The VLDB Journal (2010) 19:477–501
DOI 10.1007/s00778-009-0176-8

REGULAR PAPER

Supporting ranking queries on uncertain and incomplete data

Mohamed A. Soliman · Ihab F. Ilyas · Shalev Ben-David

Received: 12 May 2009 / Revised: 28 November 2009 / Accepted: 12 December 2009 / Published online: 10 February 2010

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 introduces 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 under two widely adopted distance metrics. In addition, we design sampling techniques based on Markov chains 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 under various configurations.

Keywords Ranking · Top-k · Uncertain data · Probabilistic data · Partial orders · Rank aggregation · Kendall tau

M. A. Soliman (✉) · I. F. Ilyas · S. Ben-David
School of Computer Science, University of Waterloo,
Waterloo, Canada
e-mail: m2ali@cs.uwaterloo.ca

I. F. Ilyas
e-mail: ilyas@cs.uwaterloo.ca

S. Ben-David
e-mail: s4bendavid@uwaterloo.ca

1 Introduction

Uncertain data are becoming more common in many applications. Examples include managing sensor data, consolidating information sources, and tracking moving objects. 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 [1–3].

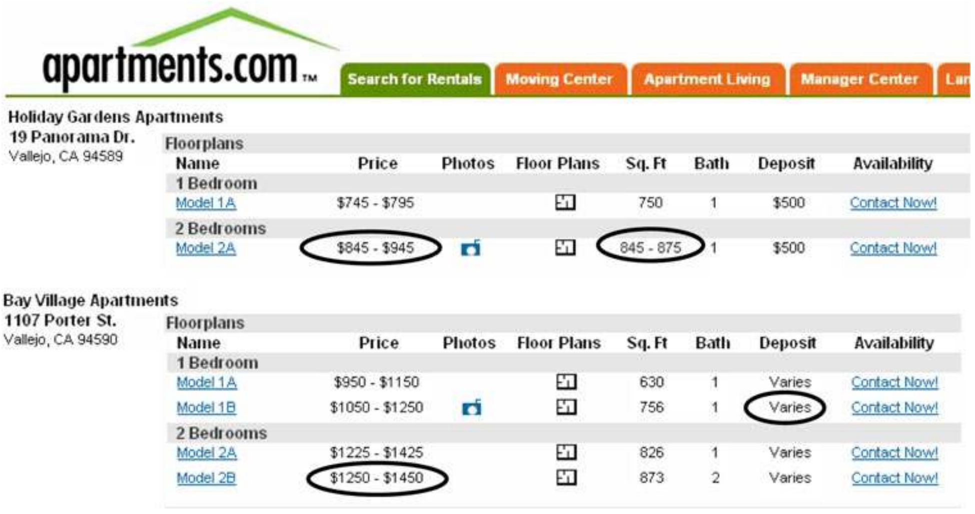
Top-k (ranking) queries report the k records with the highest scores in query output, based on a scoring function defined on one or more scoring predicates (e.g., functions defined on one or more database columns). A scoring function induces a *total order* over records with different scores (ties are 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 rankings are valid. Studying the formulation and processing of top-k queries in this context is lacking in the current proposals.

1.1 Motivation and challenges

Consider Fig. 1 which shows a snapshot of actual search results reported by `apartments.com` for a simple search

Fig. 1 Uncertain data in search results



apartments.com™ | Search for Rentals | Moving Center | Apartment Living | Manager Center | Lar

Holiday Gardens Apartments
19 Panorama Dr.
Vallejo, CA 94589

Floorplans

Name	Price	Photos	Floor Plans	Sq. Ft	Bath	Deposit	Availability
1 Bedroom							
Model 1A	$745 - $795			750	1	$500	Contact Now!
2 Bedrooms							
Model 2A	$845 - $945			845 - 875	1	$500	Contact Now!

Bay Village Apartments
1107 Porter St.
Vallejo, CA 94590

Floorplans

Name	Price	Photos	Floor Plans	Sq. Ft	Bath	Deposit	Availability
1 Bedroom							
Model 1A	$950 - $1150			630	1	Varies	Contact Now!
Model 1B	$1050 - $1250			756	1	Varies	Contact Now!
2 Bedrooms							
Model 2A	$1225 - $1425			826	1	Varies	Contact Now!
Model 2B	$1250 - $1450			873	2	Varies	Contact Now!

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 Fig. 1 may originate from different sources including the following: (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 scraped 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 Fig. 1.

Example 1 Assume an apartment database. Figure 2a gives a snapshot of the results of some user query posed against such database. Assume that the user would like to rank 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 $a4$.

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

(a)

$$\begin{tikzcd} & a1 \arrow[dl] \arrow[dr] & & a4 \\ a2 \arrow[dr] & & a3 \arrow[dl] & \\ & a5 & & \end{tikzcd}$$

(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

Figure 2b depicts a diagram for the partial order induced by apartment scores (we formally define partial orders in Sect. 2.1). 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 compute a ranking based on 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. These rankings can be unreliable in some cases. For example, assume 3 apartments, $a1$, $a2$, and $a3$ with score intervals [0, 100], [40, 60], and [30, 70], respectively. Assume that score values are distributed uniformly within each interval. The expected score of each apartment is thus 50, and hence all apartment permutations are equally likely rankings. However, based on how the score intervals intersect, we show in Sect. 4

[1] Imputation methods [6,7] can give better guesses for missing values. We study the effect of using these methods in Sect. 7.

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 when dealing 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 size of 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 2c shows all linear extensions of the partial order in Fig. 2b. 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 $a1$ appears at 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 need 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.

1.2 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:

– We introduce a novel probabilistic ranking model based on partial orders (Sect. 2.1).
– We formulate the problem of ranking under score uncertainty by introducing new semantics of ranking queries that can be adopted in different application scenarios (Sect. 2.2).
– We introduce a space pruning algorithm to cut down the answer space, allowing efficient query evaluation to be conducted subsequently (Sect. 6.1).
– We introduce a set of query evaluation techniques:

  1. We show that exact query evaluation is expensive for some of our proposed queries (Sect. 6.3).
  2. We give branch-and-bound search algorithms to compute exact query answers based on A^* search. The search algorithms lazily explore the space of possible answers, and early-prune partial answers that do not lead to final query answers (Sect. 6.4.1).
  3. We propose novel sampling techniques based on a Markov Chain Monte-Carlo (MCMC) method to compute approximate query answers (Sect. 6.4.2).

– We study the novel problem of optimal rank aggregation in partial orders induced by uncertain scores under both *Spearman footrule* and *Kendall tau* distance metrics (Sect. 6.5):

  1. We give a polynomial time algorithm to solve the problem under Spearman footrule distance (Sect. 6.5.1).
  2. We thoroughly study the problem of rank aggregation in partial orders induced by uncertain scores under Kendall tau distance. While the problem is NP-Hard in general [13], we identify key properties that define different classes of partial orders in which computing the optimal rank aggregation has polynomial time cost. We give the

corresponding query processing algorithms, and provide a detailed complexity analysis (Sect. 6.5.2).

– We give new methods to construct probability density functions of records' scores based on uncertain and incomplete attribute values. Our methods leverage kernel density estimation and attribute correlations discovery techniques to compute and aggregate uncertain scores from multiple scoring attributes (Sect. 7).

We also conduct an extensive experimental study using real and synthetic data to examine the robustness and efficiency of our techniques in various settings (Sect. 8).

2 Data model and problem definition

In this section, we describe the data model we adopt in this paper (Sect. 2.1), followed by our problem definition (Sect. 2.2). We define the notations we use throughout this paper in Table 1.

2.1 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 function f_i can be obtained directly from uncertain attributes (e.g., a uniform distribution on possible apartment's rent values as in Fig. 1). Alternatively, f_i 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 [14]. 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 (i.e., $[(t_i > t_j) \wedge (t_j > t_k)] \Rightarrow (t_i > t_k)$).

Table 1 Frequently used notations

Symbol	Description
D	Database with uncertain scores
t_i	A record with uncertain score
$[lo_i, up_i]$	Score interval of t_i
f_i	Score density function of t_i
$\acute{D}$	Database after pruning k-dominated records
PPO	Probabilistic partial order
ω	A linear extension of a PPO
v_x	A linear extension prefix of length x
s_x	A set of x records
$\lambda_{(i,j)}(t)$	Probability of t appearing at a rank in $[i, j]$

Table 2 Modeling score uncertainty

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]	$f_6 = 1$

We assume in the next discussion that the score intervals and density functions are given. In Sect. 7, we give general techniques to construct these components from uncertain attributes, as well as missing and incomplete attributes.

Table 2 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., Fig. 2b), which is a directed acyclic graph the nodes of which are the elements of $\mathcal{R}$, and edges are the binary relationships in $\mathcal{O}$, except relationships derived by transitivity. An edge (i, j) indicates that i is ranked above 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 assumed to be equally likely.

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

Definition 2 [*Score 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

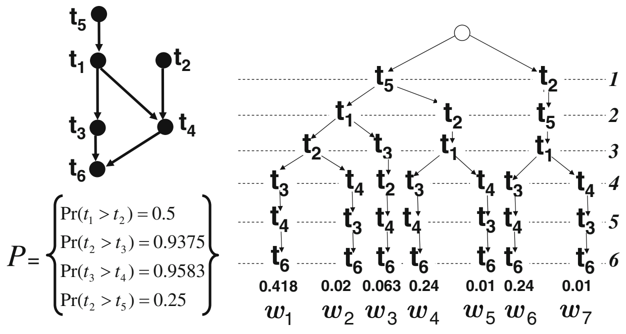


Fig. 3 Probabilistic partial order and linear extensions

scores. Based on Definition 2, Property 1 immediately follows:

Property 1 Score 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 above record t_j, denoted $\Pr(t_i > t_j)$, is given by the following two-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 $(0, 1)$, and $\Pr(t_j > t_i) = 1 - \Pr(t_i > t_j)$. On the other hand, if t_i dominates t_j, then we have $\Pr(t_i > t_j) = 1$ and $P(t_j > t_i) = 0$.

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 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 above 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 3 shows the Hasse diagram and the probabilistic dominance relation of the PPO of records in Table 2. We also show the set of linear extensions of the PPO.

The linear extensions of 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 Sect. 5.

Due to probabilistic dominance, the space of possible linear extensions is viewed as 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 Fig. 3, the probability value associated with each linear extension. We show how to compute these probabilities in Sect. 4.

2.2 Problem definition

Based on the data model in Sect. 2.1, we consider three classes of ranking queries:

RECORD- RANK QUERIES. Queries that report 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 $i \ldots j$ (i.e., from i to j inclusive) in possible linear extensions. That is, for a linear extensions space Ω of a PPO, the query UTop-Rank(i, j), 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 $i, \ldots, j$.

For example, in Fig. 3, the query UTop-Rank(1, 2) reports t_5 with probability $\Pr(\omega_1) + \cdots + \Pr(\omega_7) = 1.0$, since t_5 appears at all linear extensions at either rank 1 or rank 2.

TOP- k- QUERIES. Queries that report a group 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, the query UTop-Prefix(k) reports $argmax_p(\sum_{\omega \in \Omega_{(p,k)}} \Pr(\omega))$, where $\Omega_{(p,k)} \subseteq \Omega$ is the set of linear extensions having p as the k-length prefix.

