A Rank-Based Approach to Recommender System's Top-K Queries with Uncertain Scores (Technical Report)

ABSTRACT

Top-K queries provide an answer that is ranked using a score that can either be given explicitly or computed from tuple values. Recommender systems use scores, based on user feedback on items with which they interact, to answer top-K queries. Such scores pose the challenge of correctly ranking elements using scores that are more often than not, uncertain. In this work, we address top-K queries based on uncertain scores. We propose to explicitly model the inherent uncertainty in the provided data and to consider a distribution of scores instead of a single score. Rooted in works of database probabilistic ranking, we offer the use of probabilistic ranking as a tool of choice for generating recommendation in the presence of uncertainty. We argue that the ranking approach should be chosen in a manner that maximizes user satisfaction, extending state-of-the-art on quality aspect of top-K answers over uncertain data, their relationship to top-K semantics, and improve ranking with uncertain scores in recommender systems. Towards this end, we introduce RankDist, an algorithm for efficiently computing probability of item position in a ranked recommendation. We show that rank-based (rather than score-based) methods that are computed using RankDist, which were not applied in recommender systems before, offer a guaranteed optimality by expectation and empirical superiority when tested on common benchmarks.

1 INTRODUCTION

Top-K queries are prevalent in recommender systems where the goal is to present users with a ranked list of items (a Top-K answer) in descending order of relevance [2]. Recommender systems often rely on scoring, where users provide feedback on items with which they interact, a feedback that is utilized by other users who wish to enhance their decision making with user recommendations.

Scoring poses a challenge to recommender systems that need to correctly rank elements using scores that are more often than not, uncertain. There are multiple sources of uncertainty in recommender systems, with the two major ones being unreliable and missing data. *Data unreliability* in recommender systems is mainly attributed to user's feedback on items with which they interact, which may be provided either implicitly (*e.g.*, purchases, clicks) or explicitly (*e.g.*, reviews, ratings). When giving explicit feedback, each user expresses a subjective opinion that might be wrong or imprecise, and therefore, its trustworthiness is doubtful. A second source of uncertainty with which recommender systems are faced involves *missing data*. Acquiring complete feedback data in recommender systems is unrealistic, partially due to the inability of users to express their opinion on all items and users' biases that affect the decision of which items to rate. Therefore, the knowledge about interactions of users with items is limited. To counter the missing data phenomenon, user preferences are usually assessed based on historical interactions or users that are deemed similar.

Data unreliability and missing data calls for embedding uncertainty considerations into the recommendation process. One solution to scoring with uncertain scores is to ignore uncertainty and assume scores are deterministic. In such a setting, each tuple is assigned with a single score according to which ranking is performed. Here, there is a single response to a top-K query, retrieving the K items with the highest scores. While this scenario may be attractive in terms of processing time, not acknowledging uncertainty may cripple recommender systems severely in terms of quality performance. In practice, top-K queries need to process uncertain data.

In this work, we address top-K queries with uncertain scores. As scores may be erroneous or imprecise, we propose to explicitly model the inherent uncertainty in the provided data and to consider a **distribution** of scores instead of a single score. While we assume the existence of an underlying true score, the exact (deterministic) ground truth is unknown. Therefore, our objective is to incorporate uncertainty as an integral part of the top-K recommendation pipeline.

Several studies [4, 17, 22, 28] have dealt with score uncertainty, proposing score-based methods for ranking with uncertain scores (see details in Section 6). Our proposed method also supports rank-based semantics, wherein possible ranking results are examined.

Our work relies on a vast amount of research that was devoted to probabilistic ranking (see summary of efforts in [15, 29]), offering the use of probabilistic ranking as a tool of choice for generating recommendation in the presence of uncertainty. Probabilistic ranking was discussed in reference to approximating top-K results [7], computing skyline queries [9], improving data lake retrieval [11, 32], and for data cleaning [6] and predictive analytics [8], among others. It was also applied to multiple data types, including geospatial data [33]. To this end, numerous alternative semantics of top-K query results with probabilistic ranking have been suggested [3, 31]. For instance, the Global top-k approach returns K tuples having the highest probability to be ranked at the first K places [31]. Other approaches include the Expected score [3], returning K tuples having the highest expected scores, and U-kRanks, returning a multi-set of K tuples having the highest probability to be ranked at the respective top-K positions.

With such a wide range of probabilistic ranking semantics for what should be the answer to a top-K query with uncertain scores, the question of what is a good semantics presents itself. Intuitively, the top-K ranked items, for some reasonable value of K, are likely to be the items the user considers to be most relevant, those items to call up and choose from.

We argue that the ranking approach should be chosen in a manner that the outputted result optimizes the selected objectives. A flexible approach towards query processing is needed, as application target quality may vary. For instance, an application willing to accept top-K items disregarding their ordering may use a different query processing method compared to an application that emphasizes result ordering. By mapping the closely related method to the objectives, in terms of quality measures, we should be able to generate a high quality result and accordingly maximize user satisfaction.

We propose to extend state-of-the-art (*e.g.*, [3, 19]) on quality aspect of top-K answers over uncertain data and their relationship to top-K semantics, and also extend the state-of-the-art (*e.g.*, [4]) in ranking with uncertain scores in recommender systems. We investigate two quantitative, a posteriori quality measures, *Precision-at-K (P@K)* and *Discounted Cumulative Gain-at-K (DCG@K)*, typically used in recommender systems. P@K and DCG@K are set-based and list-based quality measures, respectively, concentrating on ranking quality, with roots in the research area of Information Retrieval (IR) [21]. For probabilistic ranking, the expectation of this measure attests to the quality of the query answer, where a higher expectation value suggests a higher quality answer. In addition, we focus on P@$K(K)$, a special case of P@K, where P@K probability is 1. Our goal is to provide quality answers for top-K queries that are posed to a database with uncertain scores, aiming to optimize P@$K(K)$ and the expectation of P@K and DCG@K.

In this work, we offer the following contributions:

(1) We define P@K, P@$K(K)$ and DCG@K in the context of top-K query answering, providing ranking approaches for computing top-K answers in the presence of uncertain scores, and characterizing selection criteria that provably optimize the expectation of these metrics (Section 3).
(2) We provide methods to generate top-K recommendation answers when dealing with uncertain scores (Section 3), and introduce an efficient algorithm for computing the top-K recommendation answers for various probabilistic ranking approaches (Section 4).
(3) We provide an evaluation framework for comparing the quality of various ranking methods in the presence of scores uncertainty (Section 4).
(4) We provide a thorough empirical investigation to evaluate the performance of several ranking methods using real world recommender system datasets and show the superiority of the rank-based approach over the score-based approaches (Section 5).

The rest of the paper is organized as follows. We introduce a top-K pipeline for ranking queries with uncertain scores within recommender systems in Section 2. In Section 3, we formulate the ranking problem with the aim of generating high-quality top-K recommendation when dealing with uncertain scores. We provide a method for computing top-K result using multiple approaches, and assessing the expected quality of ranking results in Section 4. Our empirical evaluation are presented in Section 5. We review related work in Section 6 and conclude the paper in Section 7.

2 TOP-K QUERY PROCESSING PIPELINE

We introduce a top-K query processing pipeline (Figure 1) as an overview of the processing steps involved in answering ranking queries over uncertain data in recommender systems.

When it comes to ranking queries using scores, the input dataset contains information about users, items, and their interactions. We define U_I and U_U to be a universe of M items and Q users where $p_i \in U_I$ and $u_j \in U_U$ indicate a single item and a single user, respectively. Users' feedback on items with which they interact is collected and recorded in a database D. A tuple in the database contains (at least) an item (p_i), a user (u_j), and a score s. We assume

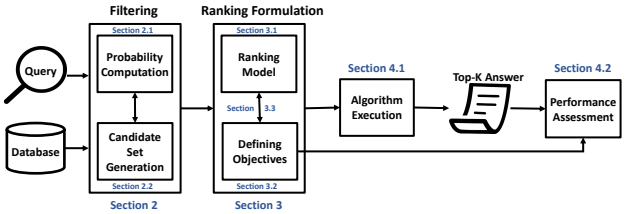


Figure 1: Top-K processing pipeline for generating high-quality recommendation from uncertain scores. Each component is annotated with the section where it is discussed.

a user provides as an explicit feedback some score s, which belongs to a closed set $Scores$ of possible scores (*e.g.*, the set $\{1, 2, 3, 4, 5\}$). We note that from D, one can extract a rating matrix R in which each row and column represents a user id and an item id, respectively, and a score entry that is assigned to the item by the user. We focus on answering top-K queries in recommender systems for an individual user. Hence, the input query for the pipeline is recommending K most relevant items for users according to their preferences.

The first step in the processing pipeline involves *filtering*, where we perform *probability computation* (Section 2.1) and *candidate set generation* (Section 2.2). This step is preliminary to the main contribution of our work, drawing from existing recommender systems and probabilistic ranking literature. Next, Section 3 discusses the *ranking formulation*, followed by the *algorithm execution* and *performance assessment* in Section 4. Each component of the pipeline is annotated in Figure 1 with the section where it is discussed.

2.1 Probability Computation

Viewing each user as a social sensor that provides an opinion regarding items, the gathered information serves in generating top-K recommendation. User feedback is subject to user's cognitive biases and noise and hence, similar to sensor errors, user's feedback may be imprecise [4]. Such imprecision is reflected in the database D, which may contain erroneous or imprecise information. To cope with imprecision, we model data uncertainty using uncertain scores.

To explicitly account for the uncertainty surrounding the scores assigned by users to items, we utilize a machine learning technique to generate score distribution for each user-item pair [4]. Focusing on Collaborative Filtering methods for recommender systems, the method receives the rating matrix R as input and generates a probabilistic rating matrix R^P as output, in which for each user u_j and item p_i, a score distribution is defined. In Section 5, we detail the specific approaches employed in the empirical evaluation to estimate probabilities and generate candidate sets for ranking using scores.

We base our model of data uncertainty on the probabilistic database literature [27]. We adopt the *attribute-level* uncertainty, which assumes tuple attributes have uncertain values. Accordingly, a tuple can be associated with uncertain scores as given by discrete or continuous score distribution. We use $S(p_i, u_j)$ to denote score distribution assigned to item p_i by user u_j. $S(p_i, u_j)$ is a random variable, in which $s(p_i, u_j)$ is a possible value of $S(p_i, u_j)$ with probability of $\Pr(S(p_i, u_j) = s(p_i, u_j))$. We assume that $S(\cdot)$ has a finite set of possible values, with $s(\cdot)$ being a possible value of $S(\cdot)$. We note

that continuous scores can be handled by discretization, for example, by binning the continuous values into intervals.

2.2 Candidate Set Generation

In recommender systems, the set U_I may contain a large number of items, which can hinder the ranking performance. Recently, Coscrato and Bridge [4] proposed the Uncertainty-based filtering (UBF) method, which filters out items with low predicted confidence prior to ranking. Filtering is also a common approach in Information Retrieval [12] to improve ranking performance. Therefore, this phase in the pipeline narrows the search space by selecting a subset $U \subseteq U_I$, of size N, of candidate items for ranking. We denote an item in U by σ. Items are selected with the aim of eliminating irrelevant content that can utilize, among other things, the predicted score distribution. For example, consider the probability-of-relevance (PRR) method [4], which selects the top-N items with the highest probability of achieving a score surpassing a certain threshold.

In this work, we propose a rank-based approach for ranking, which can be more computationally expensive compared to score-based approaches. Therefore, we harness this step to improve the efficiency of the ranking algorithm by filtering out items that are regarded as irrelevant, which allows the ranking process to be faster. However, this stage may exclude items the user may like, thus preventing the recommendation algorithm from including them in the recommendation list. Therefore, this step is employed to balance effectiveness and efficiency. When no filtering is applied, this step yields the entire set U_I ($U = U_I$).

