lambda_{1,k}(t)$ be the probability of record t to be at rank $1 \ldots k$. It can be shown that the top-k set probability of any k-length set cannot exceed the k^{th} largest $\lambda_{1,k}(t)$ value. The values of $\lambda_i(t)$ and $\lambda_{1,k}(t)$ are computed as discussed in Section VI-C. The approximation error is given by the difference between the top-k prefix/set probability upper-bound and the probability of the most probable state visited during simulation.

We note that the previous approximation error can overestimate the actual error, and that chains mixing time varies based on the fluctuations in the target distribution. However, we show in Section VII that, in practice, using multiple chains can closely approximate the true top-k states, and that the actual approximation error diminishes by increasing the number of chains. We also comment in Section VIII on the applicability of our techniques to other error estimation methods.

• **Caching.** Our sample generator mainly uses 2-dimensional integrals (Equation 1) to bias generating a sample by its probability. Such 2-dimensional integrals are shared among many states. Similarly, since we use multiple chains to simulate the same distribution from different starting points, some states can be repeatedly visited by different chains. Hence, we cache the computed $\Pr(t_i > t_j)$ values and state probabilities during simulation to be reused at a small cost.

E. Computing RANK-AGGREGATION-QUERIES

Rank aggregation is the problem of computing a consensus ranking for a set of candidates $\mathcal{C}$ using input rankings of $\mathcal{C}$ coming from different voters. The problem has immediate applications in Web meta-search engines [23].

While our work is mainly concerned with ranking under possible worlds semantics (Section II-B), we note that a strong resemblance exists between ranking in possible worlds and the rank aggregation problem. To the best of our knowledge, we give the first identified relation between the two problems.

Measuring the distance between two rankings of $\mathcal{C}$ is central to rank aggregation. Given two rankings ω_i and ω_j, let $\omega_i(c)$ and $\omega_j(c)$ be the positions of a candidate $c \in \mathcal{C}$ in ω_i and ω_j, respectively. A widely used measure of the distance between two rankings is the Spearman footrule distance, defined as follows:

$$F(\omega_i, \omega_j) = \sum_{c \in \mathcal{C}} |\omega_i(c) - \omega_j(c)| \tag{8}$$

The optimal rank aggregation is the ranking with the minimum average distance to all input rankings.

Optimal rank aggregation under footrule distance can be computed in polynomial time by the following algorithm [23]. Given a set of rankings $\omega_1 \ldots \omega_m$, the objective is to find the optimal ranking ω^* that minimizes $\frac{1}{m}\sum_{i=1}^{m} F(\omega^*, \omega_i)$. The problem is modeled using a weighted bipartite graph G with two sets of nodes. The first set has a node for each candidate, while the second set has a node for each rank. Each candidate c and rank r are connected with an edge (c, r) whose weight $w(c, r) = \sum_{i=1}^{m} |\omega_i(c) - r|$. Then, ω^* is given by "the minimum cost perfect matching" of G, where a perfect matching is a subset of graph edges such that every node is connected to exactly one edge, while the matching cost is the summation of the weights of its edges. Finding such matching can be done in $O(n^{2.5})$, where n is the number of graph nodes.

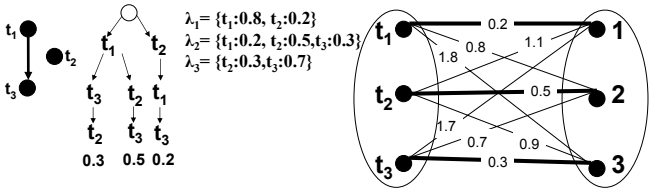


Fig. 6. Bipartite Graph Matching

In our settings, viewing each linear extension as a voter gives us an instance of the rank aggregation problem on a huge number of voters. The objective is to find the optimal linear extension that has the minimum average distance to all linear extensions. We show that we can solve this problem in polynomial time, under footrule distance, given $\lambda_i(t)$ (the probability of record t to appear at each rank i, or, equivalently, the summation of the probabilities of all linear extensions having t at rank i).