For example, in Fig. 3, the query UTop-Prefix(3) 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, the query UTop-Set(k) reports $argmax_s(\sum_{\omega \in \Omega_{(s,k)}} \Pr(\omega))$, where $\Omega_{(s,k)} \subseteq \Omega$ is the set of linear extensions having s as the set of top-k records.

For example, in Fig. 3, the query UTop-Set(3) 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 $i, \ldots, j$, 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.

RANK- AGGREGATION- QUERIES. Queries that report 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 show in Sect. 6.5 that this query can be mapped to a UTop-Rank query under a specific definition of distance measure. We also derive a correspondence between this query definition and the ranking query that orders records on their expected scores

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 contain that answer. These semantics are analogous to possible worlds semantics in probabilistic databases [15, 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 UTop-Rank$(1, k)$ probability of a record t is the summation of the UTop-Rank(i, i) probabilities of t for $i = 1, \ldots, k$.

Similar query definitions are used in [16–18], 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 (Sect. 2.1) is fundamentally different, which entails different query processing techniques. Furthermore, to the best of our knowledge, UTop-Set query as well as Rank-Agg query in partial orders have not been proposed before.

2.2.1 Example applications

Our proposed query types can be adopted in the following application examples:

– 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 above all other products.
– Rank aggregation query is widely adopted in many applications related to combining votes from different voters to rank a given set of candidates in a way that minimizes the disagreements of voter's opinions. A typical application of rank aggregation queries is building a meta-search engine (a search engine that aggregates the rankings of multiple other engines) as discussed in [13], and described in more detail in Sect. 6.5. An example application of a Rank-Agg query in our settings is finding a consensus ranking for a set of candidates, where each candidate receives a numeric score from each voter, which can be compactly encoded as a PPO. We give more details in Sect. 6.5.

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 Sect. 5. Our goal is to design efficient algorithms that overcome such prohibitive computational barrier.

3 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.

3.1 Monte-Carlo integration

The method of Monte-Carlo integration [19] computes accurate estimate of the integral $\int_{\acute{\Gamma}} f(x)\mathrm{d}x$, 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,

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

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

3.2 Markov chains

We give a brief description for the theory of Markov chains. We refer the reader to [20] 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:

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

3.3 Markov chain Monte-Carlo (MCMC) method

The concepts of Monte-Carlo method and Markov chains are combined in the MCMC method [20] 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 [21]. 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 $\alpha = \min\left(\frac{\pi(x_1).q(x_0|x_1)}{\pi(x_0).q(x_1|x_0)}, 1\right)$.
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 (Eq. 3) with arbitrary proposal distributions [21]. 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 (e.g., average) or estimate a function of interest on π.

4 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:

$$\Pr(\omega) = \int\limits_{lo_1}^{up_1} \int\limits_{lo_2}^{x_1}, \dots, \int\limits_{lo_n}^{x_{n-1}} f_1(x_1), \dots, f_n(x_n)\mathrm{d}x_n, \dots, \mathrm{d}x_1 \tag{4}$$

Monte-Carlo integration (Sect. 3) can be used to compute complex nested integrals, such as Eq. 4. For example, the probabilities of linear extensions $\omega_1, \dots, \omega_7$ in Fig. 3 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* 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 the relative order of which in ω is $t_i > t_j$, while $lo_j \geq up_i$. However, this contradicts the definition of

Algorithm 1 Build linear extension tree

```
BUILD_TREE (PPO($\mathcal{R}, \mathcal{O}, \mathcal{P}$) : $PPO, n$ : $Tree\ node$)
1  for each source $t \in \mathcal{R}$
2      do
3          $child \leftarrow$ create a tree node for $t$
4          Add $child$ to $n$'s children
5          $\acute{PPO} \leftarrow$ PPO($\mathcal{R}, \mathcal{O}, \mathcal{P}$) after removing $t$
6          BUILD_TREE( $\acute{PPO}$, $child$)
```

$\mathcal{O}$ in 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 $\omega = \langle t_1, \ldots, t_n\rangle$ to its corresponding event $e = ((t_1 > t_2) \wedge \cdots \wedge (t_{n-1} > t_n))$. Equation 4 computes $\Pr(e)$ or equivalently $\Pr(\omega)$. Second, let ω_1 and ω_2 be two linear extensions in Ω the events of which 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., $\Pr(e_1 \vee e_2 \cdots \vee e_m) = 1$, where $m = |\Omega|$). Since all e_i's are mutually exclusive, it follows that $\Pr(e_1 \vee e_2 \cdots \vee e_m) = \Pr(e_1) + \cdots + \Pr(e_m) = \sum_{\omega \in \Omega} \Pr(\omega) = 1$, and hence Eq. 4 defines a probability distribution on Ω. □

5 A baseline exact algorithm

We describe a baseline algorithm that computes the queries in Sect. 2.2 by materializing the linear extensions space. Algorithm 1 gives a simple recursive technique to build the linear extensions tree (Sect. 2.1). The first call to Procedure BUILD_TREE is passed the parameters 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 PPO($\mathcal{R}, \mathcal{O}, \mathcal{P}$), and then recurse by finding new sources in PPO($\mathcal{R}, \mathcal{O}, \mathcal{P}$) after removing t.

The space of all linear extensions of 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.

When we are interested only 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 $\omega^{(k)} = \langle t_1, t_2, \ldots, t_k\rangle$ 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 above 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, $\Pr(\omega^{(k)}) = \Pr((t_1 > t_2), \ldots, (t_{k-1} > t_k), (t_k > T(\omega^{(k)})))$, where

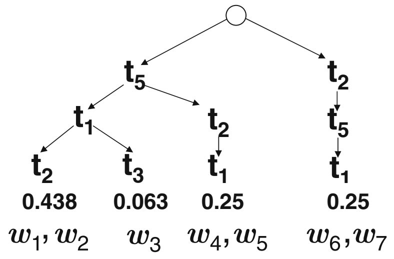


Fig. 4 Prefixes of linear extensions at depth 3

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

Hence, we have

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

Figure 4 shows the prefixes of length 3 and their probabilities for the linear extensions tree in Fig. 3. 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.

5.1 Algorithm BASELINE

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 (Eq. 6), and reporting the prefixes with the highest probabilities. We can use these probabilities to answer UTop-Rank query for ranks $1, \ldots, k$, 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 Sect. 2.2.

6 Query evaluation

The BASELINE algorithm described in Sect. 5 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 Sect. 6.1, we answer the first question by using indexes to prune records that do not contribute to query answers, while in Sects. 6.3 and 6.4, we answer the second question by exploiting query semantics for faster computation.

6.1 k-Dominance: shrinking the database

Given a database D conforming to our score uncertainty model, we call a record $t \in D$ "k-dominated" if at least k other records in D dominate t. For example in Fig. 3, 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 (Sect. 2.1).

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.

Algorithm 2 Remove k-dominated records

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

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

6.1.1 Complexity analysis

Since Algorithm 2 conducts a binary search on U, its worst case complexity is in $O(log(m))$, where $m = |D|$. The list 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 is thus $O(m\,log(m))$, 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.

6.2 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 (Sect. 5) with simpler Monte-Carlo integration exploiting the query semantics to avoid enumerating the probability space.

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 $\gamma = (x_1, x_2, \cdots, x_n)$ 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. Records with deterministic (single-valued) scores are represented by the same score value in all possible γ's. On the other hand, records with uncertain scores can be represented by different score values in different γ's according to the intervals that enclose their possible scores.

In case of a continuous f_i, the component x_i is assumed to be a tiny score interval in $[lo_i, up_i]$, and $f_i(x_i)$ is the result of integrating f_i over x_i. We assume that the components x_i's of any possible vector $\gamma = (x_1, x_2, \ldots, x_n)$ can always be totally ordered based on their values.

6.3 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 Fig. 3, 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 Sect. 6.1 does not affect record ranks smaller than k, since any k-dominated record appears only at ranks $> k$.

Hence, given a range of ranks $i, \ldots, j$, 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. Sect. 3). We assume that the number of samples is chosen such that the error (which is in $O\left(\frac{1}{\sqrt{s}}\right)$) is tolerated. We experimentally verify in Sect. 8 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 $i \ldots j$, accept γ.

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

$$\lambda_{(i,j)}(t_k) = \int\limits_{\Gamma_{(i,j,t_k)}} \Pi_{\acute{D}}(\gamma)\, d\gamma \qquad (7)$$

where $\Gamma_{(i,j,t_k)} \subseteq \Gamma$ is the volume defined by the points $\gamma = (x_1, \ldots, x_n)$, with x_k's rank is in $i, \ldots, j$. The integral in Eq. 7 is evaluated as discussed in Sect. 3.

6.3.1 Complexity analysis

Let s be the total number of samples drawn from Γ to evaluate Eq. 7. In order to compute the l most probable records to appear at a rank in $i \ldots j$, we need to apply Eq. 7 to each record in $\acute{D}$ the rank interval of which 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 $\acute{D}$ the rank intervals of which intersect $[i, j]$. In the worst case, $n_{(i,j)} = n$.

6.4 Computing Top- k- Queries

Let v be a linear extension prefix of k records, and s be a set of k records. We denote with $\Pr(v)$ the top-k prefix probability of v and, similarly, we denote with $\Pr(s)$ the top-k set probability of s. Similar to our discussion of UTop-Rank queries in Sect. 6.3, $\Pr(v)$ is computed using Monte-Carlo integration on the volume $\Gamma_{(v)} \subseteq \Gamma$ which consists of the points $\gamma = (x_1, \ldots, x_n)$ 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, $\Pr(s)$ is computed by integrating on the volume $\Gamma_{(u)} \subseteq \Gamma$ which consists of the points $\gamma = (x_1, \ldots, x_n)$ such that any value in γ, that does not correspond to a record in s, is smaller than the minimum value that corresponds to a record in s.

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

The cost of the above integrals is similar to the cost of the integral in Eq. 7 (mainly proportional to the number of samples). 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).

In the following, we describe a branch-and-bound search algorithm to compute exact query answers (Sect. 6.4.1). We also describe sampling techniques, based on the (M–H) algorithm (cf. Sect. 3), to compute approximate query answers at a lower computational cost (Sect. 6.4.2).

6.4.1 A branch-and-bound algorithm

Our branch-and-bound algorithm employs a systematic method to enumerate all possible candidate solutions (i.e., possible top-k prefixes/sets), while discarding a large subset of these solutions by upper-bounding the probability of unexplored candidates. We discuss our algorithm by describing how candidates are generated, and how candidate pruning is conducted. We conclude our discussion by giving the overall branch-and-bound algorithm. For clarity of presentation, we focus our discussion on the evaluation of UTop-Prefix queries. We show how to extend the algorithm to evaluate UTop-Set queries at the end of this section.