Table 1: An example of a universe

U_I	**Score distribution** $S(\sigma)$	**True Score**	**PRR Score**
σ_1	$\{(2, 0.4), (4, 0.6)\}$	4	0.6
σ_2	$\{(1, 0.2), (4.5, 0.8)\}$	4.5	0.8
σ_3	$\{(0.5, 0.1), (3, 0.4), (5, 0.5)\}$	5	0.5
~~σ_4~~	~~$\{(2.5, 0.7), (4.0, 0.2), (5.0, 0.1)\}$~~	~~3.5~~	~~0.3~~

EXAMPLE 1. *Consider a recommender system of an entertainment service company,* e.g.*, MovieLens, where users engage with media using an online service and provide feedback,* e.g.*, ratings, on items with which they interact. In this system, users are offered the option of limiting the number of items to browse through, using top-K queries that yield a ranked recommendation. Table 1 depicts a universe of four items, including their score distribution, for a particular user. These distributions have been created using a machine learning model that leverages users' historical feedback, such that a score is assigned with a probability that is relative to the certainty that the user would assign this score to the item. It is worth noting that the 'true' score to be assigned by the user is masked by uncertainty and remains hidden during the recommendation phase. Consider applying the PRR filtering rule, in which we select* $N = 3$ *items with the highest probability of obtaining a score greater than a threshold of* 4.0*. We present in the last column the PRR scores that determine the items selected to form the universe. For instance, item* 4 *has a PRR score of* 0.3 *(*0.2 *for a score of* $4.0 + 0.1$ *for a score of* 5.0*). For* $N = 3$*, only the top three items according to the PRR score form the candidate set* U*. Therefore, the candidate items for the user ranking task include* $\{\sigma_1, \sigma_2, \sigma_3\}$*.*

Overall, the output of the filtering step is a set of tuples (items) U to be ranked for each user u_j, along with score distribution $S(\sigma)\ \forall \sigma \in U$. We assume that $S(\sigma)$ and $S(\sigma')$ are pair-wise probabilistically independent for all $\sigma, \sigma' \in U$ [24].

3 RANKING FORMULATION

Following the filtering step, we formulate the recommendation ranking problem for a user by defining the ranking model (*Ranking Model*, see Figure 1), and the learning objectives (*Defining Objectives*). Recall that U is a universe of N tuples, consisting of all candidate tuples to be ranked with respect to a user, where each tuple $\sigma \in U$ is associated with a score distribution $S(\sigma)$. As a shortcut notation, $s \in \sigma$ states that "s is a possible value of $S(\sigma)$." These scores, in particular, serve us in ranking tuples when answering queries.

For deterministic scores there is a single correct ranking, yet uncertain scores induce a large number of *possible worlds*. Each possible world represents a possible deterministic instance of the probabilistic database and is associated with a probability to be the correct form of the database. Adopting the possible world semantics, each tuple $\sigma \in U$ can be seen as stochastically "choosing" a score value from $S(\sigma)$. Jointly, the choices of all tuples in U induce a possible world. We denote by W the set of all possible worlds, and by $s_w(\sigma)$ the (deterministic) score of σ in a world $w \in W$. The likelihood of each possible world is given by the product of marginal probabilities of tuples' selections, that is, $Pr(w) = \prod_{\sigma \in U} Pr(s_w(\sigma))$. Finally, by $r_w(\sigma) \in R(\sigma)$ we denote the rank of σ in world w, with $R(\sigma)$ being a random variable capturing the probability distribution induced by worlds W on the possible ranks of σ. That is, for $1 \leq i \leq M$, $Pr(R(\sigma) = i) = \sum_{w \in W} Pr(w) \cdot \delta(r_w(\sigma) = i)$, where δ is an indicator function, returning 1 if $r_w(\sigma) = i$ and 0 otherwise.

Table 2: An example of possible worlds

World	***Prob.***	***Ranked list***
$w_1 = \{s(\sigma_1) = 2, s(\sigma_2) = 1, s(\sigma_3) = 0.5\}$	0.008	$\sigma_1, \sigma_2, \sigma_3$
$w_2 = \{s(\sigma_1) = 2, s(\sigma_2) = 1, s(\sigma_3) = 3\}$	0.032	$\sigma_3, \sigma_1, \sigma_2$
$w_3 = \{s(\sigma_1) = 2, s(\sigma_2) = 1, s(\sigma_3) = 5\}$	0.04	$\sigma_3, \sigma_1, \sigma_2$
$w_4 = \{s(\sigma_1) = 2, s(\sigma_2) = 4.5, s(\sigma_3) = 0.5\}$	0.032	$\sigma_2, \sigma_1, \sigma_3$
$w_5 = \{s(\sigma_1) = 2, s(\sigma_2) = 4.5, s(\sigma_3) = 3\}$	0.128	$\sigma_2, \sigma_3, \sigma_1$
$w_6 = \{s(\sigma_1) = 2, s(\sigma_2) = 4.5, s(\sigma_3) = 5\}$	0.16	$\sigma_3, \sigma_2, \sigma_1$
$w_7 = \{s(\sigma_1) = 4, s(\sigma_2) = 1, s(\sigma_3) = 0.5\}$	0.012	$\sigma_1, \sigma_2, \sigma_3$
$w_8 = \{s(\sigma_1) = 4, s(\sigma_2) = 1, s(\sigma_3) = 3\}$	0.048	$\sigma_1, \sigma_3, \sigma_2$
$w_9 = \{s(\sigma_1) = 4, s(\sigma_2) = 1, s(\sigma_3) = 5\}$	0.06	$\sigma_3, \sigma_1, \sigma_2$
$w_{10} = \{s(\sigma_1) = 4, s(\sigma_2) = 4.5, s(\sigma_3) = 0.5\}$	0.048	$\sigma_2, \sigma_1, \sigma_3$
$w_{11} = \{s(\sigma_1) = 4, s(\sigma_2) = 4.5, s(\sigma_3) = 3\}$	0.192	$\sigma_2, \sigma_1, \sigma_3$
$w_{12} = \{s(\sigma_1) = 4, s(\sigma_2) = 4.5, s(\sigma_3) = 5\}$	0.24	$\sigma_3, \sigma_2, \sigma_1$

EXAMPLE 2. *Recall the filtered universe presented in Table 1, with three items and their score distribution. For this toy example, Table 2 explicitly states the set of possible worlds, each is associated with a probability, computed as the product of score probabilities of all participating tuples. For example, the probability of world* w_1 *is given by* $0.4 \times 0.2 \times 0.1$*. In addition, the ranking in each world*

is determined based on the chosen scores for each tuple. World w_2, for example, ranks σ_3 first, then σ_1 and then σ_2. σ_1 is ranked first in three of the worlds, second in six of the worlds and third in three of the worlds. Using the probabilities associated with each world, we can define the rank distribution probability of σ_1 to be first with probability 0.068*, second with probability* 0.404*, and third with probability* 0.528*.*

Example 2 may suggest that there is a single top-K answer that can be derived from the possible worlds. However, unlike the case of deterministic scores, there is no single solution to recommendation queries when dealing with uncertain data. The literature on top-K queries over uncertain data demonstrates a rich palette of semantics for top-K querying with probabilistic ranking [14, 24–26, 31]. To this end, we define $A = \{\sigma_1, \ldots, \sigma_{K'}\} \subseteq U$ to be a top-K answer. A may be unordered or ordered according to some rule. For the latter, we denote the tuple that is ranked in the i position in A as $\sigma_{(i)}$. The cardinality of A is $|A| = K'$ where K' may be different than K according to some of the semantics in the literature. Extending the shortcut to a set of tuples, $s \in \sigma_i^A$ is a shorthand notation for "s is a possible value of $S(\sigma_i)$ for some $\sigma_i \in A$."

The growing interest in recommender systems calls for a deeper investigation of the quality aspect of top-K query answering over tuples with uncertain scores. In this work, our goal is to examine the quality of a recommender system top-K query result over a database with uncertain scores, by examining a posteriori quality metrics. Section 3.1 presents alternative ranking methods that yield top-K recommendation by harnessing score and rank distributions. Section 3.2 introduces quality measures for top-K queries. Targeting the optimization of top-K query results with respect to quality measures, we provide a formal pairing of three quality measures with top-K semantics (Section 3.3).

3.1 Ranking Model

The *Ranking Model* specifies the result size, the procedure to select the items included in the answer and their ordering within. The ranking model can follow deterministic or stochastic approaches with respect to the scores. For the former, a deterministic score is computed for each candidate object, serving to form the ranking result. For the latter, the stochastic version induces distribution over object scores, enabling the adoption of diverse top-K semantics for probabilistic ranking.

We next present approaches for producing top-K recommendations tailored to a specific user u_j, given a set of objects to be ranked U, along with score distribution $S(\sigma)$ and rank distribution $R(\sigma)$ for each $\sigma \in U$.

Focusing on uncertain scores stochastic top-K semantics can be either *score-based* or *rank-based*. The former returns a top-K result based on item score distributions, while the latter examines possible ranking results induced by score uncertainty. We next introduce examples of score-based and rank-based methods. To do so, we use two user-specific thresholds, denoted as t_s and t_p, representing the threshold for score and probability, respectively.

Recently, Coscrato and Bridge [4] presented two score-based ranking approaches for recommender systems that account for the uncertainty in item scores, namely uncertainty-based filtering and probability-of-relevance ranking.

Uncertainty-Based Filtering (UBF): Given items with predicted score $\hat{s}$ and respective uncertainty p, UBF returns K tuples with highest predicted score $\hat{s}$, out of items that their uncertainty p does not exceed t_p ($p \leq t_p$). We note that uncertainty can be converted to probability (Section 2.1) of a score using $1 - p$. The interpretation of the predicted score and the respective uncertainty depends on the algorithm used for score distribution estimation. For example, in the case of CPMF [28], which estimates mean and variance of a normal distribution, predicted score corresponds to the mean and uncertainty corresponds to the variance.

Probability-of-Relevance Ranking (PRR): Given tuple score distribution $S(\sigma)$ for each tuple σ, PRR returns K tuples ranked according to the probability of having a score that is greater than t_s, computed as $Pr(S(\sigma) \geq t_s)$.

Borrowing from the probabilistic ranking literature [15], we present next three ranking semantics, the first is classified as score-based while the others are rank-based. It is worth noting that there are other rank-based approaches, see [15].

Expected Score: K tuples with the highest expected score, computed by $\sum_{s \in S(\sigma)} Pr(S(\sigma) = s) \cdot s$ for a discrete distribution.

Global Top-k: K tuples with the highest probability of being ranked within the range of ranks from 1 to K, $Pr\left(R\left(\sigma_i\right) \leq K\right)$.

U-Top$K$: K-subset A of U having largest probability to be the top-K answer over all possible worlds, $\text{argmax}_A(\sum_{w \in W_{(A,k)}} Pr\left(w\right))$ where $W_{(A,k)} \subseteq W$ is the set of possible worlds having A as the K-length prefix.