Theorem 2: For a $\mathrm{PPO}(\mathcal{R}, \mathcal{O}, \mathcal{P})$ defined on n records, the optimal rank aggregation of the linear extensions, under footrule distance, can be solved in time polynomial in n using the distributions $\lambda_i(t)$ for $i = 1 \ldots n$. □

Proof: For each linear extension ω_i of PPO, assume that we duplicate ω_i a number of times proportional to $\Pr(\omega_i)$. Let $\acute{\Omega} = \{\acute{\omega_1}, \ldots, \acute{\omega_m}\}$ be the set of all linear extensions' duplicates created in this way. Then, in the bipartite graph model, the edge connecting record t and rank r has a weight $w(t,r) = \sum_{i=1}^{|\acute{\Omega}|} |\acute{\omega_i}(t) - r|$, which is the same as $\sum_{j=1}^{n}(n_j \times |j - r|)$, where n_j is the number of linear extensions in $\acute{\Omega}$ having t at rank j. Dividing by $|\acute{\Omega}|$, we get $\frac{w(t,r)}{|\acute{\Omega}|} = \sum_{j=1}^{n}(\frac{n_j}{|\acute{\Omega}|} \times |j - r|) = \sum_{j=1}^{n}(\lambda_j(t) \times |j - r|)$. Hence, using $\lambda_i(t)$'s, we can compute $w(t,r)$ for every edge (t,r) divided by a fixed constant $|\acute{\Omega}|$, and thus the polynomial matching algorithm applies. ■

The intuition of Theorem 2 is that λ_i's provide compact summaries of voter's opinions, which allows us to efficiently compute the graph edge weights without expanding the space of linear extensions. The distributions λ_i's are obtained by applying Equation 7 at each rank i separately, yielding a quadratic cost in the number of records n.

Figure 6 shows an example illustrating our technique. The probabilities of the depicted linear extensions are summarized as λ_i's without expanding the space (Section VI-C). The λ_i's are used to compute the weights in the bipartite graph yielding $\langle t_1, t_2, t_3 \rangle$ as the optimal linear extension.

VII. Experiments

All experiments are conducted on a SunFire X4100 server with two Dual Core 2.2GHz processors, and 2GB of RAM. We used both real and synthetic data to evaluate our methods under different configurations. We experiment with two real datasets: (1) Apts: 33,000 apartment listings obtained by scrapping the search results of $apartments.com$, and (2) Cars: 10,000 car ads scrapped from $carpages.ca$. The *rent* attribute in Apts is used as the scoring function (65% of scrapped apartment listings have uncertain rent values), and similarly, the *price* attribute in Cars is used as the scoring function (10% of scrapped car ads have uncertain price).

The synthetic data sets have different distributions of score intervals' bounds: (1) Syn-u-0.5: bounds are uniformly distributed, (2) Syn-g-0.5: bounds are drawn from Gaussian distribution, and (3) Syn-e-0.5: bounds are drawn from exponential distribution. The proportion of records with uncertain scores in each dataset is 50%, and the size of each dataset is 100,000 records. In all experiments, the score densities (f_i's) are taken as uniform.

A. Shrinking Database by k-Dominance

We evaluate the performance of the database shrinking algorithm (Algorithm 2). Figure 7 shows the database size reduction due to k-dominance (Lemma 1) with different k values. The maximum reduction, around 98%, is obtained with the Syn-e-0.5 dataset. The reason is that the skewed distribution of score bounds results in a few records dominating the majority of other database records.

Figure 8 shows the number of record accesses used to find the pruning position pos^* in the list U (Section VI-A). The logarithmic complexity of the algorithm is demonstrated by the small number of performed record accesses, which is under 20 accesses in all datasets. The time consumed to construct the list U is under 1 second, while the time consumed by Algorithm 2 is under 0.2 second, in all datasets.