Candidate generation. Based on our discussion in Sect. 6.3, the rank intervals of different records can be derived from the score dominance relationships in the underlying PPO. Using the rank intervals of different records, we can incrementally generate candidate top-k prefixes by selecting a distinct record $t_{(i)}$ for each rank $i = 1 \dots k$ such that the rank interval of $t_{(i)}$ encloses i, and the selected records at different ranks form together a valid top-k prefix (i.e., a prefix of at least one linear extension of the underlying PPO). A top-k prefix v is valid if for each record $t_{(i)} \in v$, all records dominating $t_{(i)}$ appear in v at ranks smaller than i. For example in Fig. 3, the set of records that appear at ranks 1 and 2 are $\{t_5, t_2\}$ and $\{t_1, t_2, t_5\}$, respectively. The top-2 prefix $\langle t_2, t_1 \rangle$ is invalid since the record t_5, that dominates t_1, is not selected at rank 1. On the other hand, the top-2 prefix $\langle t_5, t_1 \rangle$ is valid since t_1 can be ranked after t_5.

Candidate pruning. Pruning unexplored candidates is mainly done based on the following property (Property 2). We use subscripts to denote prefixes' lengths (e.g., v_x is a top-x prefix).

Property 2 Let v_x be a top-x prefix and v_y be a top-y prefix, where $v_x \subseteq v_y$. Then, $\Pr(v_y) \leq \Pr(v_x)$.

The correctness of Property 2 follows from an implication of the definition of our probability space: the set of linear extensions prefixed by v_x includes all linear extensions prefixed by v_y. Since the probability of a prefix v_l is the summation of all linear extensions prefixed by v_l, Property 2 follows.

Hence, given a top-k prefix v_k, any top-x prefix v_x with $x \leq k$ and $\Pr(v_x) < \Pr(v_k)$ can be safely pruned from the candidates set since $\Pr(v_x)$ upper-bounds the probability of any top-k prefix $\acute{v}_k$ where $v_x \subseteq \acute{v}_k$.

Algorithm 3 Branch-and-bound UTop-Prefix query evaluation

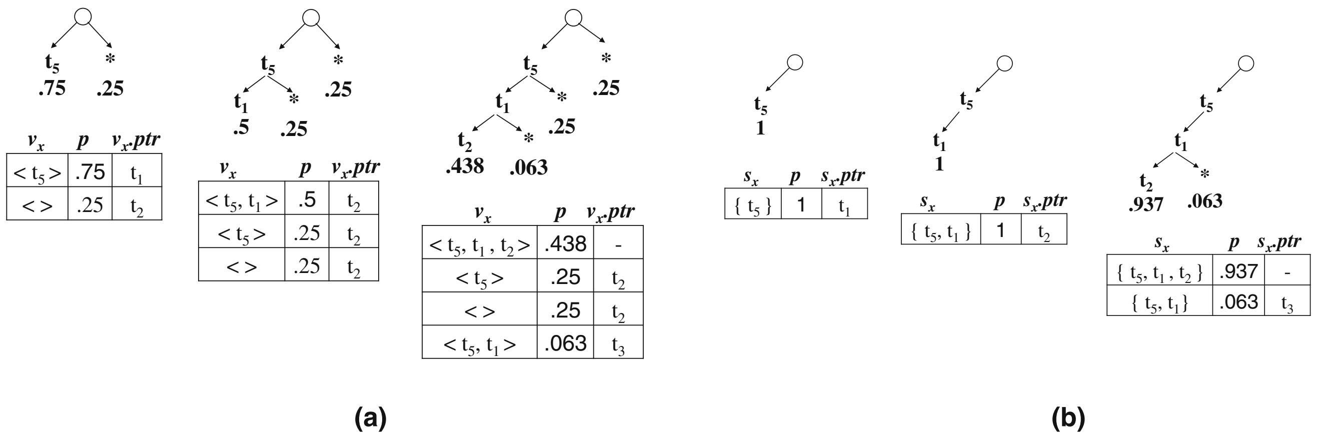
```
BB-UTop-Prefix ($D$ : $database$, $k$ : $answer\ size$)
1  {Initialization Phase}
2  $U \leftarrow$ score upper-bound list
3  $\acute{D} \leftarrow$ Shrink_DB$(D, k, U)$ {cf. Sect. 6.1}
4  for $i = 1$ to $k$
5      do
6          Compute $\lambda_{(i,i)}$ based on $\acute{D}$ {cf. Sect. 6.3}
7          $L_i \leftarrow$ sort tuples in $\lambda_{(i,i)}$ in a descending prob. order
8  {Searching Phase}
9  $\mathcal{Q} \leftarrow$ a priority queue of prefixes ordered on probability
10 $v_0 \leftarrow$ an empty prefix with probability 1
11 $v_0 \,.\, ptr \leftarrow 0$ {first position in $L_1$}
12 Insert $v_0$ into $\mathcal{Q}$
13 while ($\mathcal{Q}$ is not empty)
14     do
15         $v_x^* \leftarrow$ evict top prefix in $\mathcal{Q}$
16         if $(x = k)$
17            then {reached query answer}
18                 return $v_x^*$
19         $t^* \leftarrow$ first tuple in $L_{x+1}$ at position $pos^* \geq v_x^*.ptr$
               s.t. $\langle v_x^*, t^* \rangle$ is a valid prefix
20         $v_x^*.ptr \leftarrow pos^* + 1$
21         $v_{x+1} \leftarrow \langle v_x^*, t^* \rangle$
22         Compute $\Pr(v_{x+1})$
23         if $(x + 1 = k)$
24            then
25                 Prune all prefixes in $\mathcal{Q}$ with prob. $< \Pr(v_{x+1})$
26            else
27                 $v_{x+1}.ptr \leftarrow 0$ {first position in $L_{x+2}$}
28         if $(v_x^*.ptr < |L_{x+1}|)$
29            then {$v_x^*$ can be further extended}
30                 $\Pr(v_x^*) \leftarrow \Pr(v_x^*) - \Pr(v_{x+1})$
31                 Insert $v_x^*$ into $\mathcal{Q}$
32         Insert $v_{x+1}$ into $\mathcal{Q}$
```

The overall search algorithm. The details of the branch-and-bound search algorithm are given in Algorithm 3. The algorithm works in the two following phases:

– An *initialization phase* that builds and populates the data structures necessary for conducting the search.
– A *searching phase* that applies greedy search heuristics to lazily explore the answer space and prune all candidates that do not lead to query answers.

In the initialization phase, the algorithm reduces the size of the input database, based on the parameter k, by invoking the shrinking algorithm discussed in Sect. 6.1. The techniques described in Sect. 6.3 are then used to compute the distribution $\lambda_{(i,i)}$ for $i = 1 \dots k$. The algorithm maintains k lists $L_1 \dots L_k$ such that list L_i sorts tuples in $\lambda_{(i,i)}$ in a descending probability order.

In the searching phase, the algorithm maintains a priority queue $\mathcal{Q}$ that maintains generated candidates in descending order of probability. The priority queue is initialized with an empty prefix v_0 of length 0 and probability 1. Each maintained candidate v_x of length $x < k$ keeps a pointer $v_x \,.\, ptr$



v_x	p	$v_x.ptr$
$<t_5>$	.75	t_1
$<>$	.25	t_2

v_x	p	$v_x.ptr$
$<t_5, t_1>$	.5	t_2
$<t_5>$	.25	t_2
$<>$	.25	t_2

v_x	p	$v_x.ptr$
$<t_5, t_1, t_2>$	.438	-
$<t_5>$	.25	t_2
$<>$	.25	t_2
$<t_5, t_1>$	.063	t_3

s_x	p	$s_x.ptr$
$\{t_5\}$	1	t_1

s_x	p	$s_x.ptr$
$\{t_5, t_1\}$	1	t_2

s_x	p	$s_x.ptr$
$\{t_5, t_1, t_2\}$	.937	-
$\{t_5, t_1\}$	.063	t_3

Fig. 5 Computing Top- k- Queries using branch-and-bound. **a** Evaluating UTop-Prefix(3) query; **b** evaluating UTop-Set(3) query

pointing at the position of the next tuple in the list L_{x+1} to be used in extending v_x into a candidate of length $x+1$. Initially, $v_x . ptr$ is set to the first position in L_{x+1}. The positions are assumed to be 0-based. Hence, the value of $v_x . ptr$ ranges between 0 and $|L_{x+1}| - 1$.

Extending candidates is done slowly (i.e., one candidate is extended at a time). Following the greedy criteria of A* search, the algorithm selects the next candidate to extend as follows. At each iteration, the algorithm evicts the candidate v_x^* at the top of $\mathcal{Q}$ (i.e., $\Pr(v_x^*)$ is the highest probability in $\mathcal{Q}$). If $x = k$, the algorithm reports v_x^* as the query answer. Otherwise, if $x < k$, the algorithm extends v_x^* into a new candidate v_{x+1} by augmenting v_x^* with the tuple t^* at the first position $\geq v_x^* . ptr$ in L_{x+1} such that $v_{x+1} = \langle v_x^*, t^* \rangle$ is a valid prefix. The pointer $v_x^* . ptr$ is set to the position right after the position of t^* in L_{x+1}, while the pointer $v_{x+1} . ptr$ is set to the first position in L_{x+2} (only if $x + 1 < k$). The probabilities of v_{x+1} and v_x^* are computed ($\Pr(v_x^*)$ is reduced to $\Pr(v_x^*) - \Pr(v_{x+1})$) and the two candidates are reinserted in $\mathcal{Q}$. Furthermore, if $x + 1 = k$ (line 23), the algorithm prunes all candidates in $\mathcal{Q}$ the probabilities of which are less than $\Pr(v_{x+1})$ according to Property 2. In addition, if $v_x^* . ptr > |L_{x+1}|$, then v_x^* cannot be further extended into candidates of larger length, and so v_x^* is removed from $\mathcal{Q}$.

The correctness of Algorithm 3 follows from the correctness of our systematic candidate generation method, and the correctness of our probability upper bounding method described in the beginning of this section.

Figure 5 gives an example illustrating how Algorithm 3 works. We use the PPO in Fig. 3 in this example, where the ordered tuples list $L_1 = \langle t_5, t_2 \rangle$, $L_2 = \langle t_1, t_2, t_5 \rangle$, and $L_3 = \langle t_1, t_2, t_3 \rangle$. Figure 5a shows how the branch-and-bound algorithm computes for the answer of a UTop-Prefix(3) query. The search starts with an empty prefix v_0 with probability 1. The prefix v_0 is extended using t_5 (the first tuple in L_1). The algorithm then computes $\Pr(\langle t_5 \rangle)$ as 0.75 (the probability is computed using Monte-Carlo integration as discussed in the beginning of Sect. 6.4) , while $\Pr(v_0)$ decreases by 0.75. Both prefixes are inserted into $\mathcal{Q}$ after updating their ptr fields to point to the next tuple that can be used to create valid prefixes later. After three steps, the search terminates since the top prefix in $\mathcal{Q}$ has length 3.

Computing UTop-Set queries by branch-and-bound. The branch-and-bound prefix search algorithm can be easily extended to compute UTop-Set queries. The reason is that Property 2 also holds on sets. That is, let s_x and s_y be two record sets with sizes x and y, respectively. Then, if $s_x \subseteq s_y$, we have $\Pr(s_y) \leq \Pr(s_x)$. Hence, $\Pr(s_y)$ upper-bounds the probability of any set that can be created by appending more tuples to s_y.