Table 3: An example of tuples ranking scores

Approach \ *Tuple*	σ_1	σ_2	σ_3
UBF ($t_p = 0.5$)	4.0	4.5	**5.0**
PRR ($t_s = 4.0$)	0.6	**0.8**	0.5
Expected Score	3.2	**3.8**	3.75
Global Top-k/U-TopK	0.068	0.4	**0.532**

EXAMPLE 3. *We next demonstrate the results of top-K approaches using the example presented in Table 1. Table 3 displays the respective scores for each approach assigned to each tuple, which determine the ranking for the top-*1 *result. Scores are calculated using tuple score probabilities. For instance, for σ_3, the expected score is computed by* $0.5 \cdot 0.1 + 3 \cdot 0.4 + 5 \cdot 0.5 = 3.75$*. The top-*1 *result comprises the tuple with the highest score, given in bold for each approach. For $K = 1$, the item scores of U-TopK and Global top-k are identical, as both methods are based on the same computation of the probability of a tuple to be ranked first. Note that different approaches lead to possibly different results, yielding different evaluation results.*

3.2 Defining Objectives

To incorporate quality measures into the problem objectives we define a user-specific *reference* list $A' \subseteq U_I$, which can be theoretically conceived to be the list of K items a user would deterministically rank highest from U_I, given the opportunity to explore all items. Practically, works in recommender systems, *e.g.*, [4], follow a *liberal* approach, where the reference list includes only items in the

candidate list U (Section 2.2) with which the user interacted (assuming implicitly all other items are ranked lower), and considering **all** of those items that their score according to some ground truth exceeds a predefined threshold. We consider this approach to be liberal since typically $|A'| > K$, allowing a higher chance for the recommendation to be successful. In contrast, we propose a *conservative* approach, where A' comprises the top-K highest ranked items from U_I, selected according to their ground truth score.

Given a quality measure M, we represent the metric result at K for evaluating answer A against the reference set A' as $M@K\,(A, A')$, where $A' = A'_C$ and $A' = A'_L$ for the conservative and liberal approaches, respectively. Alternatively, we write this more concisely as $M_C@K_A$ and $M_L@K_A$. If we do not specify a particular version, we default to the conservative approach.

We next establish that $M_L@K$ serves as an upper bound to $M_C@K$ for metrics M that increases when the reference list is enlarged. The correctness of Proposition 1 is immediate from the definitions of A'_C and A'_L.

PROPOSITION 1. *Given a top-K answer A, which can be evaluated using a metric M that satisfies the condition that if* $A'_1 \subseteq A'_2$, *then* $M@K\left(A, A'_1\right) \leq M@K\left(A, A'_2\right)$, *it follows that* $M_C@K_A \leq M_L@K_A$ *for any value of K.*

It is worth noting that Proposition 1 holds only when $A'_1 \subseteq A'_2$, which may not always be the case. We analyze empirically the relationships between the conservative and liberal approaches in Section 5.2.

We focus on three quality measures, namely $P@K$, $P@K(K)$ and $DCG@K$, which we present next. P@K is a *rank-based* measure that focuses on ranking quality and serves in measuring success of top-K results in identifying relevant items. It is concerned only with the set of tuples in the top-K query answer, rather than their positioning within the answer.

Given a top-K answer A and a reference list A', where $A' \in \{A'_C, A'_L\}$, *P@K of A with respect to* A' is given by

$$\text{P@}K\left(A, A'\right) = \frac{1}{K} \cdot \left|A \cap A'\right| \tag{1}$$

EXAMPLE 4. *Using the example presented in Table 1, assume that an oracle reveals the ground truth scores of items, one can compute the conservative P@K against* A'_C *created by sorting the items according to these scores. For* $K = 2$, $A'_C = \{\sigma_3, \sigma_2\}$, *the two items with the highest true score. Consider the top-2 answer of* $A = \{\sigma_1, \sigma_3\}$, *we have that* $P_C@K_A$=0.5.

With uncertain data we are not privy to the (deterministic) ground truth. Therefore, we analyze the expectation of the metric to quantify the quality of the ranking results of $P@K$. Let A_w be the correct answer to the top-K query with respect to w for each possible world $w \in W$. The *expected P@K of A* is given by:

$$E\left[\text{P@}K_A\right] = \sum_{w \in W} Pr\left(w\right) \cdot \text{P@}K\left(A, A_w\right) \tag{2}$$

In the absence of ground truth, we introduce P@$K(K)_A$, a special case of $P@K_A$, which is the probability that P@$K = 1$.

$$\text{P@}K(K)_A = Pr\left(\text{P@}K_A = 1\right) \tag{3}$$

P@$K(K)_A$ is in fact the sum of probabilities of all worlds for which $A \subseteq A'$.

EXAMPLE 4 (CONT.). *For the previously top-2 answer of* $A = \{\sigma_1, \sigma_3\}$, *we demonstrate the computation of the* P@$K(K)_A$ *metric using the possible worlds presented in Table 2. We compute* P@$K(K)_A$ *by summing the probabilities of the worlds in which the set A′, derived from the world for* $K = 2$ *according to the conservative or liberal approach, contains the set A. For the conservative method, A′ is defined to be the top-2 answer in each world. The top-2 set equals A in world* w_2, w_3, w_8, *and* w_9. *Thus, the probability that A has (conservative) precision of* 1 *is* $0.032 + 0.04 + 0.048 + 0.06 = 0.18$.

For a rank-based non-binary quality measure we use discounted cumulative gain (*DCG*), which was shown to be an effective metric to measure user satisfaction [16]. Given a top-K answer A and reference list A', where $A' \in \{A'_C, A'_L\}$, *DCG@K of A with respect to* A' is defined as follows.

$$DCG@K\left(A, A'\right) = \sum_{i=1}^{K} \frac{g(\sigma_{(i)}, A')}{log(i+1)} \tag{4}$$

where $g(\sigma_{(i)}, A')$ is defined to be the gain from tuple $\sigma_{(i)}$ (the tuple that is ranked i-th in A) based on scores in A'. Typically, $g(\sigma_{(i)}, A') = 2^{s_{A'}(\sigma_{(i)})} - 1$ where $s_{A'}(\sigma_{(i)})$ is the deterministic score of this tuple in A'.

EXAMPLE 4 (CONT.). *Consider again* $A = \{\sigma_1, \sigma_3\}$, *where* σ_1 *is ranked first followed by* σ_3. *The computation of the DCG@K metric using the conservative approach is*

$$DCG_C@K_A = \frac{2^5 - 1}{log(2+1)} \tag{5}$$

The reference list A'_C *includes* σ_3 *and* σ_2 *with the scores of* 5 *and* 4.5, *respectively.* σ_1 *is not included in* A'_C, *and therefore it does not contribute to the* $DCG_C@K_A$ *computation (its score is undefined).*

The *expected DCG@K of A* is given by:

$$E\left[\text{DCG@}K_A\right] = \sum_{w \in W} Pr\left(w\right) \cdot \text{DCG@}K\left(A, A_w\right) \tag{6}$$

3.3 Optimizing Top-K Query Response

Selecting the appropriate top-K semantics from a broad range of options is essential when optimizing the problem objectives to ensure user satisfaction. We now establish a connection between uncertain top-K query semantics and the quality measures introduced in Section 3.2, namely expected P@K, P@$K(K)$, and expected DCG@K.

3.3.1 Maximizing $E\left[P@K\right]$.

For expected P@K, we now characterize (Theorem 1) a top-K query answering approach that, on databases with uncertain scores, optimizes the outcome quality. Specifically, a top-K query answer of K tuples with the highest probability of being at the top-K answer maximizes $E\left[\text{P@}K\right]$. This set was proposed as the Global Top-k semantics [31] (Section 3.1).[1]

THEOREM 1. *Given a candidate set U, for* $A = \{\sigma_1, \ldots, \sigma_K\} \subseteq U$ *such that* $\forall(\sigma \in A,\ \sigma' \in U - A)$, $Pr\left(R\left(\sigma\right) \leq K\right) \geq Pr\left(R\left(\sigma'\right) \leq K\right)$, *A maximizes the expected P@K over all K-subsets of U.*

[1] There is an interesting relationship between the quality measure P@K and the symmetric difference measure, introduced by Li and Deshpande [19]. In fact, maximizing one results in minimizing the other.

PROOF. Combining the definitions of $\text{P@}K\,(A, A')$ (Eq. 1) and $E\,[\text{P@}K_A]$ (Eq. 2), we obtain:

$$E\,[\text{P@}K_A] = \sum_{w \in W} Pr\,(w) \cdot \frac{1}{K} \cdot |A \cap A_w| \tag{7}$$

Using the definition of cardinality of intersection of two sets A and A_w, we rewrite Eq. 7 as

$$E\,[\text{P@}K_A] = \sum_{w \in W} Pr\,(w) \cdot \frac{1}{K} \cdot \left[\sum_{i=1}^{K} \delta\,(\sigma_i \in A, \sigma_i \in A_w)\right] \tag{8}$$

where δ is an indicator function, returning 1 if $\sigma_i \in A$ belongs to the set A_w and 0 otherwise.

From Eq. 8, the commutative property of multiplication and by interchanging the order of summations, we derive

$$E\,[\text{P@}K_A] = \frac{1}{K} \cdot \sum_{i=1}^{K} \sum_{w \in W} \delta\,(\sigma_i \in A, \sigma_i \in A_w) \cdot Pr\,(w) \tag{9}$$

By definition,

$$Pr\,(R\,(\sigma_i) \leq K) = \sum_{w \in W} \delta\,(\sigma_i \in A_w) \cdot Pr\,(w) \tag{10}$$

Combining Eqs. 9 and 10 yields:

$$E\,[\text{P@}K_A] = \frac{1}{K} \cdot \sum_{i=1}^{K} Pr\,(R\,(\sigma_i) \leq K) \tag{11}$$

$1/K$ is constant. Therefore, to maximize $E\,[\text{P@}K]$ we need to maximize $Pr\,(R\,(\sigma_i) \leq K)$ for each $\sigma_i \in A$, concluding the proof. □

3.3.2 Maximizing P@K(K). Theorem 2 connects $\text{P@}K(K)$ and *U-Topk* semantics (Section 3.1).

THEOREM 2. *Given a candidate set U with probabilistic scores and a top-K query, $A^* = \arg\max_{A \subseteq U} P@K(K)_A$ iff $A^* = \text{argmax}_A(\sum_{w \in W_{(A,k)}} Pr\,(w))$ where $W_{(A,k)} \subseteq W$ is the set of possible worlds having A as the K-length prefix.*

PROOF. Let $A^* \subseteq U$ such that $A^* = \arg\max_{A \subseteq U} \text{P@}K(K)$. According to Eq. 3,

$$\text{P@}K(K) \stackrel{\triangle}{=} Pr\,(\text{P@}K_{A^*} = 1) =$$

$$\sum_{w \in W} Pr\,(w) \cdot \delta(\text{P@}K(A^*, A_w) = 1) =$$

$$\sum_{w \in W} Pr\,(w) \cdot \delta(A^* = A_w)$$

$\delta(A^* = A_w) = 1$ whenever A^* is the K-length prefix of world w, which leads to the conclusion that

$$\text{P@}K(K) = \sum_{w \in W_{(A^*,k)}} Pr\,(w)$$

Therefore, $A^* = \arg\max_{A' \subseteq U} \text{P@}K(K)$ iff $A^* = \arg\max_{A' \subseteq U}(\sum_{w \in W_{(A,k)}} Pr\,(w))$, which concludes our proof. □

3.3.3 Maximizing $E\,[DCG@K]$. Next, we show a top-K query answering approach that, on databases with uncertain scores, optimizes the expected $\text{DCG@}K$. Theorem 3 below characterizes a top-K query answer with K tuples that corresponds to an adaptation of the Expected Score [3] semantics (Section 3.1).

