

Probabilistic Ranked Queries in Uncertain Databases

Xiang Lian and Lei Chen
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon
Hong Kong, China
{xlian, leichen}@cse.ust.hk

ABSTRACT

Recently, many new applications, such as sensor data monitoring and mobile device tracking, raise up the issue of uncertain data management. Compared to “certain” data, the data in the uncertain database are not exact points, which, instead, often locate within a region. In this paper, we study the *ranked queries* over uncertain data. In fact, ranked queries have been studied extensively in traditional database literature due to their popularity in many applications, such as decision making, recommendation raising, and data mining tasks. Many proposals have been made in order to improve the efficiency in answering ranked queries. However, the existing approaches are all based on the assumption that the underlying data are exact (or certain). Due to the intrinsic differences between uncertain and certain data, these methods are designed only for ranked queries in certain databases and cannot be applied to uncertain case directly. Motivated by this, we propose novel solutions to speed up the *probabilistic ranked query* (PRank) over the uncertain database. Specifically, we introduce two effective pruning methods, *spatial* and *probabilistic*, to help reduce the PRank search space. Then, we seamlessly integrate these pruning heuristics into the PRank query procedure. Extensive experiments have demonstrated the efficiency and effectiveness of our proposed approach in answering PRank queries, in terms of both wall clock time and the number of candidates to be refined.

1. INTRODUCTION

Recently, query processing over uncertain data has gained much attention from the database community due to the inherent uncertainty of data in many real-world applications, such as sensor network monitoring [15], object identification [4], moving object search [9, 8, 25], and the like [31, 32]. For example, in an application to track and monitor moving objects, the exact positions of objects may not be available in the database at query time. This phenomenon may result from low precision of positioning devices or long transmission delay. Therefore, each moving object has “dynamic” coordinates and can locate anywhere with any distribution in a so-called *uncertainty region* [9, 33], which is inferred by its last reported position, maximum speed, moving directions, and so on. As another example, in sensor networks, sensor data are col-

lected from different sites and transmitted back to the sink. During the transmission, data might be distorted by environmental factors, transmission delay, or packet losses. Thus, the collected data are often imprecise and contain noises deviating from their actual values. Figure 1 illustrates a 2D example of small *uncertain database* \mathcal{D} , which contains six uncertain objects $A, B, C, D, E,$ and F . In particular, each uncertain object is represented by an *uncertainty region*, denoted as the shaded area in the figure. Without loss of generality, we simply model each uncertainty region as of circular shape [33] (i.e. hypersphere in a multidimensional data space). The uncertain objects can only appear in their own uncertainty regions with arbitrary probability distributions, and they cannot locate outside the regions.

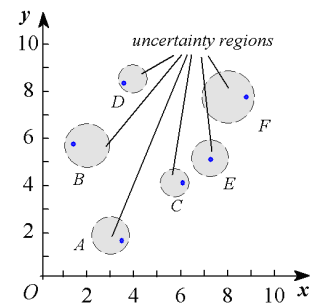


Figure 1: Illustration of Uncertainty Regions in 2D Space

In this paper, we investigate an important type of query, the *ranked query* [34], over uncertain data. Specifically, given a preference function specified by users, a *ranked query* retrieves k data objects in the database such that their scores (calculated by the given preference function) are the highest. In fact, due to its popularity, the ranked query has been used in many applications, such as decision making, recommendation raising, and data mining tasks. Many proposals have been made to improve the efficiency in answering ranked queries. However, these approaches all assume that the underlying database is exact (or certain). As shown in Figure 1, due to the uncertain property, each data object is now a region rather than a *precise* point; moreover, the distance between any two uncertain objects is a variable instead of a definite value. These intrinsic differences between certain and uncertain data make previous approaches (proposed for ranked queries over certain data) not directly applicable to the uncertain scenario. To the best of our knowledge, no previous work has considered the ranked query under the settings of the uncertain database, namely the *probabilistic ranked query* (PRank). In particular, given a preference function and an uncertain database, a PRank query retrieves k uncertain ob-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EDBT'08, March 25-30, 2008, Nantes, France.

Copyright 2008 ACM 978-1-59593-926-5/08/0003...\$5.00.

jects that are expected to have the highest scores.