The main difference between prefix search and set search is that multiple prefixes map to the same set. For example, both prefixes $\langle t_2, t_5 \rangle$ and $\langle t_5, t_2 \rangle$ map to the set $\{t_2, t_5\}$. We thus need to filter out prefixes that map to already instantiated sets. This is done by maintaining an additional hash table of instantiated sets. Each generated candidate is first looked up in the hash table, and a new set is instantiated only if the hash table does contain a corresponding set.

Figure 5b shows how the branch-and-bound algorithm computes for the answer of a UTop-Set(3) query. The search starts by instantiating an empty set s_0 with probability 1. The set s_0 is extended using t_5 (the first tuple in L_1), which results in having $\Pr(\{t_5\}) = 1$ (i.e., t_5 appears in all linear extensions at ranks 1 , . . . , 3), and hence $\Pr(s_0)$ is set to 0, and can thus be removed from $\mathcal{Q}$. After three steps, the search terminates since the top set in $\mathcal{Q}$ has size 3.

6.4.2 A sampling-based algorithm

In this section we describe a sampling-based algorithm to compute approximate answers of Top- k- Queries.

Sampling space. A state in our space is a linear extension ω of the PPO induced by $\acute{D}$. Let θ and Θ be the distributions of the top-k prefix probabilities and top-k set probabilities, respectively. 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 (Eq. 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 $j = 1, \ldots, z$ 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 [21]. 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 $R_{(r_j,r)} = \{r_j - 1, r_j - 2, \ldots, r\}$. 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 $\alpha = \min\left(\frac{\pi(\omega_{i+1}).q(\omega_i|\omega_{i+1})}{\pi(\omega_i).q(\omega_{i+1}|\omega_i)}, 1\right)$.

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 [22] 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 [22], a widely used statistic in evaluating the convergence of multiple independent Markov chains [23]. 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 $e = ((t_{(1)}$ ranked $1^{st}) \wedge \cdots \wedge (t_{(k)}$ 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 Sect. 6.3. 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 Sect. 8 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 Sect. 9 on the applicability of our techniques to other error estimation methods.

Caching. Our sample generator mainly uses two-dimensional integrals (Eq. 1) to bias generating a sample by its probability. Such two-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.

6.5 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 [13].

While our work is mainly concerned with ranking under possible worlds semantics (Sect. 2.2), 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 the set of candidates $\mathcal{C}$ 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. Two widely used measures of the distance between two rankings are the Spearman footrule distance and the Kendall tau distance.

The Spearman footrule distance is the summation, over all candidates, of the distance between the positions of the same candidate in the two lists, formally defined as follows:

$$\mathrm{F}(\omega_\mathrm{i}, \omega_\mathrm{j}) = \sum_{\mathrm{c} \in \mathcal{C}} |\omega_\mathrm{i}(\mathrm{c}) - \omega_\mathrm{j}(\mathrm{c})| \tag{8}$$

On the other hand, the Kendall tau distance is the number of pairwise disagreements in the relative order of candidates in the two lists, formally defined as follows:

$$\begin{aligned}\mathrm{K}(\omega_i, \omega_j) = &|\{(c_a, c_b) : a < b,\ \omega_i(c_a) < \omega_i(c_b),\\ &\omega_j(c_a) > \omega_j(c_b)\}|\end{aligned} \tag{9}$$

The optimal rank aggregation is the ranking with the minimum average distance to all input rankings. It is well-known that optimal rank aggregation under Kendall tau distance (also known as Kemeny-optimal aggregation) is the only aggregation that satisfies the following intuitive properties [13,24]:

- Neutrality: if two candidates switch their positions in all input rankings, then their positions must be switched in the aggregate ranking.
- Consistency: if the set of input rankings is split into two sets A and B, such that the aggregate rankings of both A and B prefer candidate c_1 to candidate c_2, then the overall aggregate ranking must also prefer c_1 to c_2.
- Extended Condorcet criterion: for two candidate sets $\mathcal{C}_1$ and $\mathcal{C}_2$, if for every $c_i \in \mathcal{C}_1$ and $c_j \in \mathcal{C}_2$, the majority of input rankings prefer c_i to c_j, then the aggregate ranking must prefer $\mathcal{C}_1$ to $\mathcal{C}_2$.

Unfortunately, rank aggregation under Kendall tau distance is NP-Hard in general. The optimal aggregation under Spearman footrule distance is a 2-approximation of the Kendall tau aggregation [13,24].

In the following sections we discuss evaluating Rank- Aggregation- Queries, based on our probabilistic partial order model, under each of the Spearman footrule distance (Sect. 6.5.1) and the Kendall tau distance (Sect. 6.5.2).

6.5.1 Rank- Aggregation- Queries *with Spearman footrule distance*

Optimal rank aggregation under footrule distance can be computed in polynomial time by the following algorithm [13]. Given a set of rankings $\omega_1, \ldots, \omega_m$, the objective is to find the optimal ranking ω^* that minimizes $\frac{1}{m}\sum_{i=1}^{m} \mathrm{F}(\omega^*, \omega_i)$. 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) the weight of which is $w(c, r) = \sum_{i=1}^{m} |\omega_i(c) - r|$. Then, ω^* (the optimal ranking) 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 [13].

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* 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

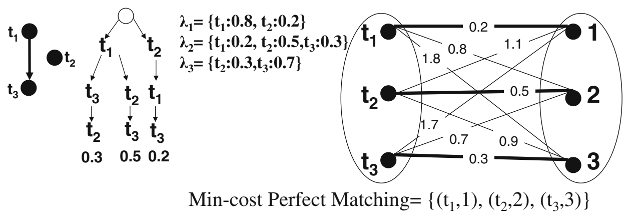


Fig. 6 Bipartite graph matching

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 $\frac{w(t,r)}{|\acute{\Omega}|} = \sum_{j=1}^{n}\left(\frac{n_j}{|\acute{\Omega}|} \times |j-r|\right) = \sum_{j=1}^{n}(\lambda_j(t) \times |j-r|)$. 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 weights of graph edge without expanding the space of linear extensions. The distributions λ_i's are obtained by applying Eq. 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 (Sect. 6.3). 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.

6.5.2 Rank- Aggregation- Queries *with Kendall tau distance*

Optimal rank aggregation under Kendall tau distance is known to be NP-Hard in general by reduction to the problem of minimum feedback arc set [24]: construct a complete weighted directed graph the nodes of which are the candidates, such that an edge connecting nodes c_i and c_j is weighted by the proportion of voters who rank c_i before c_j. The problem is to find the set of edges with the minimum weight summation the removal of which converts the input graph to a DAG. Since the input graph is complete, the resulting DAG defines a total order on the set of candidates, which is the optimal rank aggregation.

The hardness of the rank aggregation problem gives rise to approximation methods similar to the Markov chains-based methods in [13] to find the optimal rank aggregation. Spearman footrule aggregation is also known to be a 2-approximation of Kendall tau aggregation [24].

However, under our settings, we identify key properties that influence the hardness of computing optimal Kendall tau rank aggregation. We show that optimal rank aggregation can be computed in polynomial time depending on the properties of the underlying PPO, summarized as follows:

1. If the PPO is induced by records with non-uniform score densities, and the PPO is weak stochastic transitive (see Definition 9 below), then query computation cost is polynomial in n (the database size).
2. If the PPO is induced by records with uniform score densities, then the PPO is guaranteed to be weak stochastic transitive, and a polynomial time algorithm to compute Kendall tau aggregation exists. Moreover, by exploiting score uniformity, the complexity can be further reduced to $O(nlog(n))$.

We start our discussion by defining the property of weak stochastic transitivity in the context of probabilistic partial orders.

Definition 9 [*Weak Stochastic Transitivity*] A PPO induced by a database D is weak stochastic transitive iff $\forall$ records $x, y, z \in D$: $[\Pr(x > y) \geq 0.5$ and $\Pr(y > z) \geq 0.5] \Rightarrow \Pr(x > z) \geq 0.5$.

The property of weak stochastic transitivity is formulated and used in many probabilistic preference models. We refer the reader to [25] for a detailed discussion. We briefly contrast our interpretation of probabilistic preference against current interpretations in the following.

In many probabilistic preference models [25–27], for a pair of alternatives x and y, $\Pr(x > y)$ is interpreted as the probability that x is chosen over y. The origin of such probabilistic preferences can be related to changes in the internal state of the selecting agent (e.g., as a result of learning), to noise in the preferences obtained from users, or to the process of condensing users' votes into pairwise comparisons among candidates. In our settings, however, the origin of probabilistic preferences is the uncertainty in attribute values in the database, which in turn induces uncertainty in records' scores that we use for comparison and ranking. Our underlying probability space gives a concrete interpretation of $\Pr(x > y)$ as the summation of the probabilities of linear extensions (possible ranked instances of the database), where x is ranked above y.

Given an input PPO, the property of weak stochastic transitivity can be decided in $O(n^3)$, where n is the database size, since the property needs to be checked on record triples.

Rank- Aggregation- Queries *on a PPO with non-uniform score densities.* Let $\Omega = \{\omega_1, \ldots, \omega_m\}$ be the set of linear extensions of a PPO. The members of Ω represent voters associated with probabilistic weights. Hence, our objective is to find the optimal rank aggregation ω^* that minimizes $\frac{1}{m}\sum_{i=1}^{m}\Pr(\omega_i) \cdot \mathrm{K}(\omega^*, \omega_i)$.

Let $\Omega_{(t_i > t_j)} \subseteq \Omega$ be the set of linear extensions, where t_i is ranked above t_j. Then, $\Pr(t_i > t_j) = \sum_{\omega \in \Omega_{(t_i > t_j)}}\Pr(\omega)$. Hence, ω^* is the ranking that minimizes the probability summation of pairwise preferences violating the order given by

ω^*. That is, ω^* is the ranking that minimizes the following penalty function:

$$\mathrm{pen}(\omega) = \sum_{t_i, t_j \in D: i<j, \omega(t_j) < \omega(t_i)} \Pr(t_i > t_j) \quad (10)$$

If the property of weak stochastic transitivity holds on the underlying PPO, then ω^* can be efficiently computed based on Theorem 3:

Theorem 3 *Given a weak stochastic transitive PPO induced by a database D, the optimal rank aggregation ω^* under Kendall tau distance is defined as: $\forall$ records $x, y \in D$: $[\omega^*(x) < \omega^*(y)] \Leftrightarrow [\Pr(x > y) \geq 0.5]$ while breaking probability ties deterministically.*

Proof Since the underlying PPO is weak stochastic transitive, then ω^* is a valid ranking of D, since the definition of ω^* does not introduce cycles in the relative order of records in D.

Assume a rank aggregation $\acute{\omega}$ that is identical to ω^* except for the relative order of two records x and y. We consider the following three possible cases:

1. $[\Pr(x > y) = p > 0.5]$ In this case we have $\omega^*(x) < \omega^*(y)$ while $\acute{\omega}(x) > \acute{\omega}(y)$. Hence, $\mathrm{pen}(\omega^*) = \mathrm{pen}(\acute{\omega}) - (2p - 1)$.
2. $[\Pr(x > y) = p < 0.5]$ In this case we have $\omega^*(y) < \omega^*(x)$ while $\acute{\omega}(y) > \acute{\omega}(x)$. Hence, $\mathrm{pen}(\omega^*) = \mathrm{pen}(\acute{\omega}) - (1 - 2p)$.
3. $[\Pr(x > y) = p = 0.5]$ In this case assume that the deterministic tie-breaker $\tau(x, y)$ states that $(x > y)$. Then, $\omega^*(x) < \omega^*(y)$ while $\acute{\omega}(x) > \acute{\omega}(y)$. Hence, $\mathrm{pen}(\omega^*) = \mathrm{pen}(\acute{\omega})$. The same result also holds if $\tau(x, y)$ states that $(y > x)$.