THEOREM 3. *Given a candidate set U, for $A = (\sigma_1, \ldots, \sigma_K) \subseteq U$, ordered from left to right, such that $\forall(i \in \{1, \ldots, K\}, j \in \{1, \ldots, K\} : i < j), E\,[S\,(\sigma_i)] \geq E\left[S\,(\sigma_j)\right]$ and $\forall(\sigma \in A, \sigma' \in U - A), E\,[S\,(\sigma)] \geq E\,[S\,(\sigma')]$, A maximizes the $E\,[DCG@K_A]$ over all K-subsets of U.*

PROOF. Recall that $\text{DCG@}K$ and $E\,[\text{DCG@}K_A]$ are given by Eqs.4 and6. Therefore,

$$E\,[\text{DCG@}K_A] = \sum_{w \in W} Pr\,(w) \cdot \sum_{i=1}^{K} \frac{g(\sigma_{(i)}, A_w)}{log(i+1)} \tag{12}$$

By reordering of summations and using independence of terms from summation, we get

$$\begin{aligned} E\,[\text{DCG@}K_A] &= \sum_{i=1}^{K} \sum_{w \in W} Pr\,(w) \cdot \frac{g(\sigma_{(i)}, A_w)}{log(i+1)} \\ &= \sum_{i=1}^{K} \frac{1}{log(i+1)} \sum_{w \in W} Pr\,(w) \cdot g(\sigma_{(i)}, A_w) \end{aligned} \tag{13}$$

Finally, we observe that $\sum_{w \in W} Pr\,(w) \cdot g(\sigma_{(i)}, A_w)$ is the expectation of $g(\sigma_{(i)}, .)$ over worlds W, yielding

$$E\,[\text{DCG@}K_A] = \sum_{i=1}^{K} \frac{1}{log(i+1)} \cdot E\left[g(\sigma_{(i)}, .)\right] \tag{14}$$

$E\,[\text{DCG@}K_A]$ is maximized whenever A consists of K tuples with the highest value of $E\left[g(\sigma_{(i)}, .)\right]$. Assuming that $g(\sigma_{(i)}, A') = 2^{s_{A'}(\sigma_{(i)})} - 1$ where $s_{A'}(\sigma_{(i)})$ is the deterministic score of this tuple in A', the K tuples in A are those with the highest expected score. □

4 ALGORITHM EXECUTION & PERFORMANCE ASSESSMENT

Given the ranking problem formulation (Section 3), an execution method (marked as *Algorithm Execution* in Figure 1) is performed over the database to produce the top-K answer according to the ranking model. Then, the generated top-K answer is being assessed (*Performance Assessment*).

We propose a method for computing the distribution of tuples across ranks in Section 4.1. This method offers a basis for the computation of a wide range of probabilistic ranking semantics for top-K recommendation that use rank distribution, some of which are presented in Section 3.1. In Section 4.2, we show that the expectation of the metric $\text{P@}K$ can be efficiently computed using the algorithm suggested in Section 4.1.

4.1 The RankDist Operator

RankDist is a basic operator for computing the query result of multiple semantics, and for assessing rank-based evaluation metrics, such as the expected $\text{P@}K$ (Eq. 2). The RankDist operator receives as input a universe U, its subset $A \subseteq U$, and $K \leq |U|$, and computes the probability of each tuple $\sigma \in A$ to be ranked at each of the top-K ranks within U. Therefore, the output of RankDist is a relation

RankDist(tupleID, Rank, PR) such that tupleID is a unique identifier of a tuple $\sigma \in A$, Rank (ρ) ranges from 1 (with the interpretation that $\sigma_{(1)}$ is the element ranked at position 1) to K, and PR is the probability that the rank of σ is ρ in universe U, that is,[2]

$$\mathsf{PR}(\sigma, \rho, U) = Pr\left(R(\sigma) \overset{U}{=} \rho\right) \tag{15}$$

Throughout this work we shall use a technique of *universe transformation*, which involves changing the content of a universe (*e.g.*, tuples removal) and computing probabilities in a different, more convenient universe.

4.1.1 Computing RankDist. Algorithm 1 details an efficient algorithm for computing RankDist. We define $P_i = \left\{p_{i,j,k}\right\}$ to be $N \times K$ matrix (for $|U| = N$) where each $p_{i,j,k}$ is devoted to capture the probability of tuple σ_i to be ranked at the k-th position in the (selected by the algorithm) sub-universe of j tuples.

Due to tuple scores uncertainty, $p_{i,j,k}$ is actually a vector $p_{i,j,k} = \left\langle p_{i,j,k,1}, ..., p_{i,j,k,|S(\sigma_i)|}\right\rangle$, entries of which capture the contribution of a single possible score of σ_i to the rank of that tuple.

For ease of presentation, we present a simplified version of RankDist, ignoring score ties. Equal scores can be handled through deterministic or stochastic tie-breaking rules (see Section 4.1.2).

At the beginning of the algorithm (Lines 2-5), we set $p_{i,j,k,l} = 0$ for all i, j, k and l where $j \neq 1$ and $k \neq 1$. For $j = k = 1$, $p_{i,1,1,l}$ is set to the probability that σ_i is assigned with score s_l. Then, for each tuple $\sigma_i \in A$, we consider all tuples in $U \backslash \{\sigma_i\}$ in some arbitrary order. $\sigma_{(j)}$ denotes the j-th tuple according to that order, with $\sigma_{(1)} = \sigma_i$. The algorithm introduces each tuple in U in turn and computes the probability of σ_i to be ranked at any of the positions $\{1, ..., K\}$, as long as the position is not greater than the number of the introduced tuples j. This iterative introduction of tuples can be viewed as an incremental insertion of tuples into the universe. Alternatively, one can view this iterative process as an incremental expansion of the set of tuples that are "allowed" to compete with σ_i on a ranked position in the universe. At the end of the algorithm, we can compute for each σ_i its probability to be ranked at each of the K relevant ranks.

EXAMPLE 5. *Consider the process of computing the rank distribution for σ_1 in our toy example. In the first iteration (Line 4), $p_{1,1,1,1} = 0.4, p_{1,1,1,2} = 0.6$ since tuple σ_1 is ranked first in a world that contains only itself, whether its score is 2 or 4 and therefore the probability is exactly the probability of being assigned 2 (or 4). $p_{1,2,1}$ is the vector of probabilities of σ_1 to remain at the top position in a universe with one more tuple, say σ_2. This requires σ_1 to win over σ_2, which is only possible if σ_1's score is 2 or 4 and σ_2's score is 1. Therefore, $p_{1,2,1,1} = 0.4 \cdot 0.2 = 0.08$, and $p_{1,2,1,2} = 0.6 \cdot 0.2 = 0.12$.*

The heart of the algorithm is in computing $p_{i,j,k,l}$ (lines 11-12), explained next. When a new tuple $\sigma_{(j)}$ is introduced to the current universe of tuples, the new universe contains j tuples (σ_i and $\sigma_{(2)}, ..., \sigma_{(j)}$); we shall denote this universe as $U[j]$. We partition the event $R(\sigma_i) \overset{U[j]}{=} \rho$ into mutually exclusive events $s \wedge R(\sigma_i) \overset{U[j]}{=} \rho$, based on the possible score values $s \in \sigma_i$. Using our short-hand notation (Section 3), Eq. 15 can be rewritten as

[2]We explicitly specify a universe whenever it is not clear from the context.

Algorithm 1 RankDist

```
1: input: U, A, K
2: for all p_{i,j,k,l} do
3:    p_{i,j,k,l} = 0
4:    p_{i,1,1,l} = Pr(S(σ_i) = s_l)
5: end for
6: for all σ_i ∈ A do
7:    rearrange tuple order with σ_(1) = σ_i
8:    for all j = 2, ..., |U| do
9:       for all k = 1, ..., min(j, K) do
10:         for all s_l ∈ S(σ_i) do
11:            p_{i,j,k,l} = Pr(S(σ_j) < s_l) * p_{i,j-1,k,l} +
12:                          Pr(S(σ_j) > s_l) * p_{i,j-1,k-1,l}
13:         end for
14:      end for
15:   end for
16:   for all k = 1, ..., K do
17:      INSERT INTO RankDist VALUES (i, k, Σ_{l=1}^{|S(σ_i)|} p_{i,|U|,k,l})
18:   end for
19: end for
```

$$\mathsf{PR}(\sigma_i, \rho, U[j]) = \sum_{s \in S(\sigma_i)} Pr(S(\sigma_i) = s, R(\sigma_i) \overset{U[j]}{=} \rho) = \sum_{s \in S(\sigma_i)} Pr(s, R(\sigma_i) \overset{U[j]}{=} \rho) \tag{16}$$

Each of these events is partitioned into mutually exclusive events of the form $s \wedge R(\sigma_i) \overset{U[j]}{=} \rho \wedge R(\sigma_i) \overset{U[j-1]}{=} \rho'$, based on the possible ranking of σ_i in $U[j-1]$. Note that the score assignment is the same in both universes and therefore Eq. 16 can be rewritten as

$$\mathsf{PR}(\sigma_i, \rho, U[j]) = \sum_{\rho'=1}^{j-1} \sum_{s \in S(\sigma_i)} Pr\left(s, R(\sigma_i) \overset{U[j]}{=} \rho, R(\sigma_i) \overset{U[j-1]}{=} \rho'\right) \tag{17}$$

We observe that with the introduction of $\sigma_{(j)}$, tuple σ_i can be ranked at the ρ-th position in only two cases—if it has already been at the ρ-th position when $\sigma_{(j)}$ was introduced, and if σ_i was at the $(\rho-1)$-th position. In other words, with the introduction of a single new tuple, σ_i cannot move up in its ranking and cannot be pushed down more than one position. Therefore, we have

$$Pr\left(s, R(\sigma_i) \overset{U[j]}{=} \rho, \rho(\sigma_i) \overset{U[j-1]}{=} \rho'\right) = 0$$

for all s and $\rho' \notin \{\rho - 1, \rho\}$. Therefore, we rewrite Eq. 17 as

$$\mathsf{PR}(\sigma_i, \rho, U[j]) = \sum_{s \in S(\sigma_i)} Pr\left(s, R(\sigma_i) \overset{U[j]}{=} \rho, R(\sigma_i) \overset{U[j-1]}{=} \rho\right) + \sum_{s \in S(\sigma_i)} Pr\left(s, R(\sigma_i) \overset{U[j]}{=} \rho, R(\sigma_i) \overset{U[j-1]}{=} \rho - 1\right) \tag{18}$$

and using the chain rule transformation we have

$$\sum_{s\in S(\sigma_i)} Pr\left(R(\sigma_i) \overset{U[j]}{=} \rho \mid s, R(\sigma_i) \overset{U[j-1]}{=} \rho\right) \cdot Pr\left(s, R(\sigma_i) \overset{U[j-1]}{=} \rho\right) + \sum_{s\in S(\sigma_i)} Pr\left(R(\sigma_i) \overset{U[j]}{=} \rho \mid s, R(\sigma_i) \overset{U[j-1]}{=} \rho-1\right) \cdot Pr\left(s, R(\sigma_i) \overset{U[j-1]}{=} \rho-1\right) \tag{19}$$

In the case σ_i maintains its rank from $U[j-1]$ to $U[j]$, it needs to be ranked higher than $\sigma_{(j)}$. Since σ_i's score within the summation is fixed to s, we should find the probability of σ_i "winning"over $\sigma_{(j)}$ with that score s, that is,

$$Pr\left(R(\sigma_i) \overset{U[j]}{=} \rho \mid s, R(\sigma_i) \overset{U[j-1]}{=} \rho\right) = Pr\left(S(\sigma_j) < s\right) = \sum_{s'\in S(\sigma_j)} Pr(s') \cdot \delta(s' < s) \tag{20}$$

In the second case, σ_i drops a notch, which means it is ranked lower than $\sigma_{(j)}$, that is,

$$Pr\left(R(\sigma_i) \overset{U[j]}{=} \rho \mid s, R(\sigma_i) \overset{U[j-1]}{=} \rho-1\right) = Pr\left(S(\sigma_j) > s\right) = \sum_{s'\in S(\sigma_j)} Pr(s') \cdot \delta(s' > s) \tag{21}$$