B. Accuracy and Efficiency of Monte-Carlo Integration

We evaluate the accuracy and efficiency of Monte-Carlo integration in computing UTop-Rank queries. The probabilities computed by the BASELINE algorithm are taken as the ground truth in accuracy evaluation. For each rank $i = 1 \ldots 10$, we compute the relative difference between the probability of record t to be at rank i, computed as in Section VI-C, and the same probability as computed by the BASELINE algorithm. We average this relative error across all records, and then across all ranks to get the total average error. Figure 9 shows the relative error with different space sizes (different number of linear extensions' prefixes processed by BASELINE). The different space sizes are obtained by experimenting with different subsets from the Apts dataset. The relative error is sensitive to the number of samples, and not to the space size. For example, increasing the number of samples from 2,000 to 30,000 diminishes the relative error by almost half, while for the same sample size, the relative error only doubled when the space size increased by 100 times.

Figure 10 compares (in log-scale) the efficiency of Monte-Carlo integration against the BASELINE algorithm. While the time consumed by Monte-Carlo integration is fixed with the same number of samples regardless the space size, the time consumed by the BASELINE algorithm increases exponentially when increasing the space size. For example, for a space of 2.5 million prefixes, Monte-Carlo integration consumes only 0.025% of the time consumed by the BASELINE algorithm.

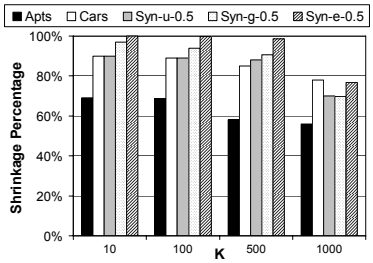


Fig. 7. Reduction in Data Size

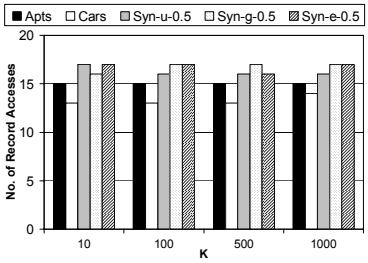


Fig. 8. Number of Record Accesses

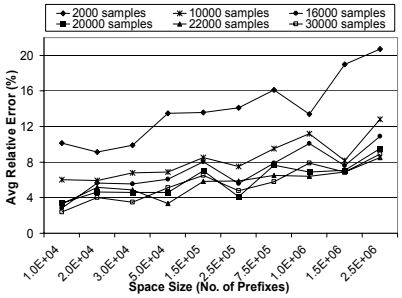


Fig. 9. Accuracy of Monte-Carlo Integration

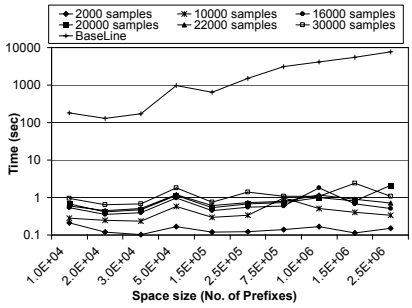


Fig. 10. Comparison with BASELINE

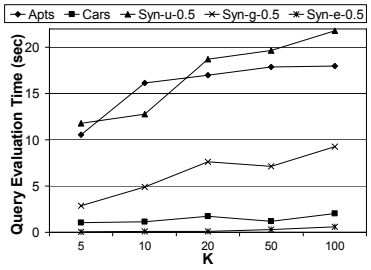


Fig. 11. UTop-Rank Query Evaluation Time

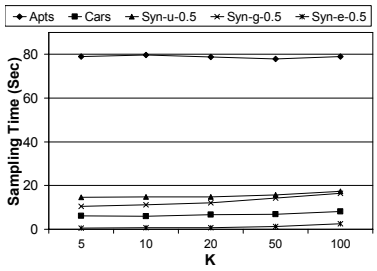


Fig. 12. UTop-Rank Sampling Time (10,000 Samples)

C. Scalability with respect to k

We evaluate the efficiency of our query evaluation for UTop-Rank$(1, k)$ queries with different k values. Figure 11 shows the query evaluation time, based on 10,000 samples. On the average, query evaluation time doubled when k increased by 20 times. Figure 12 shows the time consumed in drawing and ranking the samples. We obtain different sampling times with different datasets due to the variance in the reduced sizes of the datasets based on the k-dominance criterion.