Moreover, for any other rank aggregation $\acute{\acute{\omega}}$ that is different from ω^* in the relative order of more than two records, we have $\mathrm{pen}(\acute{\acute{\omega}}) \geq \mathrm{pen}(\acute{\omega}) \geq \mathrm{pen}(\omega^*)$. It follows that ω^* is the optimal rank aggregation. □

Query evaluation and complexity analysis. The result given by Theorem 3 allows for an efficient evaluation procedure to find the optimal rank aggregation in a weak stochastic transitive PPO. The procedure computes $\Pr(x > y)$ for each pair of records (x, y), and uses the computed probabilities to sort the database. That is, starting from an arbitrary ranking of records of D, the positions of any two records x and y need to be swapped iff $\Pr(x > y) \geq 0.5$ and x is ranked below y. Based on the weak stochastic transitivity of the PPO, this procedure yields a valid ranking of D since transitivity does not introduce cycles in the relative order of records. Hence, the overall complexity of the query evaluation procedure is $O(n^2)$, where $n = |D|$, which is the complexity of computing $\Pr(x > y)$ on each pair of records (x, y). If it is not a priori known if the property of weak stochastic transitivity holds on the PPO, then the overall complexity becomes $O(n^3)$ since the PPO needs to be checked for being weak stochastic transitive first.

RANK- AGGREGATION- QUERIES *on a PPO with uniform score densities.* If the records in the database that induces the PPO have uniform score densities, the cost of computing RANK- AGGREGATION- QUERIES drops considerably. We first prove in Theorem 4 below an important property that holds on the PPO induced by uniform score densities. In the following, we denote with $\mathrm{E}[f_i]$ the expected value of the score density f_i.

Theorem 4 *Given a PPO induced by records with uniform score densities in a database D, then $\forall$ records $t_i, t_j \in D$: $(\mathrm{E}[f_i] \geq \mathrm{E}[f_j]) \Leftrightarrow (\Pr(t_i > t_j) \geq 0.5)$.*

Proof First, we prove that $(\mathrm{E}[f_i] \geq \mathrm{E}[f_j]) \Rightarrow (\Pr(t_i > t_j) \geq 0.5)$. We first compute the integral that defines $\Pr(t_i > t_j)$ as follows. $\Pr(t_i > t_j) = \frac{1}{(up_i - lo_i) \times (up_j - lo_j)} \times \int_{lo_i}^{up_i} \int_{lo_j}^{x} dy dx$. By solving the integral we get $\Pr(t_i > t_j) = \frac{1}{up_j - lo_j} \times \left(\frac{up_i + lo_i}{2} - lo_j\right) = \frac{1}{up_j - lo_j} \times (\mathrm{E}(f_i) - lo_j)$. We rewrite the given $(\mathrm{E}[f_i] \geq \mathrm{E}[f_j])$ as $\mathrm{E}[f_i] = \mathrm{E}[f_j] + \epsilon$, where $\epsilon \geq 0$. Then, $\Pr(t_i > t_j) = \frac{1}{up_j - lo_j} \times ((\mathrm{E}(f_j) - lo_j) + \epsilon) = \frac{1}{2} + \frac{\epsilon}{up_j - lo_j}$, which means that $\Pr(t_i > t_j) \geq 0.5$.

Second, we prove that $(\Pr(t_i > t_j) \geq 0.5) \Rightarrow (\mathrm{E}[f_i] \geq \mathrm{E}[f_j])$. Assume that $\mathrm{E}[f_i] - \mathrm{E}[f_j] = \epsilon$, where ϵ is an arbitrary (positive/negative) real number. Since we have $\Pr(t_i > t_j) = \frac{1}{2} + \frac{\epsilon}{up_j - lo_j}$, and based on the given $(\Pr(t_i > t_j) \geq 0.5)$, we get $\frac{1}{2} + \frac{\epsilon}{up_j - lo_j} \geq \frac{1}{2}$, which means that $\epsilon \geq 0$. It follows that $\mathrm{E}[f_i] \geq \mathrm{E}[f_j]$, which concludes the proof. □

Based on Theorem 4, for records $t_i, t_j, t_k \in D$, if $\Pr(t_i > t_j) \geq 0.5$ and $\Pr(t_j > t_k) \geq 0.5$, then we have $\mathrm{E}[f_i] \geq \mathrm{E}[f_j]$ and $\mathrm{E}[f_j] \geq \mathrm{E}[f_k]$. It follows that $\mathrm{E}[f_i] \geq \mathrm{E}[f_k]$, which also means that $\Pr(t_i > t_k) \geq 0.5$. Hence, a PPO that is induced by uniform score densities is weak stochastic transitive.

Query evaluation and complexity analysis. Since the PPO is weak stochastic transitive, we do not need to conduct the transitivity checking step. We can compute RANK- AGGREGATION- QUERIES using the polynomial algorithm we described previously for weak stochastic transitive PPO's. However, based on Theorem 4, we can further optimize the computation cost. Specifically, for any two records t_i and t_j, we have $(\mathrm{E}[f_i] \geq \mathrm{E}[f_j]) \Leftrightarrow (\Pr(t_i > t_j) \geq 0.5)$. Hence, we can avoid computing $\Pr(t_i > t_j)$ for all record pairs (t_i, t_j), and sort the database based on the expected records' scores, which results in the same sorting based on $\Pr(t_i > t_j)$ values. Computing $\mathrm{E}[f_i]$ for every record t_i requires a linear scan over D, which has a complexity of $O(n)$, while the subsequent sorting step has a complexity of $O(nlog(n))$. It follows that the query evaluation procedure has an overall complexity of $O(nlog(n))$.

7 Uncertain scores construction

In this section, we give a method to construct uncertain scores from probabilistic scoring attributes, i.e., attributes defined as random variables with associated probability distributions. We start by describing how to model attributes with missing values as probabilistic attributes (Sect. 7.1), which widens the scope of our methods to databases with incomplete data. We then show how, given a scoring function defined on one or more probabilistic attributes, we compute a score interval and a score density for each record (Sect. 7.2).

7.1 Estimating missing values

We describe a simple technique to construct a probability distribution for the estimates of missing attribute values (e.g., the rent of apartment $a4$ in Fig. 2a), based on attribute correlations. We emphasize, however, that other methods, e.g., machine learning methods [6,7], can also fit our purposes. We contrast our method against other techniques in Sect. 9.

The strength of the correlation between two attributes a_i and a_j, denoted $c(a_i, a_j)$ is expressed as $\frac{|a_j|}{|a_j, a_i|}$, where $|.|$ refers to the number of distinct values (which can be obtained from system catalog). Similar definition is used in [28] to quantify the dependence among attributes in attribute pairs. The value $c(a_i, a_j)$ expresses the confidence that every distinct value in a_j is associated with a unique value in a_i. Our strategy is to predict missing attribute values, in an off-line stage, by identifying a set $\mathcal{S}$ of strong attribute correlations that are used, under independence assumption, to impute missing attribute values.

Some strong correlations may not be useful predictors. Specifically, if $|a_j|$ is close to the cardinality of the whole relation, then a_j is (approximate) key. In such case, a_j is trivially correlated with every other attribute [6,28]. An approximate key a_j has, with a high probability, a distinct value in each record. Hence, for a record t with missing a_i value and non-missing a_j value, the value of $t.a_j$ is most likely different from all other records. Therefore, the set of records the a_j value of which is the same as $t.a_j$ is most likely empty, and thus (a_i, a_j) is not a useful predictor for the missing $t.a_i$ value.

Similar to [28], for a relation R, we include in the set of attribute correlations $\mathcal{S}$ each correlation (a_i, a_j), with $c(a_i, a_j) > \epsilon_1$ and $(\epsilon_2 < |a_j|/|R| < \epsilon_3)$, where ϵ_1, ϵ_2, and ϵ_3 are input parameters in [0,1]. If there are already known correlations (functional dependencies), they can be directly added to $\mathcal{S}$ (e.g., a Car table usually involves the correlation $(make, model)$). Moreover, by evaluating dependencies among value distributions in attribute pairs (e.g., using the chi-square test of independence in [28]), correlations in $\mathcal{S}$ can be merged together capturing their dependence. For example, if attributes a_j and a_k are dependent, we can replace the correlations (a_i, a_j) and (a_i, a_k) in $\mathcal{S}$ with one correlation $(a_i, \{a_j, a_k\})$ that captures the joint distribution of $\{a_j, a_k\}$. For clarity of discussion, we focus on correlations of the form (a_i, a_j) in the following discussion.

Let a_i be an attribute of interest containing missing values. We use the correlation (a_i, a_j) to construct a two-dimensional histogram materializing the relative frequencies of all value combinations of a_i and a_j based on known (non-missing) values in both attributes. For a record t, with missing a_i value, and non-missing a_j value, we estimate $t.a_i$ using the values in the histogram bin associated with $t.a_j$. We thus obtain a number of $t.a_i$ estimates along with their relative frequencies. For every other correlation (a_i, a_k), we obtain similar estimates for $t.a_i$. We derive overall estimates of $t.a_i$ by averaging the frequencies of identical values obtained from individual histograms, weighted by the strength of the corresponding correlations.

We illustrate our technique using the following example. Assume an apartment record $t = (rooms = 2, area = 1000, zip = 94123, rent =?)$. Assume the correlations $(rent, area)$ and $(rent, zip)$ have strengthes of 0.9 and 0.8, respectively. Assume the bin of the histogram $(rent, area)$ at $area = 1,000$ has the following (value, frequency) pairs: $\{(700, 0.5), (800, 0.25), (850, 0.25)\}$. Similarly, assume the bin of the histogram $(rent, zip)$ at $zip = 94123$ has the following pairs $\{(700, 0.5), (800, 0.5)\}$. We combine both histogram bins into an overall $t.rent$ histogram $\{(700, 0.85), (800, 0.625), (850, 0.225)\}$, where, for example, the overall weight of the pair $(800, 0.625)$ is a weighted average of the frequencies of the pairs $(800, 0.25)$ and $(800, 0.5)$, with weights 0.9 and 0.8, respectively. Hence, we boost the weight of an estimate if multiple correlations agree on such estimate. We demonstrate, in Sect. 8, the effectiveness of such prediction method using real-world data.

The final step is normalizing the resulting histogram to generate a corresponding probability distribution on the possible fillers of the missing attribute value. We fit a probability distribution on the histogram $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ using kernel density estimation method, a widely used non-parametric regression technique to compute a density function from observations, defined as follows:

$$p(x) = \frac{1}{h.\sum_{i=1}^{m} y_i} \sum_{i=1}^{m} y_i.\kappa\left(\frac{x_i - x}{h}\right) \tag{11}$$

where $\kappa(.)$ is a standard Gaussian kernel with mean 0 and standard deviation 1. The intuition of Eq. 11 is to average the observations close to x weighted by their distances from x. The normalization constant $\sum_{i=1}^{m} y_i$ guarantees valid computed probabilities (i.e., the area under $p(.)$ curve is 1). In our experiments, we set the bandwidth parameter h, which determines the span of the kernel, to 1% of the histogram's span.