Finally, we can embed Eqs. 20 and 21 in Eq. 19, getting

$$\text{PR}(\sigma_i, \rho, U[j]) = \sum_{s\in S(\sigma_i)} Pr\left(S(\sigma_j) < s\right) \cdot Pr\left(s, R(\sigma_i) \overset{U[j-1]}{=} \rho\right) + \sum_{s\in S(\sigma_i)} Pr\left(S(\sigma_j) > s\right) \cdot Pr\left(s, R(\sigma_i) \overset{U[j-1]}{=} \rho-1\right) \tag{22}$$

The terms of these two summations are computed in lines 11-12 as

$$Pr\left(s, R(\sigma_i) \overset{U[j-1]}{=} \rho\right) = p_{i,j-1,\rho,l} \tag{23}$$

$$Pr\left(s, R(\sigma_i) \overset{U[j-1]}{=} \rho-1\right) = p_{i,j-1,\rho-1,l} \tag{24}$$

We define $p_{i,j-1,0,l} = 0$ for all i, j, and l. The outcome of the algorithm, $\text{PR}(\sigma_i, \rho, U)$, is computed in line 17 as

$$\text{PR}(\sigma_i, \rho, U) = \sum_{l=1}^{|S(\sigma_i)|} p_{i,|U|,\rho,l} \tag{25}$$

Table 4: An example of the algorithm execution

↓ j k →	1	2
1	{(2, 0.4), (4, 0.6)}	
2	{(2, 0.08), (4, 0.12)}	{(2, 0.32), (4, 0.48)}
3	{(2, 0.008), (4, 0.06)}	{(2, 0.104), (4, 0.3)}

EXAMPLE 6. *Table 4 demonstrates the algorithm execution on the universe in Table 1 by depicting the outcome of the computation of the top-2 rank distribution of* σ_1. *For instance, consider the* $(3, 2)$ *entry in the matrix, representing the* σ_1*'s probability of being ranked second after the introduction of* σ_3. *For the score of* 2, *the probability of* 0.104 *is a combination of the probability of being ranked first before (*0.08*) times the probability of losing to* σ_3 *(*0.9, *sum of the probability of assigning a score of either* 3 *or* 5 *to* σ_3*), and the probability of being ranked second before (*0.32*) times the probability of winning* σ_3 *(*0.1, *the probability of assigning a score of* 0.5 *to* σ_3*).*

THEOREM 4. *Algorithm 1 correctly computes the rank distribution of tuples in* U.

The (omitted here) proof of Theorem 4 is by induction on the number of tuples in the database, showing the equivalence of the outcome of the incremental computations of Eq. 15 and the summation terms in Eq. 22.

Using deterministic tie-breaking, $Pr\left(S\left(\sigma_j\right) < s\right)$ (Eq. 20) and $Pr\left(S\left(\sigma_j\right) > s\right)$ (Eq. 21) can be computed once and used throughout the algorithm. Given an upper bound c on the number of possible scores per tuple, this computation is done in $O\left(c^2N^2\right)$ and requires $O\left(cN^2\right)$ space complexity. The algorithm itself is dominated by the four loops in lines 6-15. The overall run-time complexity of computing RankDist for a single tuple is $O(cNK)$. It is worth noting that the algorithm can be easily parallelized so that RankDist of all tuples in A can be computed in parallel.

4.1.2 *Computing RankDist with Stochastic Tie-breaking.*

In this section, we provide an extension of the RankDist algorithm presented in Section 4.1, incorporating stochastic tie-breaking when comparing two tuples that share the same score. In this tie breaking approach, we allow both tuples to be winners, each with an equal probability. When using such a tie-breaking rule, some probabilities (see Eqs. 20 and 21) depend not only on the tuples themselves, but also on other tuples, which requires a different computation method for RankDist. Algorithm 2 details the algorithm for computing RankDist with stochastic tie-breaking, which we will next explain.

In this version, we extend P_i to be $P_i = \left\{p_{i,t,j,k,l}\right\}$, where $p_{i,t,j,k,l}$ represents the probability of tuple σ_i being ranked at the k-th position with a possible score $s_l \in S(\sigma_i)$ in the (selected by the algorithm) sub-universe of j tuples, with t ties occurring on this score with other tuples. We are interested in computing the rank distribution of the tuples in the universe U over K places. Since there can be at most $N-1$ ties, as long as $t \le j$ (recall that $|U| = N$) for a tuple with a score, P_i is $N \times N \times K \times |S(\sigma_i)|$ matrix.

At the beginning of the algorithm (lines 2-5), we set $p_{i,t,j,k,l} = 0$ for all i, t, j, k and l where $t \neq 0$, $j \neq 1$ and $k \neq 1$. For $j = k = 1$ and $t = 1$, $p_{i,0,1,1,l}$ is set to the probability that σ_i is assigned with score s_l. Then, for each tuple $\sigma_i \in A$, we consider all tuples in $U \backslash \{\sigma_i\}$ in some arbitrary order. $\sigma_{(j)}$ denotes the j-th tuple according to that order, with $\sigma_{(1)} = \sigma_i$. The algorithm introduces each tuple in U in turn and computes the probability of σ_i to be ranked at any of the positions $\{1, ..., K\}$, as long as the position is not greater than the number of the introduced tuples j. This iterative introduction of tuples can be viewed as an incremental insertion of tuples into the universe. Alternatively, one can view this iterative process as an incremental expansion of the set of tuples that are "allowed " to compete with σ_i on a ranked position in the universe.

When a new tuple $\sigma_{(j)}$ is introduced into the current universe of tuples, denoted as $U[j]$, the new universe contains j tuples (σ_i and $\sigma_{(2)}, ..., \sigma_{(j)}$). We partition the event $R(\sigma_i) \overset{U[j]}{=} \rho$ into mutually exclusive events $s \wedge R(\sigma_i) \overset{U[j]}{=} \rho$, based on the possible score values $s \in S(\sigma_i)$. Similar to the RankDist version presented in Section 4.1, the computation is based on the rank distribution when the universe

includes $j-1$ tuples, in which a tuple can change its rank by at most 1 place. Then, we further partition the event that σ_i will be ranked at the k place with a score s_l by dividing it based on the interaction of tuple scores with $\sigma_{(j)}$. If there are no ties between s_l and $\sigma_{(j)}$ scores, we divide the event into cases where s_l is greater than $S(\sigma_{(j)})$ and where $S(\sigma_{(j)})$ is greater than s_l. In the former case, which occurs with probability of $Pr(S(\sigma_j) < s_l)$, σ_i will be ranked at the k-th position in $U[j]$ if it maintained the same rank in $U[j-1]$, which occurs in probability of $p_{i,t,j-1,k,l}$. Otherwise, if $S(\sigma_{(j)})$ is greater than s_l (with probability of $Pr(S(\sigma_j) > s_l)$), σ_i will be ranked at the k-th position in $U[j]$ only if it ranked one position higher in $U[j-1]$ (with probability of $p_{i,t,j-1,k-1,l}$). When σ_i has the same s_l score as $\sigma_{(j)}$, σ_i retains its k-th rank from $U[j-1]$ to $U[j]$ with t ties if there where $t-1$ ties before $\sigma_{(j)}$ was introduced.

Next, we initialize $p^i_{t,k,l}$ to store the probability of tuple σ_i across number of ties t, k and score s_l, after all the tuples $\sigma_{(j)}$ have been considered, namely when $j = |U|$.

The computed $p^i_{t,k,l}$ reflects the probability that σ_i with score s_l is ranked at the k-th place with t ties. Given these t ties with other tuples, σ_i (with the respective score) can be ranked at the successive t ranks of $k, \ldots, k+t$, with a probability equal to dividing $p^i_{t,k,l}$ by the number of places it spans, which is $t+1$. In lines 20-24, we initialize $p^i_{k,l}$, which represents the probability that σ_i with score s_l will be ranked at the k-th place, based on probabilities stored in $p^i_{t,k',l'}$ where $k' = k$ and $l' = l$. Subsequently, in lines 25-33, we augment $p^i_{k,l}$ with probabilities derived from the spanning resulted from the ties. When $t > 1$ ties exist for tuple σ_i with score s_l, we shift the probabilities $p^i_{t,k,l}$ to span over $k, \ldots, k+t$ places. We define k_s to be the start index for which we span probabilities over the next t ranks. Let d_k be the distance of the updated k from k_s. Since our focus is on the rank distribution across K places, d_k ranges between 1 to the minimum of t (the number of ties) and $K - k_s$ (the lowest relevant rank). For each possible score s_l, we update the probability $p^i_{k_s+d_k,l}$ to include $\frac{1}{t+1} * p^i_{t,k_s,l}$.

At the end of the algorithm (lines 34-36), we can compute for each σ_i its probability to be ranked at each of the K relevant ranks.

4.1.3 RankDist *and Certain Joint Probabilities.*

Using an extension of Algorithm 1, we can also compute the joint probability of two tuples ranked in two consecutive positions

$$Pr\left(R\left(\sigma_i\right) \overset{U}{=} \rho, R\left(\sigma_j\right) \overset{U}{=} \rho + 1\right) \tag{26}$$

for the compound events $R(\sigma_i) = \rho \wedge R(\sigma_j) = \rho + 1$, and this for all pairs of tuples $\sigma_i, \sigma_j \in U$, and all possible ranks ρ. For instance, in our running example we have $Pr(R(\sigma_1) = 1, R(\sigma_2) = 2) = 0.02$. For each two scores s of σ_i and s' of σ_j we compute

$$Pr\left(s, s', R\left(\sigma_i\right) \overset{\{\sigma_i,\sigma_j\}}{=} 1, R\left(\sigma_j\right) \overset{\{\sigma_i,\sigma_j\}}{=} 2\right) \tag{27}$$

If $s' > s$, then the probability that σ_i and σ_j are ranked first and second is 0. Otherwise, if $s' < s$, in the universe consisting of only σ_i and σ_j this probability is given by $Pr\left(S\left(\sigma_i\right) = s\right) \cdot Pr\left(S\left(\sigma_j\right) = s'\right)$, the probability of σ_i and σ_j to be assigned s and s', respectively. A newly added tuple is ranked higher than both σ_i and σ_j if its score is higher than that of σ_i. It is ranked lower than both if its score is lower than that of σ_j. Since σ_i and σ_j should be positioned in consecutive ranks, we do not compute the probability of the new tuple to be positioned in between them.

Algorithm 2 `RankDist- Stochastic Tie-breaking`

1: **input**: U, A, K
2: **for all** $p_{i,t,j,k,l}$ **do**
3: $p_{i,t,j,k,l} = 0$
4: $p_{i,0,1,1,l} = Pr\left(S\left(\sigma_i\right) = s_l\right)$
5: **end for**
6: **for all** $\sigma_i \in A$ **do**
7: rearrange tuple order with $\sigma_{(1)} = \sigma_i$
8: **for all** $j = 2, \ldots, |U|$ **do**
9: **for all** $k = 1, \ldots, min(j, K)$ **do**
10: **for all** $s_l \in S\left(\sigma_i\right)$ **do**
11: $p_{i,t,j,k,l} = Pr(S(\sigma_j) < s_l) * p_{i,t,j-1,k,l}$ +
12: $Pr(S(\sigma_j) > s_l) * p_{i,t,j-1,k-1,l}$ +
13: $Pr(S(\sigma_j) = s_l) * p_{i,t-1,j-1,k,l}$
14: **end for**
15: **end for**
16: **end for**
17: **for all** $p_{i',t,j,k,l}$ where $i' = i$ **do**
18: $p^i_{t,k,l} = p_{i',t,|U|,k,l}$
19: **end for**
20: **for all** $k = 1, \ldots, K$ **do**
21: **for all** $s_l \in S\left(\sigma_i\right)$ **do**
22: $p^i_{k,l} = \sum_{t=0}^{N-1} \frac{1}{t+1} * p^i_{t,k,l}$
23: **end for**
24: **end for**
25: **for all** $t = 1, \ldots, N-1$ **do**
26: **for all** $k_s = 1, \ldots, K-1$ **do**
27: **for all** $d_k = 1, \ldots, min(t, K - k_s)$ **do**
28: **for all** $s_l \in S\left(\sigma_i\right)$ **do**
29: $p^i_{k_s+d_k,l} = p^i_{k_s+d_k,l} + \frac{1}{t+1} * p^i_{t,k_s,l}$
30: **end for**
31: **end for**
32: **end for**
33: **end for**
34: **for all** $k = 1, \ldots, K$ **do**
35: INSERT INTO RankDist VALUES $\left(i, k, \sum_{l=1}^{|S(\sigma_i)|} p^i_{k,l}\right)$
36: **end for**
37: **end for**

In Section 4.2 we use the likelihood of the joint events (Eq. 26) to compute the expected P@K. The algorithm for computing these probabilities is almost identical to Algorithm 1 and thus we omit its detailed description here. Considering its complexity, when computing the probabilities for a pair of tuples and all the relevant ranks ρ, we perform $O(KN)$ steps, for each we compute $O\left(c^2\right)$ computations, one for each pair of score values. The total run-time complexity of computing all the probabilities as in Eq. 26 for a pair of tuples is $O\left(c^2 NK\right)$, a factor of c beyond the complexity of Algorithm 1.