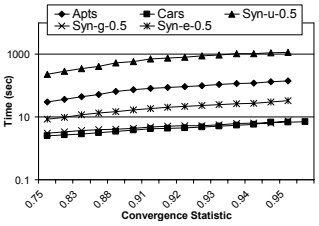


Fig. 13. Chains Convergence

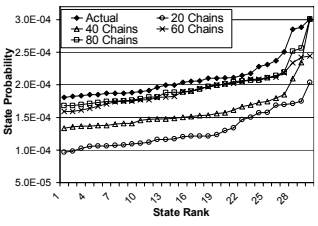


Fig. 14. Space Coverage

D. Markov Chains Convergence

We evaluate the Markov chains mixing time (Section VI-D). For 10 chains and $k = 10$, Figure 13 illustrates the Markov chains convergence based on the value of Gelman-Rubin statistic as time increases. While convergence consumes less than one minute in all real datasets, and most of the synthetic datasets, the convergence is notably slower for the Syn-u-0.5 dataset. The interpretation is that the uniform distribution of the score intervals in Syn-u-0.5 increases the size of the prefixes space, and hence the Markov chains consume more time to cover the space and mix with the target distribution. In real datasets, however, we note that the score intervals are mostly clustered, since many records have similar or the same attribute values. Hence, such delay in covering the space does not occur.

E. Markov Chains Accuracy

We evaluate the ability of Markov chains to discover states whose probabilities are close to the most probable states. We compare the most probable states discovered by the Markov chains to the true envelop of the target distribution (taken as the 30 most probable states). After mixing, the chains produce representative samples from the space, and hence states with high probabilities are frequently reached. This behavior is illustrated by Figure 14 for UTop-Prefix(5) query on a space of 2.5 million prefixes drawn from the Apts dataset. We compare the probabilities of the actual 30 most probable states and the 30 most probable states discovered by a number of independent chains after convergence, where the number of chains ranges from 20 to 80 chains.

The relative difference between the actual distribution envelop and the envelop induced by the chains decreases as the number of chains increase. The relative difference goes from 39% with 20 chains to 7% with 80 chains. The largest number of drawn samples is 70,000 (around 3% of the space size), and is produced using 80 chains. The convergence time increased from 10 seconds to 400 seconds when the number of chains increased from 20 to 80.

VIII. RELATED WORK

Several recent works have addressed query processing in probabilistic databases. The TRIO project [1], [2] introduced

different models to capture data uncertainty on different levels focusing on relating uncertainty with lineage. The ORION project [12], handles constantly evolving data using efficient query processing and indexing techniques designed to manage uncertain data in the form of continuous intervals. The problems of score-based ranking and top-k processing have not been addressed in these works.

Probabilistic top-k queries have been first proposed in [15], while [16], [17] proposed other query semantics and efficient processing algorithms. The uncertainty model in all of these works assume that records have deterministic single-valued scores, and they are associated with membership probabilities. The proposed techniques assume that uncertainty in ranking stems only from the existence/non-existence of records in possible worlds. Hence, these methods cannot be used when scores are in the form of ranges that induce a partial order on database records.

Dealing with the linear extensions of a partial order has been addressed in other contexts (e.g., [11], [24]). These techniques mainly focus on the theoretical aspects of uniform sampling from the space of linear extensions for purposes like estimating the count of possible linear extensions. Using linear extensions to model uncertainty in score-based ranking is not addressed in these works. To the best of our knowledge, defining a probability space on the set of linear extensions to quantify the likelihood of possible rankings is novel.