7.2 Aggregating uncertain scores

Given a query-specified scoring function $\mathcal{F}$, we show how to construct for each record t_i a score interval $v_i = [lo_i, up_i]$ enclosing t_i's possible scores, and a probability density function f_i defined on v_i.

7.2.1 Computing score intervals

Let $\mathcal{F}$ be a query-specified scoring function of the attributes $p_1, \ldots, p_n$. In many practical use cases, users adopt simple ranking functions reflecting their preferences. Monotone and bounded scoring functions are assumed in many recent top-k query processing proposals [4,29,30]. We call $\mathcal{F}(p_1, \ldots, p_n)$ a monotone function if $\mathcal{F}(x_1, \ldots, x_n) \leq \mathcal{F}(\acute{x_1}, \ldots, \acute{x_n})$ whenever $x_i \leq \acute{x_i}$ for every i, while we call $\mathcal{F}(p_1, \ldots, p_n)$ a bounded function if the range of $\mathcal{F}$ is bounded using the boundary values of p_i's.

Given a monotone or bounded scoring function, we derive v_i based on the boundary values of p_i's. If $\mathcal{F}(p_1, \ldots, p_n)$ is *monotone*, then $v_i = [\mathcal{F}(\underline{p_1}, \ldots, \underline{p_n}), \mathcal{F}(\overline{p_1}, \ldots, \overline{p_n})]$, where $\underline{p_i}$ and $\overline{p_i}$ are the minimum and maximum values in p_i's probability distribution (if p_i is a deterministic attribute, we use its value for both bounds). For example, assume a monotone function $\mathcal{F}_1(t) = t \,.\, p_1 + t \,.\, p_2$. For record t_i, assume $t_i \,.\, p_1$ and $t_i \,.\, p_2$ are defined as probability distributions over the intervals [8, 10] and [2, 5], respectively. Then, $v_i = [10, 15]$. Similarly, if $\mathcal{F}(p_1, \ldots, p_n)$ is *bounded*, then v_i is computed based on the boundary values of p_i's. For example, for a bounded function $\mathcal{F}_2(t) = (t \,.\, p_1 - t \,.\, p_2)^2$ (note that $\mathcal{F}_2$ is non-monotone), the score interval $v_i = [9, 64]$.

Relaxing our assumptions, regarding the class of scoring functions we support, requires employing multi-dimensional optimization techniques, e.g., gradient methods, to search for global minima and maxima of a multi-dimensional function in order to derive v_i's. We do not address such generalization in this paper.

7.2.2 Computing score densities

For a simple scoring function defined on a single scoring attribute p_j, the score density function f_i is the same as $t_i \,.\, p_j$'s probability distribution. If $t_i \,.\, p_j$ is a deterministic value, then f_i is equal to such value with probability 1.

For a multi-attribute scoring function $\mathcal{F}(p_1, \ldots, p_n)$, we need to combine the densities of different attributes to compute the overall score density f_i. For each record t, we use the probability distributions of $t \,.\, p_1, \ldots, t \,.\, p_n$ to sample m points $x_1^i, \ldots, x_m^i$ from the distribution of each attribute $t \,.\, p_i$. The score density of record t is computed as a joint density over the densities of individual attributes using a multi-dimensional kernel density estimator, defined as follows:

$$f(x_1, \ldots, x_n) = \frac{1}{m} \sum_{i=1}^{m} \prod_{j=1}^{n} \frac{1}{h_j} \kappa \left(\frac{x_i^j - x_j}{h_j} \right) \tag{12}$$

Equation 12 assumes the independence of scoring attributes (through multiplying the individual kernel estimators). Similar to Eq. 11, we set each bandwidth parameter h_j to 1% of the width of its corresponding attribute interval.

8 Experiments

All experiments are conducted on a SunFire X4100 server with two Dual Core 2.2 GHz processors, and 2 GB 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 scraping the search results of `apartments.com`, and (2) Cars: 10,000 car ads scraped from `carpages.ca`. The *rent* attribute in Apts is used as the scoring function (65% of scraped apartment listings have uncertain rent values), and similarly, the *price* attribute in Cars is used as the scoring function (10% of scraped car ads have uncertain price).

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

For synthetic data, the bounds of the score interval of each record t_i is generated by drawing a random interval starting point lo_i from the dataset corresponding distribution (uniform, Gaussian ($\mu = 0.5, \sigma = 0.05$), or exponential($\mu = 0.1$)) defined on the score range [0,1]. The width of the interval is uniform in [0,1]. The main intuition is to create different patterns of filling the score range with uncertain scores of different records. For example, while uniform distribution distributes the uncertain scores uniformly over the score range, exponential distribution creates a skewed pattern in which a few records have high scores, while the majority of records have low scores.

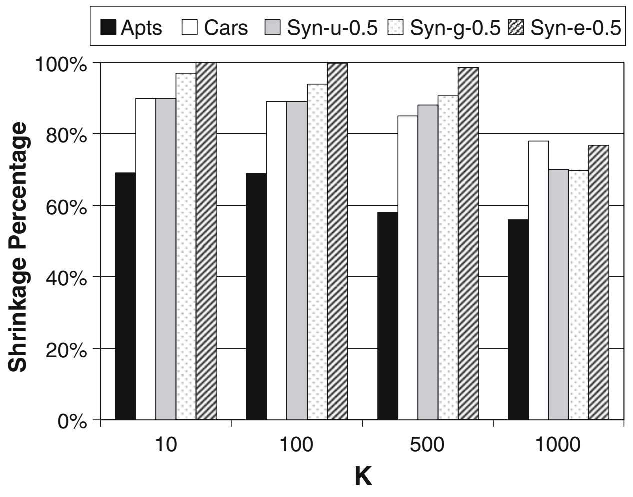


Fig. 7 Reduction in data size

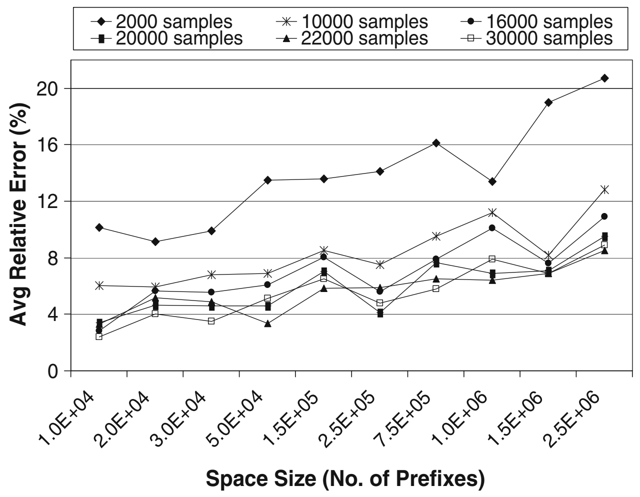


Fig. 8 Accuracy of Monte-Carlo integration

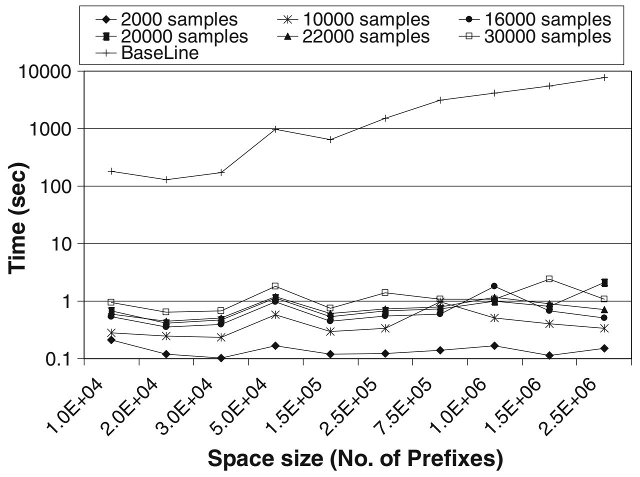


Fig. 9 Comparison with BASELINE

8.1 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.

We also evaluate the number of record accesses used to find the pruning position pos^* in the list U (Sect. 6.1). The logarithmic complexity of the algorithm guarantees a small number of record accesses of under 20 accesses in all datasets. The time consumed to construct the list U is under 1 s, while the time consumed by Algorithm 2 is under 0.2 s, in all datasets.

8.2 Accuracy and efficiency of Monte-Carlo integration

We evaluate the accuracy and efficiency of Monte-Carlo integration in computing UTop-Rankqueries. 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 Sect. 6.3, 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 8 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 more sensitive to the number of samples than 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 9 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.

8.3 Scalability with respect to k

We evaluate the efficiency of our query evaluation for UTop-Rank(1, k) queries with different k values. Figure 10 shows the query evaluation time, based on 10,000 samples. On the average, query evaluation time doubled when k increased by

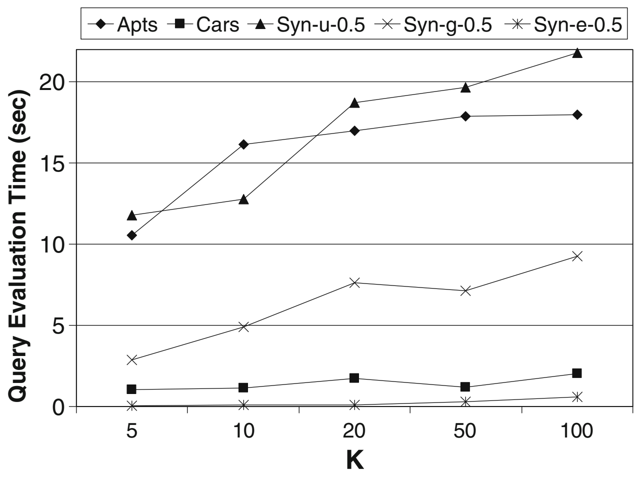


Fig. 10 UTop-Rank query evaluation time

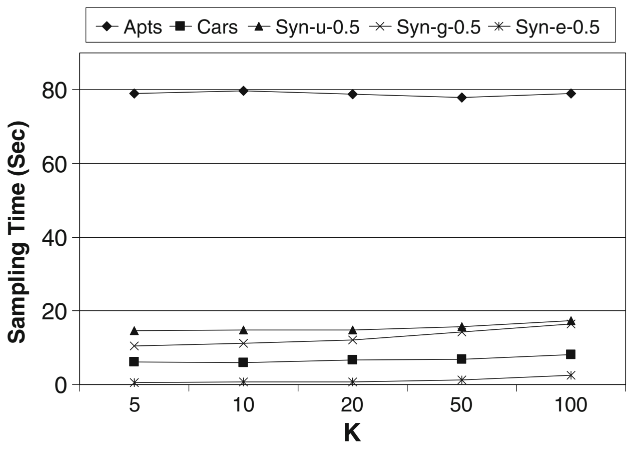


Fig. 11 UTop-Rank sampling time (10,000 samples)

20 times. Figure 11 shows the time consumed in drawing the samples.