4.2 Computing Expected Quality

To analyze the performance of semantics with respect to quality measures, we show how to compute the **expected** quality of a **given**

top-K query answer. Equipped with methods for computing Eqs. 15 and 26, we first show how to compute the various components of P@K's expectation (sections 4.2.1 and 4.2.2) and then we discuss the computation of expected P@K (Section 4.2.3). We note that expected DCG@K can be computed directly using Eq. 14 (Section 3.3.3).

Given a tuple set $A = \{\sigma_1, \sigma_2, ..., \sigma_K\}$, $\text{P@}K_A$ is a random variable over $\{0, \frac{1}{K}, \frac{2}{K}, \ldots, 1\}$. For $1 \leq i \leq K$, we use the notation

$$\text{P@}K(i) \triangleq Pr\left(\text{P@}K_A = \frac{i}{K}\right) \tag{28}$$

To illustrate P@$K(\cdot)$, consider the following (highly inefficient) computation method. Given tuple set A, we scan each possible world w in turn, identify A_w, the top-K result in w, and compute $\text{P@}K(A, A_w)$. Once all worlds are scanned, we can compute P@$K(i)$ as the sum of probabilities of all worlds $w \in W$ such that $\text{P@}K(A, A_w) = \frac{i}{K}$, that is, $\text{P@}K(i) = \sum_{w \in W} Pr(w) \cdot \delta(\text{P@}K(A, A_w) = \frac{i}{K})$.

EXAMPLE 7. *In our running example, let $A = \{\sigma_1, \sigma_3\}$, be the outcome of some method for the top-2 query. The precision of A in world w_1 is 0.5 since (from A) only σ_1 is present in the top-2 subset of this world. Similarly, the precision of A in worlds w_2, w_3, and w_4 is 1, 1, and 0.5, respectively. Computing this over all worlds, we obtain that P@$K(2)$, the probability of A containing both correct items, thus having precision of 1 is 0.18, P@$K(1)$, the probability of A containing one of two correct items, thus having precision of 0.5 is 0.82 and P@$K(0)$, missing both items, is 0.*

Scanning all possible worlds is obviously infeasible. We now show an efficient method for computing P@$K(\cdot)$. We start with a special case of P@$K(K)$, the probability of maximum precision. We proceed with the partial success of P@$K(i)$ for $0 < i < K$. Computing P@$K(0)$ can be done in a fashion similar to this for P@$K(K)$, but this term is not useful for our purpose of computing expected P@K, since for the computation of expectation we multiply P@$K(0)$ by 0.

4.2.1 Computing P@$K(K)$. By definition, P@$K(K)$ is the probability of $A = \{\sigma_1, \sigma_2, ..., \sigma_K\}$ to be the intended top-K query result:

$$\text{P@}K(K) = Pr\left(R(\sigma_1) \leq K, \ldots, R(\sigma_K) \leq K\right) \tag{29}$$

Note that, in any world providing A the precision of 1, *some* tuple $\sigma_m \in A$ is ranked K-th within the sub-universe A, and that tuple is ranked *first* within the *almost* complementary sub-universe obtained from U by removing from it all tuples of A *but* σ_m (denoted as U^{+m}_{-A}). Thus, we can partition the event addressed by Eq. 29 into K mutual excluding events and rewrite Eq. 29 as

$$\text{P@}K(K) = \sum_{m=1}^{K} Pr\left(R(\sigma_m) \overset{U^{+m}_{-A}}{=} 1, R(\sigma_m) \overset{A}{=} K\right) \tag{30}$$

Similarly to computing RankDist, we can rewrite the right-hand side of Eq. 30 to consider the possible scores of σ_m, obtaining

$$\begin{aligned}&\sum_{m=1}^{K} \sum_{s \in S(\sigma_m)} Pr\left(s, R(\sigma_m) \overset{U^{+m}_{-A}}{=} 1, R(\sigma_m) \overset{A}{=} K\right) = \\ &\sum_{m=1}^{K} \sum_{s \in S(\sigma_m)} Pr\left(R(\sigma_m) \overset{U^{+m}_{-A}}{=} 1 \mid s, R(\sigma_m) \overset{A}{=} K\right) \cdot Pr\left(s, R(\sigma_m) \overset{A}{=} K\right)\end{aligned} \tag{31}$$

Given a score of σ_m, the rank of σ_m in U^{+m}_{-A} is independent of the rank of σ_m in A, since none of the other tuples in A are also present in U^{+m}_{-A}. Note that this is not the case if we do not fix the score of σ_m– the same score that brought σ_m to the K-th position in A may prevent it from reaching the first place in U^{+m}_{-A}. Exploiting this independence yields

$$\text{P@}K(K) = \sum_{m=1}^{K} \sum_{s \in S(\sigma_m)} Pr\left(R(\sigma_m) \overset{U^{+m}_{-A}}{=} 1 \mid s\right) \cdot Pr\left(s, R(\sigma_m) \overset{A}{=} K\right) \tag{32}$$

where $Pr(s, R(\sigma_m) \overset{A}{=} K)$ can be computed with RankDist. Computing $Pr(R(\sigma_m) \overset{U^{+m}_{-A}}{=} 1 \mid s)$ can be done by a single iteration over the tuples in the universe, where we compute the probability that σ_m wins each introduced tuple. Computing those probabilities for all tuples in A takes $O\left(c^2K^3\right)$ and $O\left(c^2NK\right)$, respectively. The overall runtime complexity of computing P@$K(K)$ is therefore $O\left(c^2NK\right)$.

4.2.2 Computing P@$K(i)$. We now turn our attention to the computation of Eq. 28 for $0 < i < K$. In each world w providing A with the precision $\frac{i}{K} < 1$, some $K - i$ tuples from A do not appear in A_w. We now extend our observation from Section 4.2.1 and examine hypothesis corresponding to **pairs** of boundary tuples $\sigma_m, \sigma_n \in A$ with σ_m being the lowest-ranked tuple from A present in A_w, and σ_n being the highest-ranked tuple from A, not present in A_w. Under these hypotheses, P@$K(i)$ can be computed by summing over all pairs of boundary tuples $\sigma_m, \sigma_n \in A$ the following.

$$Pr\left(R(\sigma_m) \overset{U^{+mn}_{-A}}{\leq} K - i + 1, R(\sigma_n) \overset{U^{+mn}_{-A}}{>} K - i + 1, R(\sigma_m) \overset{A}{=} i, R(\sigma_n) \overset{A}{=} i + 1\right) \tag{33}$$

Bringing in the specific scores of σ_m and σ_n, and then switching to conditional probabilities using the multiplication rule, allows rewriting the expression for P@$K(i)$, as follows:

$$\begin{aligned}\text{P@}K(i) = \sum_{m=1}^{K} \sum_{n=1}^{K} \sum_{s \in S(\sigma_m)} \sum_{s' \in S(\sigma_n)} &Pr(R(\sigma_m) \overset{U^{+mn}_{-A}}{\leq} K - i + 1, \\ &R(\sigma_n) \overset{U^{+mn}_{-A}}{>} K - i + 1 \mid s, s', R(\sigma_m) \overset{A}{=} i,\ R(\sigma_n) \overset{A}{=} i + 1) \cdot \\ &Pr\left(s, s', R(\sigma_m) \overset{A}{=} i, R(\sigma_n) \overset{A}{=} i + 1\right)\end{aligned} \tag{34}$$

Computing $Pr(s, s', R(\sigma_m) \overset{A}{=} i, R(\sigma_n) \overset{A}{=} i + 1)$ is described in Section 4.1.3 and needs to be performed $O\left(K^2\right)$ times (once for each pair of tuples in A), with a total run-time complexity of $O\left(c^2K^4\right)$. To compute the first probability term in Eq. 34, we extend the algorithm for computing the joint probability of the tuples in consecutive ranks, computing $Pr(R(\sigma_m) \overset{U^{+mn}_{-A}}{=} r_1, R(\sigma_n) \overset{U^{+mn}_{-A}}{=} r_2, s, s')$ for all $1 \leq r_1 \leq K$, $1 \leq r_2 \leq K + 1$. The run-time complexity for computing those probabilities is $O\left(c^2NK^2\right)$, the same runtime shown for computing the joint probability of the tuples in consecutive ranks (Section 4.1.3). Therefore, the total run-time complexity is $O\left(\max\left(c^2NK^2, c^2K^4\right)\right)$

4.2.3 Computing E [P@K]. Given a universe U and a K-subset A of U, by the definitions of P@$K(i)$ and E [P@K_A], we have

$$E\left[\text{P@}K_A\right] = \sum_{i=1}^{K} \frac{i}{K} \text{P@}K(i) \tag{35}$$

Together with the computation of P@$K(i)$ we obtain Proposition 2.

PROPOSITION 2. *Given a universe U and a K-subset A of U, Eqs. 32, 34, 35 correctly compute the expected P@K of A.*

As to the runtime complexity of computing Eq. 35, note that a single run of the procedure described in Section 4.2.2 gathers the quantities P@$K(i)$ for all $1 \leq i \leq K$. Therefore, the runtime complexity remains $O\left(\max\left(c^2NK^2, c^2K^5\right)\right)$.

5 EMPIRICAL EVALUATION

We now present an empirical evaluation of the quality-wise performance of various semantics to top-K answering, as well as assessing the scalability of the RankDist algorithm. Main results are:

- *Global Top-k* (Section 3.1), a rank-based semantics, generally outperforms score-based semantics in terms of the two precision variants (Section 3.2). These results demonstrate the effectiveness of the approach in generating high-quality recommendations, validating its performance beyond the theoretically proven dominance, by expectation (Section 3.3.1).
- While theoretically expected DCG@K is maximized by *Expected Score* (Section 3.3.3), empirically the rank-based *Global Top-k* generally achieves the highest DCG@K scores.
- The conservative approach may yield different results that its liberal counterpart. This suggests the importance of evaluating recommendation results using both versions of these metrics.
- RankDist (Section 4.1) exhibits scalable computation time.

Section 5.1 details experimental methodology, followed by quality analysis (Section 5.2) and RankDist runtime analysis (Section 5.3).

5.1 Experimental Methodology

We next describe the datasets used in the experiments, the filtering methods, and the examined ranking semantics. The ranking pipeline is implemented in Python and Code repository is available in the github. [3] Experiments were conducted on a server with 2 Nvidia RTX A6000 GPUs, a 3.5GHz CPU with multiple cores, 125GB RAM, and a CentOS 7 operating system.