Figure 2 presents an example of ranked query in both traditional (containing “certain” objects) and uncertain databases. The preference function $f(\vec{O})$ is used to calculate the score of an object $\vec{O}(O_1, O_2)$, where O_1 and O_2 are two coordinates of object \vec{O} . For the sake of simplicity, we use O to denote vector (object) \vec{O} throughout this paper. In the example, we assume the score, $f(O)$, of object O is given by $(O_1 + O_2)$, where the same weight (i.e. 1) is assigned to both dimensions. In a general case, different weights can be introduced by users to bias the preference of different dimensions. The line, $f(O) = O_1 + O_2$, in Figure 2 contains a set of points that have the same score value (i.e. $f(O)$). In the traditional database of Figure 2(a), object E has the second largest score (only smaller than that of object F). In an uncertain database, however, as illustrated in Figure 2(b), it is not clear any more whether object E indeed has the second largest score, since object D is also a potential candidate to have the second largest score. Therefore, we have to re-define the probabilistic ranked query in the context of the uncertain database.

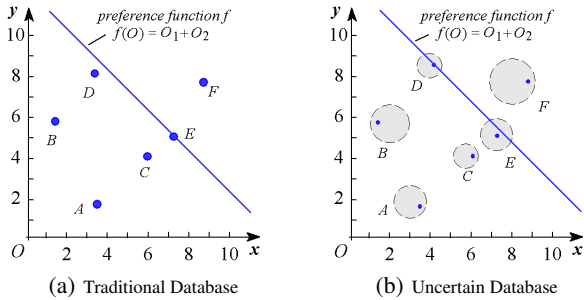


Figure 2: A 2D Example of Ranked Query

In this paper, we propose an efficient and effective approach to answer the PRank query. Specifically, we provide effective pruning heuristics to significantly reduce the PRank search space, and utilize a multidimensional index to efficiently perform the PRank query processing.

In particular, we make the following contributions.

1. We formalize a novel query, the *probabilistic ranked query* (PRank), in the context of uncertain databases.
2. We illustrate a general framework for answering the PRank query, and propose effective pruning heuristics to help reduce the PRank search space.
3. We seamlessly integrate the proposed pruning methods into the query procedure, which can efficiently retrieve the PRank query results.
4. Last but not least, we demonstrate through extensive experiments the effectiveness of our pruning methods as well as the efficiency of PRank query processing.

The rest of the paper is organized as follows. Section 2 briefly overviews previous methods to answer ranked queries over the traditional “certain” database, as well as various query processing over uncertain databases. Section 3 formally defines our problem of *probabilistic ranked query*. Section 4 proposes the general framework and pruning heuristics for answering PRank queries. Section 5 presents the PRank query procedure to perform the PRank

search. Section 6 demonstrates the query performance of PRank under different experimental settings. Finally, Section 7 concludes this paper.

2. RELATED WORK

Section 2.1 briefly reviews the ranked query in traditional databases that contain “certain” objects. Section 2.2 presents the query processing over uncertain databases.

2.1 Ranked Queries in Traditional Databases

Ranked query has many real applications such as decision making, recommendation raising, and data mining tasks. Specifically, given a d -dimensional database \mathcal{D} and a linear preference function $f(\cdot)$, a *ranked query* retrieves k objects $O \in \mathcal{D}$, such that their corresponding scores, $f(O)$, are the highest in the database, where $f(O) = \sum_{i=1}^d w_i \cdot O_i$ (note: w_i is a weight indicating the user’s preference and O_i is the i -th coordinate of object O).

Due to the importance of ranked queries, many previous work focus on efficient methods to retrieve the query answer. Specifically, Chang et al. [7] first proposed an Onion technique to perform the ranked search. They consider each data object in the database \mathcal{D} as a multidimensional point in the data space. Let CH_1 be the (layer-1) *convex hull* of all the points in \mathcal{D} , CH_2 be the layer-2 convex hull of those points that are not in CH_1 (i.e. $\mathcal{D} \setminus CH_1$), and so on. In the general case, CH_i is the layer- i convex hull of those points in $\mathcal{D} \setminus (CH_1 \cup CH_2 \cup \dots \cup CH_{i-1})$. The basic idea of the Onion technique is as follows. The top-1 object (i.e. with rank 1) is always in CH_1 ; the second ranked object is always in $CH_1 \cup CH_2$; the third ranked one is in $CH_1 \cup CH_2 \cup CH_3$; ...; and so on. Therefore, data objects can be pre-processed by sorting them according to their layer numbers. Whenever a ranked query that retrieves k top-ranked objects arrives, the Onion algorithm starts scanning the data set from the most-exterior (layer-1) convex hull (i.e. CH_1) to layer- k convex hull (i.e. CH_k). All the retrieved objects are candidate answers to the ranked query. A similar *layer-based* method, AppRI, has recently been proposed by Xin et al. [39].