The difference in sampling and ranking times for different datasets is attributed to two main factors:

- The variance in the reduced sizes of the datasets based on the k-dominance criterion. For example, the majority of records in Syn-e-0.5 dataset are pruned using k-dominance, while a much smaller number of records are pruned in Syn-u-0.5 dataset. This happens due to the different distributions of the bounds of score intervals. In general, the dataset size is inversely proportional to processing time.
- The percentage of records with uncertain scores. For example, the percentage of records with uncertain scores in Apts is 65%, while it is only 10% in Cars. Records with uncertain score results in longer processing times since space size (number of possible rankings) increases with score uncertainty.

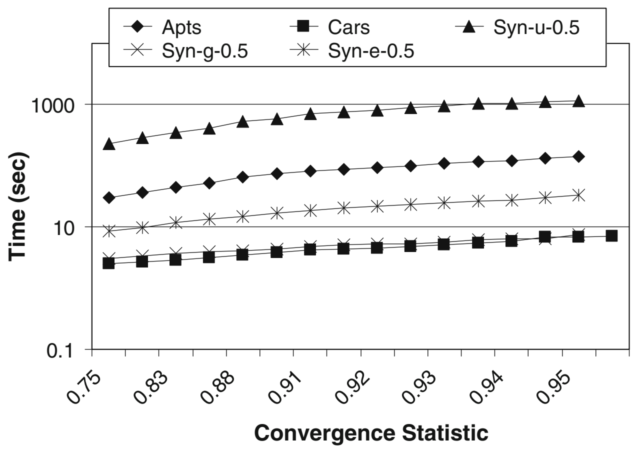


Fig. 12 Chains convergence

8.4 Markov chains convergence

We evaluate the Markov chains mixing time (Sect. 6.4). For 10 chains and $k = 10$, Fig. 12 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.

8.5 Markov chains accuracy

We evaluate the ability of Markov chains to discover states the probabilities of which 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 Fig. 13 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 range 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

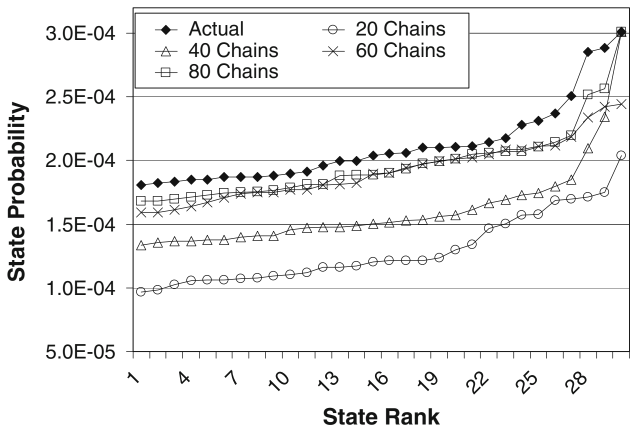


Fig. 13 Space coverage

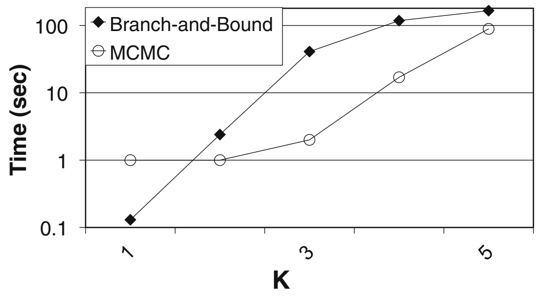


Fig. 14 Evaluation time (Syn-u-0.5, UTop-Prefix)

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 s to 400 s when the number of chains increased from 20 to 80.

8.6 Branch-and-bound search

In this experiment, we evaluate the branch-and-bound techniques we propose in Sect. 6.4.1 to evaluate UTop-Prefixand UTop-Setqueries. Figures 14 and 15 compare the processing time of branch-and-bound prefix search (Algorithm 3) and the MCMC sampling method (using 5 chains) for the datasets Syn-u-0.5 and Syn-g-0.5, respectively. The branch-and-bound search shows smaller running times with small k values, as it does not have the overhead of proposing states as in the MCMC method. As the value of k increases, the number of materialized candidates by the branch-and-bound search increases, which negatively impacts the running times.

The MCMC method is, on the average, one order of magnitude faster than the branch-and-bound search. The savings in processing time in MCMC method comes with the price of giving approximate answers. The average absolute error in the probability of the answer reported by the MCMC method, with respect to the branch-and-bound exact search, is 0.0012 and 0.0007 for Syn-u-0.5 and Syn-g-0.5, respectively. The

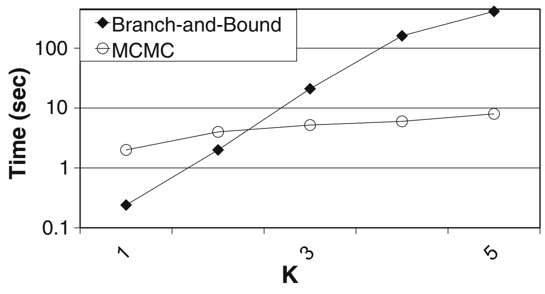


Fig. 15 Evaluation time (Syn-g-0.5, UTop-Prefix)

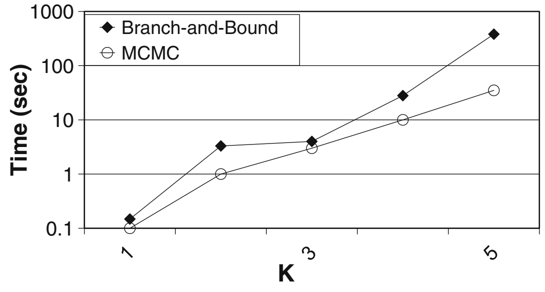


Fig. 16 Evaluation time (Apts, UTop-Prefix)

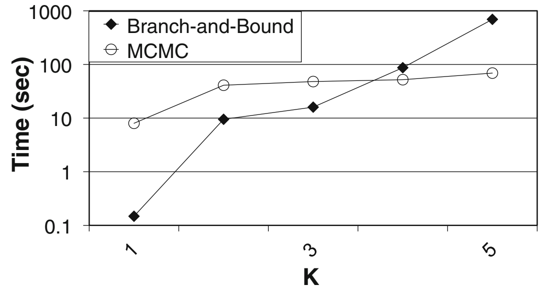


Fig. 17 Evaluation time (Apts, UTop-Set)

error decreases as the number of MCMC chains increases as we show in Sect. 8.5. Figures 16 and 17 show similar result for Apts dataset.

Next we evaluate the effectiveness of the greedy criteria adopted by branch-and-bound search. Figures 18 and 19 compare the processing times of branch-and-bound search against the BASELINE algorithm using Apts dataset for UTop-Prefixand UTop-Setqueries, respectively. The BASELINE algorithm shows an exponential increase in running time as space size (number of prefixes) increases (we omit running times that are significantly large). On the other hand, branch-and-bound search locates query answer in times below 30 s for both query types. Figure 20 compares the memory requirements (computed as the number of materialized candidates) of branch-and-bound and BASELINE algorithms. The BASELINE algorithm has, on the average, 3 orders of magnitude larger number of materialized

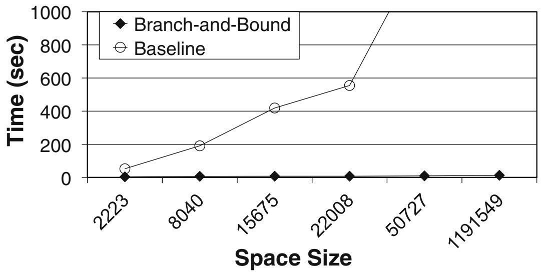


Fig. 18 Evaluation time (Apts, UTop-Prefix)

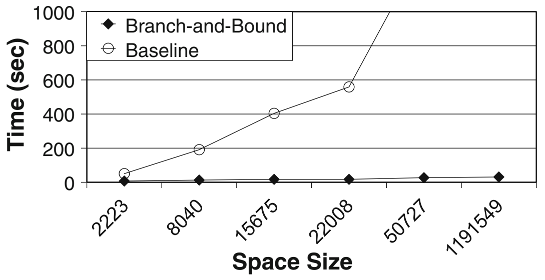


Fig. 19 Evaluation time (Apts, UTop-Set)

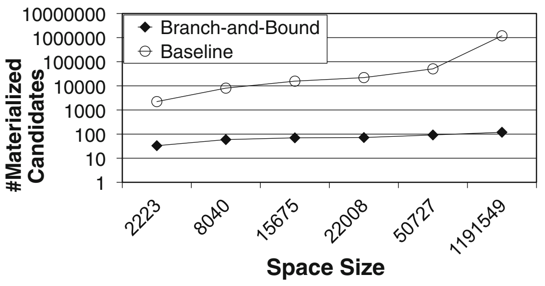


Fig. 20 Consumed memory (Apts, UTop-Prefix)

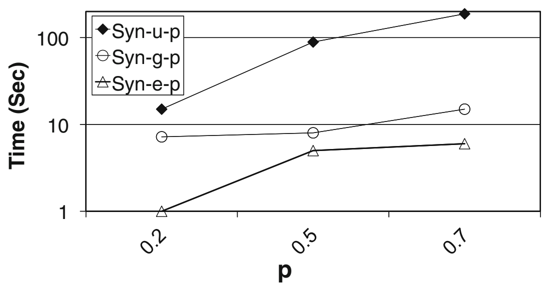


Fig. 21 Effect of records with uncertain scores (MCMC,UTop-Prefix(5))

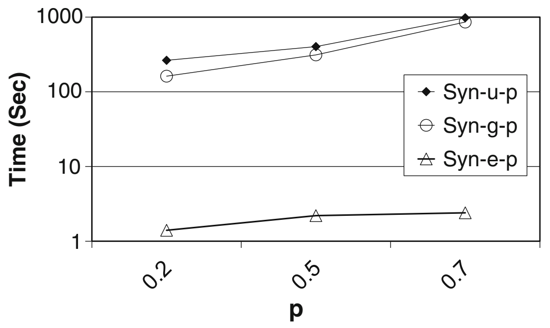


Fig. 22 Effect of records with uncertain scores (branch-and-bound,UTop-Prefix(5))

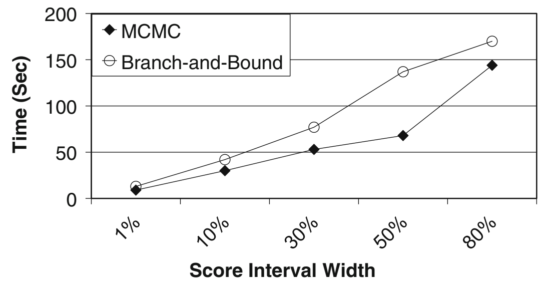


Fig. 23 Effect of score interval width (UTop-Prefix(5))

candidates, which illustrates the effectiveness of the pruning techniques adopted by branch-and-bound search.

8.7 Score uncertainty

In this experiment, we evaluate the effect of score uncertainty on algorithms performance. Figures 21 and 22 show the effect of the parameter p (the proportion of records with uncertain scores) on the running times of MCMC and branch-and-bound search in different datasets. Increasing p results in linear increase in the running times of both algorithms. On the average, as p doubled by 3.5 times, the running time of the MCMC method doubled by 5 times, while the running time of the branch-and-bound search doubled by 2.5 times.