5.1.1 Datasets: We conduct experiments on two real-world datasets containing user-movie ratings, which were utilized in a recent survey addressing uncertainty in recommender systems [4]:

- **ml-25m,** collected from the MovieLens website. We use the latest dataset with 24,945,870 ratings of 162,541 users on 32,720 movies. Rating values are in $\{0.5, 1.0, 1.5, \ldots 5.0\}$.
- **Netflix,** collected from the Netflix website, with 100,480,507 ratings of 480,189 users on 17,770 movies. Rating values are in $\{1.0, 2.0, 3.0, 4.0, 5.0\}$.

5.1.2 Filtering Methods: Probability distribution was generated for each candidate user and item using CPMF [28], which was shown to achieve state-of-the-art results among algorithms providing distribution over the whole set of scores [4]. CPMF is trained using non-test users rating to predict items distributions for each test user and for each item with which the user did not interact during training.

We adopt the data pre-processing method of Coscrato and Bridge [4]. Specifically, we randomly selected 10,000 users as test users and use probability-of-relevance [4] (Section 3.1).

[3] https://github.com/authorAnonymousGit/Ranking-in-RS-with-Uncertain-Scores

5.1.3 Ranking Formulation: For each dataset and for each test user, we process a top-K query of movie recommendation. For quality analysis, we compare four ranking semantics, namely Uncertainty-Based Filtering (UBF), Probability-of-Relevance Ranking (PRR), Global Top-k, and Expected Score (Section 3.1).

5.1.4 Algorithm Execution & Performance Assessment: Global Top-k uses RankDist (Section 4.1) for ranking. We use here a stochastic tie-breaking, where tuples can share a spot with equal probability leading to more complex computation (*e.g.*, eqs. 20 and 21). The stochastic tie-breaking version of Algorithm 1 is given in Section 4.1.2. The other evaluated ranking approaches use item score distributions, as computed in the filtering step.

We discretize the continuous scores of CPMF as a preliminary step to RankDist execution. Discretization is performed by binning according to the possible scores of each dataset. Specifically, given an ordered set of scores $Scores$, we define $|Scores|-1$ intervals where interval i ($1 \leq i \leq |Scores| - 1$) is assigned with the score s_i and the probability mass equals to the cumulative probability between s_i and s_{i+1} computed using the truncated continuous distribution.

We vary the number of tuples (K) in $\{1, 2, \ldots, 20\}$ while fixing the candidate set size to $N = 1000$. For each semantics, we report on the quality of the returned answer using the conservative and liberal versions of $P@K$ and $DCG@K$ (Section 3.2).

5.2 Quality Analysis

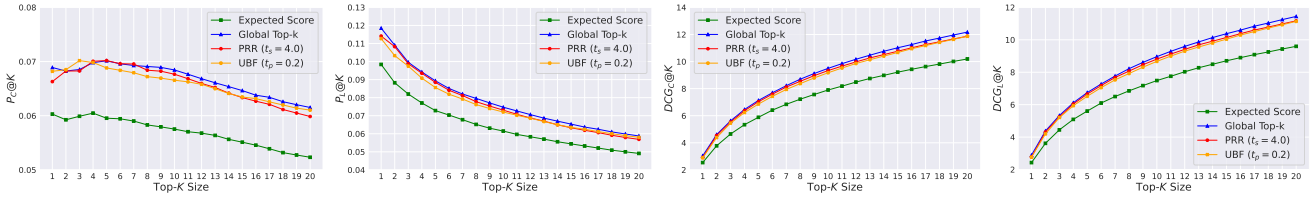
Section 3.3 shows that the Global top-K and Expected Score semantics maximize P@K and DCG@K by expectation, respectively and that *U-Topk* maximizes P@$K(K)$. Here, we focus on comparing the actual performance of P@K and DCG@K against UBF and PRR.

Figure 2 reports on the average quality obtained by each of the four semantics. The top row refers to the ml-25m dataset, while the bottom row presents the results for the Netflix dataset. The columns refer, from left to right, to $P_C@K$, $P_L@K$, $DCG_C@K$, and $DCG_L@K$. It is worth noting that for recommender systems, low precision is the rule rather than the exception due to the large set of items and the small user-specific reference list.

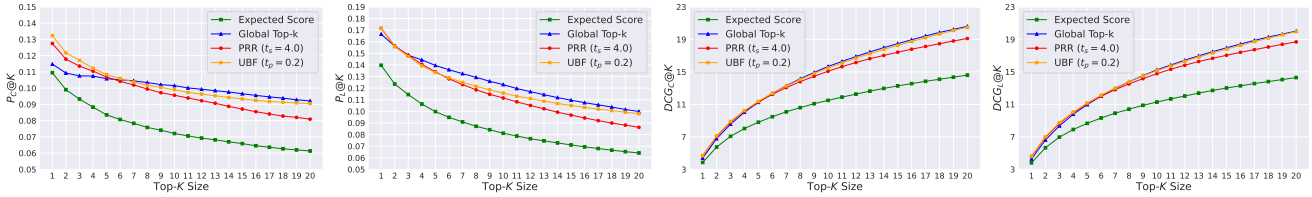
For $P_C@K$ (figures 2a, 2e) and $P_L@K$ (figures 2b, 2f), we observe that for $K \leq 6$ the top performance is shared between PRR and UBF. For larger K values, the Global top-k semantics demonstrates superior or competitive performance. Additionally, the Global top-k approach surpasses other approaches in more K values when evaluating the performance using $P_L@K$. Overall, Global Top-K exhibits superior performance compared to other semanticss, while Expected Score performance is consistently inferior.

Turning to DCG@K (figures 2c, 2d, 2g, and 2h), Global top-k demonstrates superior or competitive results for the MovieLens dataset. On the Netflix dataset, Global top-K shows competitive results relative to the UBF approach for larger k values ($k \geq 10$), while exhibiting slightly inferior but still competitive results compared to the UBF and PRR approaches for smaller k values. Surprisingly, while Expected Score maximizes the expected DCG@K, it has inferior results compared to the other approach. This phenomenon is likely connected to the large variance of possible scores.

Subsequently, we analyze the Global top-K semantics, comparing the liberal and conservative precision versions on the two datasets (Figure 3). Figure 3a, presenting metrics over the ml-25m dataset,

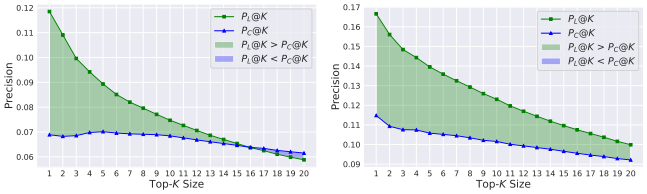


(a) $P_C@K$ **on ml-25m** **(b)** $P_L@K$ **on ml-25m** **(c)** $DCG_C@K$ **on ml-25m** **(d)** $DCG_L@K$ **on ml-25m**



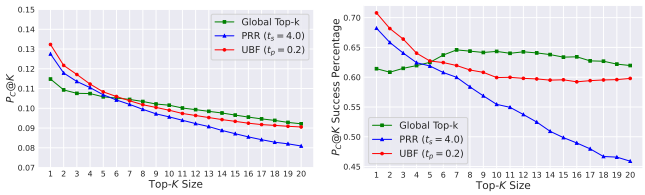
(e) $P_C@K$ **on Netflix** **(f)** $P_L@K$ **on Netflix** **(g)** $DCG_C@K$ **on ml-25m** **(h)** $DCG_L@K$ **on ml-25m**

Figure 2: Metrics Results on the ml-25m dataset (top row) and Netflix dataset (bottom row). The leftmost column presents the metric $P_C@K$, followed by the metrics $P_L@K$, $DCG_C@K$ and $DCG_L@K$.



(a) ml-25m **(b) Netflix**

Figure 3: Comparison of the $P_L@K$ and $P_C@K$ metrics Results on the ml-25m dataset (left) and Netflix dataset (right).



(a) $P_C@K$ **(b)** $P_C@K$ **Success**

Figure 4: Comparison of the $P_C@K$ on the Netflix dataset.

show $P_L@K$ to be larger than $P_C@K$ for $K < 16$, while $P_C@K$ is greater than $P_L@K$ for larger K values ($K > 16$). On the Netflix dataset (Figure 3b), $P_L@K$ is greater than $P_C@K$ for all K values. Recall the reference lists, A'_C and A'_L, for the $P_C@K$ and $P_L@K$ metrics, respectively (Section 3.2). Since A'_L contains only the items with true score exceeding a threshold (set to 4.0 in our experiments), A'_L may contain fewer than K items. On the other hand, as long as there are at least K items in the test set of each user, A'_C will contain exactly K items, including items with score below the threshold. We note that as K becomes larger, the list A'_C may not be contained in A'_L, and therefore, Theorem 1 no longer hold. Overall, we see that the observations derived from analyzing the conservative and liberal versions differ and assert the importance of evaluating recommendation results not only using the traditional liberal approach, but also considering our proposed version of conservative computation.

Next, we focus on $P_C@K$ and analyze performance in terms of *success rate at K*, the percentage of users for whom the top-K result generated by a semantics achieved the highest non-zero $P_C@K$ value. We note that at times, multiple semantics may be equally successful. Figure 4 shows side-by-side $P_C@K$ and success rate on the Netflix dataset for the three leading semantics. For small K values ($K < 7$), UBF exhibits the highest performance. For larger K values, Global Top-K takes the lead. While the relative performance of semantics has the same tendency, for large K values, the difference between Global Top-K and UBF is more significantin terms of success rate.

Dataset	Metric	Semantics		
		UBF	PRR	Global Top-K
ml-25m	$P_C@K$	0.6902	0.6726	**0.6932**
	$P_L@K$	0.6755	0.667	**0.6995**
	$DCG_C@K$	**0.484**	0.3513	0.4701
	$DCG_L@K$	**0.4871**	0.3541	0.4757
Netflix	$P_C@K$	0.5915	0.5273	**0.6096**
	$P_L@K$	0.5736	0.5116	**0.6279**
	$DCG_C@K$	0.4315	0.2831	**0.4419**
	$DCG_L@K$	**0.4378**	0.2884	0.436

Table 5: Success Rate by Dataset, Metric, and Approach

Table 5 presents success rate (over all K values) by dataset and metric for the three leading semantics. Bold values represent the winning semantics. Global Top-K achieves the best results in terms of $P_C@K$ and $P_L@K$, followed by UBF and PRR. While for $P_C@K$, Global Top-K surpasses UBF by a small margin, it shows large

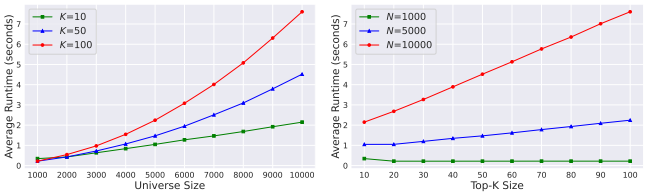
margin when evaluated using $P_L@K$. The results confirm that Global Top-K is the semantics of choice for maximizing precision. As for DCG, UBF demonstrates generally better performance than Global Top-k, with the exception of $DCG_C@K$ for the Netflix dataset.

Finally, we observe that although the semantics dominance remain consistent between the two datasets, the performance over the MovieLens dataset is lower compared to Netflix dataset. One reason for this disparity may be attributed to the fact that MovieLens has a greater number of score values compared to the Netflix dataset (10 versus 5), which could impact the quality of score distributions and subsequently affect overall performance.

5.3 RankDist Runtime Analysis

Next, we analyze runtime scalability of RankDist, a central component in computing Global-top$K$, and $E[\text{P@}K]$. We start by present the runtime for deterministic tie-breaking, in accordance with the runtime complexity reported in Section 4.1. Then, we also present the runtime of RankDist using stochastic tie-breaking.