Hristidis et al. [17, 18] provided a *view-based* approach, PREFER, to answer the ranked query. In particular, Hristidis et al. pre-defined some preference functions $f_v(\cdot)$, and created a materialized view for each function by sorting data objects in descending order of their scores. Given a query preference function $f(\cdot)$, PREFER first selects one of the pre-computed views, with respect to a preference function $f_v(\cdot)$ that is the most similar to $f(\cdot)$, and then sequentially scans only a portion of this view, stopping at the *watermark* point. Another *view-based* technique, LPTA, proposed by Das et al. [13], also maintains sorted record id lists according to the view’s preference functions.

Fagin et al. [14] proposed the *threshold algorithm* (TA) to answer the top- k query, which is based on the ranked lists. Specifically, along each dimension i , they sort the data objects in descending order of the i -th coordinate. Thus, in a d -dimensional database, d sorted lists can be obtained, which are accessed sequentially in a round-robin fashion. Whenever an object is obtained from a ranked list, TA immediately computes its score. Let T be a score threshold defined as the maximum possible score for those objects that have not been seen so far. If there exist k objects (that we have seen so far) that have scores higher than T , then TA terminates and reports these k objects as the query result. In the context of relational databases, many previous work also studied top- k queries, including [5, 6, 27, 37, 20, 23, 26].

Tao et al. [34] aimed to improve the retrieval efficiency of ranked queries with the help of the R-tree index [16]. In particular, they proposed a *branch-and-bound ranked search* (BRS) algorithm. BRS maintains a maximum heap to help traverse the R-tree in a *best-first* manner, where the key of heap entry is defined as the maximum possible score of data points in this entry. The query performance of BRS is proved to be I/O optimal. Yiu et al. [42] defined top- k spatial preference queries, which also utilize the R-tree index to retrieve objects with high scores. Recently, there are some other work on the top- k query or its variants, to list a few of them, [40, 24, 43, 1, 19, 3]. However, these work only handle certain data, and cannot be directly applied to uncertain databases, which motivates us to propose a new method to answer ranked queries over uncertain data.

2.2 Query Processing in Uncertain Databases

Query processing over uncertain data has gained much attention due to its importance in many applications [15, 4, 9, 8, 25, 31, 32], where real-world data inherently contain uncertainty. For example, the Orion system [11] is a system for managing uncertain data in applications such as sensor data monitoring. Previous studies have considered many query types, such as *range query* [10, 12, 33], *nearest neighbor query* [9, 10, 22], *skyline query* [28], and *similarity join* [21], with which specific techniques are designed for searching over uncertain databases. To the best of our knowledge, so far no existing work has studied the ranked query in the context of the uncertain database, which assumes that data objects can have “dynamic” attributes in the data space (i.e. locating anywhere within the uncertainty regions). Note that, although previous works [29, 32, 41] studied top- k queries in the probabilistic database, they consider the *possible world* semantics in relational databases [30, 2], whereas our work focuses on the uncertain query processing in the spatial database.

Due to the inherent uncertainty in many real-world data from various applications, previous methods to handle “certain” data cannot be directly used and we have to find a solution which can efficiently answer the PRank query in uncertain databases, which is the focus of this paper.

3. PROBLEM DEFINITION

In this section, we formally define the problem of the *probabilistic ranked query* (PRank). In particular, assume we have a static *uncertain database* \mathcal{D} in a d -dimensional space, in which each uncertain object $O(O_1, O_2, \dots, O_d)$ can locate anywhere within an (hyper-spherical) *uncertainty region* $UR(O)$ [9, 33] centered at point C_O with radius r_O . Let $pdf(O)$ be the *probability density function* (pdf) with respect to the location that object O appears. We have $pdf(O) \in [0, 1]$, if $O \in UR(O)$; $pdf(O) = 0$, otherwise. Following the convention [10, 9, 28], we assume that all the data objects are independent of each other in the database \mathcal{D} . The problem of retrieving the PRank query results is defined as follows.