Monte-Carlo methods are used in [25] to compute top-k queries, where the objective is to find the top-k probable records in the answer of conjunctive queries that do not have the score-based ranking aspect discussed in this paper. Hence, the data model, problem definition, and processing techniques are quite different in both papers. For example, the proposed Monte-Carlo multi-simulation method in [25] is mainly used to estimate the satisfiability ratios of DNF formulae corresponding to the membership probabilities of individual records, while our focus is estimating and aggregating the probabilities of individual rankings of multiple records.

The techniques in [26] draw i.i.d. samples from the underlying distribution to compute statistical bounds on how far is the sample-based top-k estimate from the true top-k values in the distribution. This is done by fitting a gamma distribution encoding the relationship between the distribution tail (where the true top-k values are located), and its bulk (where samples are frequently drawn). The gamma distribution gives the probability that a value that is better than the sample-based top-k values exists in the underlying distribution. In our TOP-k-QUERIES, it is not straightforward to draw i.i.d. samples from the top-k prefix/set distribution. Our MCMC method produces such samples using independent Markov chains after mixing time. This allows using methods similar to [26] to estimate the approximation error.

IX. CONCLUSION

In this paper, we introduced a novel probabilistic model that extends partial orders to represent the uncertainty in the scores of database records. The model encapsulates a probability distribution on all possible rankings of database records. We formulated several types of ranking queries on such model. We designed novel query processing techniques including sampling-based methods based on Markov chains to compute approximate query answers. We also gave a polynomial time algorithm to solve the rank aggregation problem in partial orders, based on our model. Our experimental study on both real and synthetic datasets demonstrates the scalability and accuracy of our techniques.

REFERENCES

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

[2] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom, "Uldbs: Databases with uncertainty and lineage," in *VLDB*, 2006.

[3] D. S. Nilesh N. Dalvi, "Efficient query evaluation on probabilistic databases," in *VLDB*, 2004.

[4] K. C.-C. Chang and S. won Hwang, "Minimal probing: supporting expensive predicates for top-k queries," in *SIGMOD*, 2002.

[5] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, 2008.

[6] G. Wolf, H. Khatri, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati, "Query processing over incomplete autonomous databases," in *VLDB*, 2007.

[7] X. Wu and D. Barbará, "Learning missing values from summary constraints," *SIGKDD Explorations*, vol. 4, no. 1, 2002.

[8] J. Chomicki, "Preference formulas in relational queries," *ACM Trans. Database Syst.*, vol. 28, no. 4, 2003.

[9] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *SIGMOD*, 2006.

[10] Y. Tao, X. Xiao, and J. Pei, "Efficient skyline and top-k retrieval in subspaces," *TKDE*, vol. 19, no. 8, 2007.

[11] G. Brightwell and P. Winkler, "Counting linear extensions is #p-complete," in *STOC*, 1991.

[12] R. Cheng, S. Prabhakar, and D. V. Kalashnikov, "Querying imprecise data in moving object environments," in *ICDE*, 2003.

[13] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-based approximate querying in sensor networks," *VLDB J.*, vol. 14, no. 4, 2005.

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

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

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

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

[18] D. P. O'Leary, "Multidimensional integration: Partition and conquer," *Computing in Science and Engineering*, vol. 6, no. 6, 2004.

[19] M. Jerrum and A. Sinclair, "The markov chain monte carlo method: an approach to approximate counting and integration," 1997.

[20] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, 1970.

[21] A. Gelman and D. B. Rubin, "Inference from iterative simulation using multiple sequences," *Statistical Science*, vol. 7, no. 4, 1992.

[22] M. K. Cowles and B. P. Carlin, "Markov chain Monte Carlo convergence diagnostics: A comparative review," *Journal of the American Statistical Association*, vol. 91, no. 434, 1996.

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

[24] R. Bubley and M. Dyer, "Faster random generation of linear extensions," in *SODA*, 1998.

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

[26] M. Wu and C. Jermaine, "A bayesian method for guessing the extreme values in a data set," in *VLDB*, 2007.