For this purpose, we simulated the generation of score distributions with synthetic data using the bi-uniform distribution $(U[0,1), U[0,1))$. We employed different parameter configurations (N, K) with universe sizes (N) sampled in [1,000, 10,000] items and top-K size (K) in [10, 100]. Each experiment configuration was repeated 10 times, and for each we report on the average runtime for computing the rank distribution of all items (in seconds).

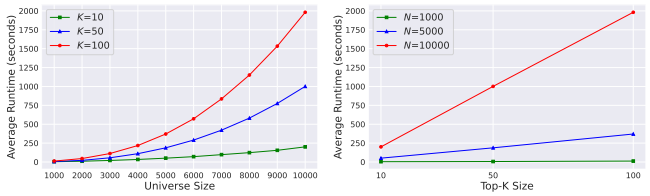


(a) Universe size (N) **(b) Top-K answer size**

Figure 5: RankDist average runtime with respect to N and K.

We show runtime analysis of RankDist with deterministic tie-breaking with respect to N for three representative K values (Figure 5a) and with respect to K for three representative N values (Figure 5b), with $c = 10$ possible scores. RankDist runtime follows the worst-case analysis in Section 4.1.1 ($O(cNK)$ per tuple and $O(cN^2K)$ for the whole dataset) with linear trend as K increases and quadratic trend as N increases. Through code optimization, including utilization of vectorized operations for computing the rank distribution of multiple tuples over all possible ranks, we managed to keep constants low, suggesting a practical scalability.

Next, we present the runtime analysis of RankDist with stochastic tie-breaking with respect to N for three representative K values (Figure 6a) and with respect to these representative K values for three representative N values (Figure 6b), with $c = 10$ possible scores. While the RankDist with stochastic tie-breaking shows a similar runtime trend to the runtime of the deterministic tie-breaking, the runtime of the stochastic version needs to be further optimized.



(a) Universe size (N) **(b) Top-K answer size**

Figure 6: RankDist with Stochastic tie-breaking average runtime with respect to N and K.

6 RELATED WORK

Multiple semantics for top-K querying with probabilistic ranking were proposed in the literature, see surveys [15, 29]. Works on top-K querying over databases with uncertain scores ([3, 10, 13, 20, 23, 26, 31]) use rank distribution, parameterized ranking, and more. Soliman et al. [26] were the first to address probabilistic ranking, formulating two query semantics, namely U-Top$k$ and U-kRanks. Proposition 2 shows that U-Top$k$ maximizes P@K(K). U-kRanks returns a multi-set of k tuples having the highest probability to be ranked at the top-k positions. Re et al. [23] define a top-K query over probabilistic data to be a query that returns tuples with the highest probabilities while others also take into account the ranking that results from these probabilities. Uncertain-scores semantics is based on x-relations [1], where the existence of each tuple is assumed to be deterministic (or has a positive probability to be "null"), while the score of each tuple is probabilistic (possibly based on probabilistic assignment of values to attributes). Efficient algorithms for multiple approaches, including U-Top$k$, U-kRanks, Expected rank, and UTop-Rank were also presented [3, 24, 30]. Additional approaches include the PT-k and Global Top-k approaches, proposed by Hua et al. [13] and Zhang and Chomicki [31], respectively (see Section 3.1 for details).

Cormode *et al.* [3] proposed to evaluate ranking with uncertain scores using a-priori properties. However, while some will certainly agree with the naturalness of specific properties (such as, *e.g.*, "query answer will contain no more than K tuples"), others may as well disagree. The connection between these declarative properties and the expected user satisfaction from the answer to her query was never established. Devic et al. [5] examined ranking functions that use uncertain predictions using multiple aspects, such as stability, which are less related to the accuracy of the query result.

Several distance measures were proposed as posteriori quality measures of top-K answers over uncertain data [19], including the symmetric difference and Kendall's tau distance, and the error of several ranking methods, including PT-k, with respect to distance measures was analyzed. To the best of our knowledge, they were the first to explicitly introduce *a posteriori* quality measures for the outcome of a top-K query, aiming at finding a consensus top-K answer, which aims at quantifying the degree of consensus over all possible determinizations of the probabilistic data, parameterized with a measure of distance between rankings. We extend the work presented by [19] to a broad investigation of the quality of top-K answers over uncertain data. In particular, we establish the optimality of Global top-k approach for expected P@K, Expected Score semantics for expected DCG@K and the U-Top$k$ semantics for P@K(K).

Numerous studies address uncertainty in recommender systems by investigating techniques to estimate and evaluate user-item score probabilities, as well as leveraging both scores and uncertainties for top-K recommendations. For instance, Koren and Sill [18] predict probability distribution of the ordinal set of scores for each user-item pair by integrating a collaborative filtering method with an ordinal regression model. Ortega *et al.* [22] utilize multiple Bernoulli factorizations combined with a classification model to generate score distribution, which is then employed for generating recommendations. Coscrato and Bridge [4] conduct a survey on methods for estimating and evaluating uncertainty in recommender systems and propose ranking approaches that make use of both scores and uncertainties to generate recommendations. While those ranking methods are score-based approaches, returning top-K result based on items predicted score distributions, probabilistic ranking offers also rank-based semantics, in which possible ranking results are examined. Therefore, we offer the use of probabilistic ranking as a tool of choice for generating recommendation in the presence of uncertainty.

7 CONCLUSIONS

In this work, we examined top-K queries over uncertain data in recommender systems. We established the need to consider the desired performance metric when selecting an appropriate semantics for ranking tuples according to uncertain scores. Specifically, we have shown for three ranking semantics, which performance metric they optimize (mostly by expectation). Beyond the formal analysis, we show through an empirical study that the Global top-K offers mostly dominant performance for both $P@K$ and $DCG@K$. To support the computation of Global top-K, and other rank-based semantics, we introduce RankDist, an efficient algorithm for computing the probabilty of a tuple to be ranked at any position on a top-K list.

Tying the literature of top-K queries in recommender systems and probabilistic ranking in databases shows a great promise and we intend to continue and investigate aspects of it in the future. we intend to continue and investigate the relationships between ranking semantics and additional, common performance metrics such as AP@K. Also, we intend to investigate other elements of the pipeline, presented in Section 2 in more depth, and in particular the various methods of probability computation of uncertain user scores.

REFERENCES

[1] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. 2006. ULDBs: Databases with Uncertainty and Lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim (Eds.). ACM, 953–964. http://dl.acm.org/citation.cfm?id=1164209

[2] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-based systems* 46 (2013), 109–132.

[3] Graham Cormode, Feifei Li, and Ke Yi. 2009. Semantics of ranking queries for probabilistic data and expected ranks. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 305–316.

[4] Victor Coscrato and Derek Bridge. 2023. Estimating and Evaluating the Uncertainty of Rating Predictions and Top-n Recommendations in Recommender Systems. *ACM Transactions on Recommender Systems* (2023).

[5] Siddartha Devic, Aleksandra Korolova, David Kempe, and Vatsal Sharan. 2024. Stability and Multigroup Fairness in Ranking with Uncertain Predictions. *arXiv preprint arXiv:2402.09326* (2024).

[6] Osnat Drien, Matanya Freiman, Antoine Amarilli, and Yael Amsterdamer. 2023. Query-Guided Resolution in Uncertain Databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data* 1, 2 (2023), 1–27.

[7] Su Feng, Boris Glavic, and Oliver Kennedy. 2023. Efficient Approximation of Certain and Possible Answers for Ranking and Window Queries over Uncertain Data. *Proc. VLDB Endow.* 16, 6 (feb 2023), 1346–1358. https://doi.org/10.14778/3583140.3583151

[8] Junyang Gao, Yifan Xu, Pankaj K Agarwal, and Jun Yang. 2021. Efficiently answering durability prediction queries. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*. 591–604.

[9] Xiangyu Gao, Jianzhong Li, and Dongjing Miao. 2023. Computing All Restricted Skyline Probabilities on Uncertain Datasets. *arXiv preprint arXiv:2303.00259* (2023).

[10] Tingjian Ge, Stan Zdonik, and Samuel Madden. 2009. Top-k queries on uncertain data: on score distribution and typical answers. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 375–388.

[11] Dhrubajyoti Ghosh, Peeyush Gupta, Sharad Mehrotra, Roberto Yus, and Yasser Altowim. 2022. JENNER: just-in-time enrichment in query processing. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2666–2678.

[12] Jiafeng Guo, Yinqiong Cai, Yixing Fan, Fei Sun, Ruqing Zhang, and Xueqi Cheng. 2022. Semantic models for the first-stage retrieval: A comprehensive review. *ACM Transactions on Information Systems (TOIS)* 40, 4 (2022), 1–42.

[13] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. 2008. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 1403–1405.

[14] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. 2008. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 673–686.

[15] Ihab F Ilyas and Mohamed A Soliman. 2011. Probabilistic ranking techniques in relational databases. *Synthesis Lectures on Data Management* 3, 1 (2011), 1–71.

[16] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[17] Norman Knyazev and Harrie Oosterhuis. 2023. A Lightweight Method for Modeling Confidence in Recommendations with Learned Beta Distributions. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 306–317.

[18] Yehuda Koren and Joe Sill. 2011. Ordrec: an ordinal model for predicting personalized item rating distributions. In *Proceedings of the fifth ACM conference on Recommender systems*. 117–124.

[19] Jian Li and Amol Deshpande. 2009. Consensus answers for queries over probabilistic databases. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 259–268.

[20] Jian Li, Barna Saha, and Amol Deshpande. 2011. A unified approach to ranking in probabilistic databases. *The VLDB Journal* 20 (2011), 249–275.

[21] Christopher D Manning, Hinrich Schütze, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Cambridge university press.

[22] Fernando Ortega, Raúl Lara-Cabrera, Ángel González-Prieto, and Jesús Bobadilla. 2021. Providing reliability in recommender systems through Bernoulli matrix factorization. *Information sciences* 553 (2021), 110–128.

[23] Christopher Re, Nilesh Dalvi, and Dan Suciu. 2007. Efficient top-k query evaluation on probabilistic data. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 886–895.

[24] Mohamed A Soliman and Ihab F Ilyas. 2009. Ranking with uncertain scores. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 317–328.

[25] Mohamed A Soliman, Ihab F Ilyas, and Shalev Ben-David. 2010. Supporting ranking queries on uncertain and incomplete data. *The VLDB Journal* 19, 4 (2010), 477–501.

[26] Mohamed A Soliman, Ihab F Ilyas, and Kevin Chen-Chuan Chang. 2007. Top-k query processing in uncertain databases. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 896–905.

[27] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. Probabilistic databases. *Synthesis lectures on data management* 3, 2 (2011), 1–180.

[28] Chao Wang, Qi Liu, Runze Wu, Enhong Chen, Chuanren Liu, Xunpeng Huang, and Zhenya Huang. 2018. Confidence-aware matrix factorization for recommender systems. In *Proceedings of the AAAI Conference on artificial intelligence*, Vol. 32.

[29] Yijie Wang, Xiaoyong Li, Xiaoling Li, and Yuan Wang. 2013. A survey of queries over uncertain data. *Knowledge and information systems* 37, 3 (2013), 485–530.

[30] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. 2008. Efficient processing of top-k queries in uncertain databases with x-relations. *IEEE transactions on knowledge and data engineering* 20, 12 (2008), 1669–1682.

[31] Xi Zhang and Jan Chomicki. 2009. Semantics and evaluation of top-k queries in probabilistic databases. *Distributed and parallel databases* 26, 1 (2009), 67–126.

[32] Yi Zhang and Zachary G Ives. 2020. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1951–1966.

[33] Andreas Züfle, Goce Trajcevski, Dieter Pfoser, and Joon-Seok Kim. 2020. Managing uncertainty in evolving geo-spatial data. In *2020 21st IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 5–8.