DEFINITION 3.1. (*Probabilistic k -Ranked Query, k -PRank*) Assume we have an uncertain database \mathcal{D} , a user-specified preference function f and an integer k . For $1 \leq m \leq k$, we define the *m -ranking probability* $Pr_m(O)$ of object $O \in \mathcal{D}$ as:

$$Pr_m(O) = \int_{s_1}^{s_2} \left(Pr\{f(O) = s\} \cdot \sum_{\forall \{P_1, P_2, \dots, P_{m-1}\} \in \mathcal{D} \setminus \{O\}} \left(\prod_{i=1}^{m-1} Pr\{f(P_i) \geq s\} \cdot \prod_{\forall P_j \in \mathcal{D} \setminus \{O, P_1, \dots, P_{m-1}\}} Pr\{f(P_j) \leq s\} \right) \right) ds. \quad (1)$$

Symbol	Description
\mathcal{D}	the data set with data size $ \mathcal{D} $
d	the dimensionality of the data set
$UR(O)$	the <i>uncertainty region</i> of object O
k	the number of uncertain objects to retrieve in the PRank query
$f(O)$	the preference function with respect to object O
$f_v(O)$	the pre-defined preference functions to compute the lower/upper bound probability
l	the <i>weight resolution</i>
n	the number of pre-computed probabilistic bounds with respect to a preference function $f_v(\cdot)$ for each uncertain object

Table 1: Meanings of Symbols Used

where s_1 and s_2 are the lower and upper bounds of score $f(O)$ for object O , respectively. A k -PRank query retrieves k uncertain objects $OR_1, OR_2, \dots, OR_m, \dots, OR_k$ ($\in \mathcal{D}$) such that object OR_m has the highest m -ranking probability $Pr_m(OR_m)$ among all data objects in \mathcal{D} .

Intuitively, Eq. (1) defines the expected probability $Pr_m(O)$ (i.e. *m -ranking probability*) that object O has the m -th largest score in the database \mathcal{D} . In particular, when the score $f(O)$ of object O is $s \in [s_1, s_2]$, we consider all possible cases where there are exactly $(m-1)$ objects P_1, P_2, \dots, P_{m-1} in $\mathcal{D} \setminus \{O\}$ having higher scores than s (i.e. higher ranks than O), while the other objects $P_j \in \mathcal{D} \setminus \{O, P_1, \dots, P_{m-1}\}$ have lower scores than object O . Thus, as shown in Eq. (1), for each possible combination of P_1, P_2, \dots, P_{m-1} , we calculate the probability that O has the m -th highest score by multiplying probabilities that objects have either higher or lower scores than s (due to the object independence [10, 9, 28]). Finally, we integrate the probability summation for all these combinations on s , and obtain the expected probability that O has the m -th rank. Note that, in a special case where $k = 1$, Eq. (1) can be rewritten as a much simpler form:

$$Pr_m(O) = \int_{s_1}^{s_2} \left(Pr\{f(O) = s\} \cdot \prod_{\forall P_j \in \mathcal{D} \setminus \{O\}} Pr\{f(P_j) \leq s\} \right) ds. \quad (2)$$

After defining the m -ranking probability, the problem of the PRank query is to retrieve object OR_1 that has the highest score with the highest probability $Pr_1(OR_1)$ among all the objects in \mathcal{D} ; object OR_2 that has the second highest score with the highest probability $Pr_2(OR_2)$; ...; and object OR_k that has the k -th highest score with the largest probability $Pr_k(OR_k)$. In this paper, we consider linear preference function $f(\cdot)$ and leave other interesting preference functions as our future work. Specifically, we let $f(O) = \sum_{i=1}^d w_i \cdot O_i$, where w_i are weights specified by the PRank query.

Since previous approaches are designed only for the ranked query processing over precise objects, they are not suitable for handling uncertain data. Thus, the only straightforward method to answer PRank queries is probably the *linear scan*. That is, we sequentially scan all the uncertain objects on disk one by one and calculate their expected probabilities in Eq. (1) with which the PRank results are determined. However, since the probability integration in Eq. (1) is quite complex, this method incurs high cost in terms of both computations and page accesses (i.e. I/O cost). Motivated by this, in the sequel, we aim to find pruning heuristics in order to effectively reduce the PRank search space and efficiently answer the query. Table 1 summarizes the commonly-used symbols in this paper.

4. PROBABILISTIC RANKED QUERIES

As mentioned earlier, the *linear scan* of the database incurs high computation and I/O costs. Thus, in order to speed up the procedure of answering *probabilistic ranked queries* (PRank), we index the *uncertainty regions* of data objects with a multidimensional index. Note that, since our proposed methodology is independent of the underlying index, throughout this paper, we simply use one of the popular indexes, R-tree [16]. In particular, R-tree recursively bounds the *uncertainty regions* of data objects with *minimum bounding rectangles* (MBRs), until one final node (i.e. the root) is obtained.

In the sequel, Section 4.1 presents a general framework for answering the PRank query. Section 4.2 illustrates the heuristics of the *spatial pruning* method, where the location distributions of uncertain objects can be either known or unknown. With the knowledge of the object distributions in their uncertainty region, Section 4.3 further presents the idea of *probabilistic pruning* method. Section 4.4 discusses the refinement of the PRank candidate set.