We next evaluate the effect of the width of score interval on algorithms performance. We create synthetic data with different score interval width, where the interval width is represented as a percentage of the whole score range. As the score interval width increases, the number of records with incomparable scores increases. This results in limiting the effect of pruning by score dominance, and hence increasing the overall running times. Figure 23 shows linear increase in the running times of MCMC and branch-and-bound search as the score interval width increases.

8.8 Score imputation

In this experiment, we evaluate the techniques proposed in Sect. 7 to impute score intervals, and score densities based on attribute correlations. In order to evaluate the accuracy of imputed scores, we select a subset of records with single-valued (deterministic) scores, and hide these scores before applying our score imputation method. We thus introduce missing data for which we have the ground truth. We then compute an uncertain score (i.e., a score interval and a score density) for each record with a hidden score. Finally, we

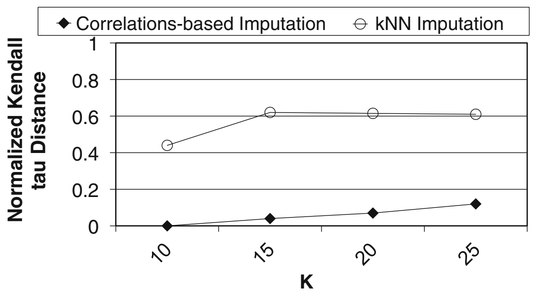


Fig. 24 Accuracy of score imputation (Apts)

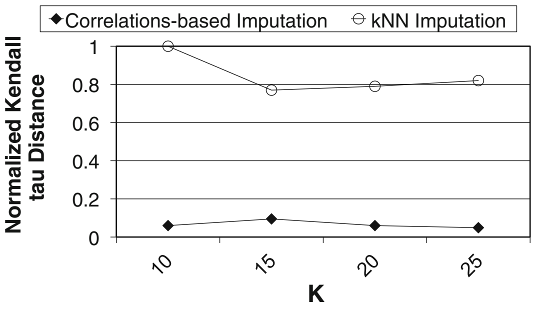


Fig. 25 Accuracy of score imputation (Cars)

evaluate the ranking generated by the MCMC method against the true ranking (given by the true values of the hidden scores). Ranking accuracy is measured using the normalized Kendall tau distance (cf. Sect. 6.5), which is a measure in [0,1] of disagreements between two rankings.

To assess the effectiveness of our imputation techniques with respect to simple imputation methods, we repeat the above procedure using the following k-NN missing value estimation technique, implemented in the R system [31]: for each record with a missing score, we find the k nearest neighbor records based on Euclidean distance metric. We impute the missing scores by averaging the non-missing scores of the neighbors. If the scores of all neighbors are missing, we use the overall score mean as an estimator. Figures 24 and 25 show the accuracy comparison of our correlation-based score imputation method and the k-NN imputation method for Apts and Cars datasets, respectively. Our score imputation method shows high ranking accuracy with a normalized Kendall tau distance below 0.1 in both datasets, which illustrates the value of exploiting uncertain scores to a compute a reliable ranking.

9 Related work

The techniques we propose in this paper are mainly related to two large research areas: *probabilistic ranking*, and *handling incomplete data*. We summarize some of the recent proposals in both areas, and highlight the major differences between these proposals and our proposal.

Probabilistic ranking. 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 [16], while [17,18] 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.

To the best of our knowledge, defining a probability space on the set of linear extensions of a partial order to quantify the likelihood of possible rankings has not been addressed before. Dealing with the linear extensions of a partial order has been addressed in other contexts (e.g., [11,32]). 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.

Monte-Carlo methods are used in [33] 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 [33] 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 [34] 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 [34] to estimate the approximation error.

The method proposed in [35] use the notion of *generating functions* to construct a unified ranking function that can be instantiated to multiple ranking functions proposed in the current literature. The given algorithms use an and–or tree model in which leaf nodes are tuple instances that can be possibly exclusive. The model in [35] is based on tuple-level uncertainty, where each tuple belongs to the database with some confidence. Hence, tuples may exist/not exist in a given possible world of the database. The model we assume in this paper captures uncertainty in tuple scores in the form of score ranges; a representation that is adopted by multiple real data sources particularly on the Web (cf. Sect. 1). Hence, in contrast to [35], our model enforces all tuples to belong to any possible world (linear extension). Moreover, since [35] assumes a fixed score per tuple, the relative order of tuples is fixed over all possible worlds. On the other hand, our model encodes different relative orders of tuples with intersecting score intervals.

The problem of computing consensus answers in probabilistic databases has been recently addressed in [36] through adopting the and–or tree model in [35]. And–or trees cannot be used to encode tuples with uncertain scores in the form of score ranges without losing information. The reason is that each tuple in this case has effectively an infinite number of instances. The algorithms given in [36] for computing a consensus ranking return a consensus top-k answer, while the methods we propose in Sect. 6.5 return a consensus full ranking. In addition, while [36] gives an approximate algorithm for rank aggregation under Kendall tau distance, we identify different classes of PPO's in which an exact polynomial time algorithm for rank aggregation under Kendall tau distance exists.

9.1 Handing incomplete data

We categorize missing value estimation techniques into three main groups:

– Statistical techniques: these techniques adopt statistical approaches to estimate missing values. Examples include estimation using mean values, regression methods, expectation maximization, and multiple imputation [37,38]. The goal of these methods is usually preserving the overall data distribution (e.g., avoiding bias in the distribution mean as a result of missing values estimation). The computed estimates are thus not primarily meant to give accurate predictions for the missing values individually, and hence they may be unsuitable when computing a ranking based on the estimated values of missing scores.
– Machine learning techniques: methods in this group learn prediction models trained with complete data instances, and use these models to derive probabilistic estimates for missing values. One example is [6] where naïve Bayes classifiers, trained with functional dependencies, are used to derive probabilistic predictions of missing values. Another example is [7], where missing values are learned from summary information derived from the raw data. The correlations-based estimation method we describe in Sect. 7.1 falls in this category.
– Database-oriented techniques: database proposals dealing with missing values focused mainly on modeling alternatives and their effect on query processing, rather than the physical learning and estimation aspects. One example is [39], where missing values are represented using intervals derived from attribute domain. Each incomplete tuple is represented as a set of different instances (duplicates), where each instance corresponds to one possible value in the interval. Applying this method when predictions are in the from of continuous intervals requires discretizing the intervals, which can have negative impact on storage cost and accuracy of reported results.

10 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 methods based on Markov chains to compute approximate query answers. We also gave polynomial time algorithms to solve the rank aggregation problem in probabilistic partial orders. Our experimental study on both real and synthetic datasets demonstrates the scalability and accuracy of our techniques.

References

1. Sarma, A.D., Benjelloun, O., Halevy, A., Widom, J.: Working models for uncertain data. In: ICDE (2006)
2. Benjelloun, O., Sarma, A.D., Halevy, A., Widom, J.: Uldbs: databases with uncertainty and lineage. In: VLDB (2006)
3. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB (2004)
4. Chang, K.C.-C., Hwang, S.: Minimal probing: supporting expensive predicates for top-k queries. In: SIGMOD (2002)
5. Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. **40**(4) (2008)
6. Wolf, G., Khatri, H., Chokshi, B., Fan, J., Chen, Y., Kambhampati, S.: Query processing over incomplete autonomous databases. In: VLDB (2007)

7. Wu, X., Barbará, D.: Learning missing values from summary constraints. SIGKDD Explor. **4**(1) (2002)
8. Chomicki, J.: Preference formulas in relational queries. ACM Trans. Database Syst. **28**(4) (2003)
9. Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: SIGMOD (2006)
10. Tao, Y., Xiao, X., Pei, J.: Efficient skyline and top-k retrieval in subspaces. TKDE **19**(8) (2007)
11. Brightwell, G., Winkler, P.: Counting linear extensions is #p-complete. In: STOC (1991)
12. Cheng, R., Prabhakar, S., Kalashnikov, D.V.: Querying imprecise data in moving object environments. In: ICDE (2003)
13. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: WWW (2001)
14. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M., Hong, W.: Model-based approximate querying in sensor networks. VLDB J. **14**(4) (2005)
15. Abiteboul, S., Kanellakis, P., Grahne, G.: On the representation and querying of sets of possible worlds. In: SIGMOD (1987)
16. Soliman, M.A., Ilyas, I.F., Chang, K.C.-C.: Top-k query processing in uncertain databases. In: ICDE (2007)
17. Zhang, X., Chomicki, J.: On the semantics and evaluation of top-k queries in probabilistic databases. In: ICDE Workshops (2008)
18. Hua, M., Pei, J., Zhang, W., Lin, X.: Ranking queries on uncertain data: a probabilistic threshold approach. In: SIGMOD (2008)
19. O'Leary, D.P.: Multidimensional integration: partition and conquer. Comput. Sci. Eng. **6**(6) (2004)
20. Jerrum, M., Sinclair, A.: The markov chain monte carlo method: an approach to approximate counting and integration. Approximation algorithms for NP-hard problems (1997)
21. Hastings, W.K.: Monte carlo sampling methods using markov chains and their applications. Biometrika **57**(1) (1970)
22. Gelman, A., Rubin, D.B.: Inference from iterative simulation using multiple sequences. Stat. Sci. **7**(4) (1992)
23. Cowles, M.K., Carlin, B.P.: Markov chain Monte Carlo convergence diagnostics: a comparative review. J. Am. Stat. Assoc. **91**(434) (1996)
24. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors. In: STOC (2007)
25. van Acker, P.: Transitivity revisited. Ann. Oper. Res. **23**(1–4) (1990)
26. Intriligator, M.D.: A probabilistic model of social choice. Rev. Econ. Stud. **40**(4) (1973)
27. Fishburn, P.C.: Probabilistic social choice based on simple voting comparisons. Rev. Econ. Stud. **51**(4) (1984)
28. Ilyas, I.F., Markl, V., Haas, P.J., Brown, P., Aboulnaga, A.: Cords: automatic discovery of correlations and soft functional dependencies. In: SIGMOD (2004)
29. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **1**(1) (2001)
30. Xin, D., Han, J., Chang, K.C.-C.: Progressive and selective merge: computing top-k with ad-hoc ranking functions. In: SIGMOD (2007)
31. The R project for statistical computing: http://www.r-project.org
32. Bubley, R., Dyer, M.: Faster random generation of linear extensions. In: SODA (1998)
33. Re, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: ICDE (2007)
34. Wu, M., Jermaine, C.: A Bayesian method for guessing the extreme values in a data set. In: VLDB (2007)
35. Li, J., Saha, B., Deshpande, A.: A unified approach to ranking in probabilistic databases. PVLDB **2**(1) (2009)
36. Li, J., Deshpande, A.: Consensus answers for queries over probabilistic databases. In: PODS (2009)
37. Little, R., Rubin, D.B.: Statistical Analysis with Missing Data. Wiley & Sons, New York (1987)
38. Rubin, D.B.: Multiple Imputation for Nonresponse in Surveys. Wiley & Sons, New York (1987)
39. Ola, A., Ozsoyoglu, G.: Incomplete relational database models based on intervals. IEEE TKDE **05**(2) (1993)