4.1 The General Framework

Figure 3 illustrates a general framework for answering the PRank query. In particular, the framework consists of four phases, *indexing*, *pruning*, *bounding*, and *evaluation*. In the first indexing phase, given an uncertain database \mathcal{D} , we construct an R-tree index \mathcal{I} over \mathcal{D} to facilitate the PRank query (line 1). As a second step, the pruning phase eliminates those uncertain objects that cannot be the PRank result, using novel *spatial* and/or *probabilistic pruning* method(s) that we propose (line 2), in the case where the location distribution of each object is either known or unknown within its uncertainty region. Next, in the retrieved candidate set, we conceptually bound those objects that are involved in the probability calculation in Eq. (1) (called bounding phase) and finally refine the candidates by computing the actual probability (in Eq. (1)) and returning answers in the evaluation phase (line 3). Below, we mainly focus on illustrating the pruning and bounding phases.

```

Procedure PRank_Framework {
  Input: a  $d$ -dimensional uncertain database  $\mathcal{D}$ , a preference function  $f(\cdot)$ ,
    and an integer  $k$ 
  Output:  $k$  objects in the PRank query result
  (1) construct a multidimensional index structure  $\mathcal{I}$  over  $\mathcal{D}$  // indexing phase
  (2) perform spatial and/or probabilistic pruning over  $\mathcal{I}$  // pruning phase
  (3) refine candidates and return the answer set // bounding and evaluation phases
}

```

Figure 3: The General Framework for PRank Queries

4.2 Spatial Pruning

In this subsection, we first propose a novel *spatial pruning* method to reduce the search space for k -PRank queries ($k \in [1, +\infty)$). Specifically, our spatial pruning method aims at eliminating those data objects that are definitely not in the k -PRank result. In fact, the rationale behind spatial pruning is that if we can clearly know that there are more than k data objects whose scores are higher than that of an object O , then we can safely prune object O .

Figure 4(a) illustrates the previous example of small uncertain database \mathcal{D} . In fact, since each uncertain object O can only locate within its uncertainty region $UR(O)$, the score $f(O)$ of O can be bounded by an interval, say $[LB_f(O), UB_f(O)]$. For instance, object F (or D) has its score within $[LB_f(F), UB_f(F)]$ (or $[LB_f(D),$

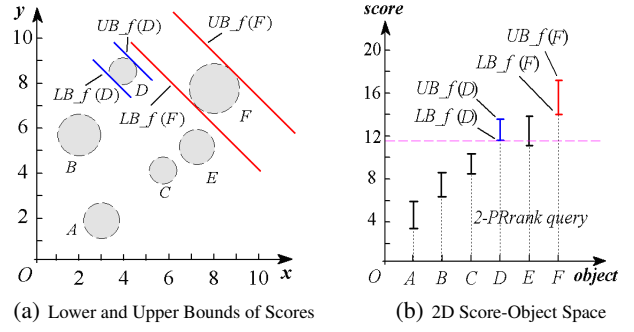


Figure 4: Heuristics of Spatial Pruning Method

$UB_f(D)]$). Now we convert the score interval of each object into a 2D *score-object* space, as shown in Figure 4(b), where the horizontal axis represents objects and the vertical one corresponds to scores. Assume we issue a 2-PRank query. From the figure, we find that object D has the second largest lower bound $LB_f(D)$ of score $f(D)$. Moreover, objects A has its upper bound score $UB_f(A)$ smaller than $LB_f(D)$. In other words, there must exist at least two data objects (e.g. D and F) which have their scores greater than A . Thus, object A can be safely pruned. Similarly, the other two objects B and C can also be discarded. On the other hand, for the last object E , however, since its upper bound score is greater than $LB_f(D)$, object E still has chance to be the query result, and thus it cannot be pruned.

We summarize our *spatial pruning* method as follows.

LEMMA 4.1. (*Spatial Pruning*) *Given an uncertain database \mathcal{D} , a user-specified preference function $f(\cdot)$ and an integer k , let P_1, P_2, \dots, P_k be the k uncertain objects that we have obtained so far. Assume $LB_f(P_k)$ is the smallest (i.e. k -th largest) lower bound of score among these k objects. The spatial pruning method can safely filter object O (with score interval $[LB_f(O), UB_f(O)]$) out, if $UB_f(O) \leq LB_f(P_k)$ holds.*

Proof. According to the assumption of the lemma, there exist at least k uncertain objects such that their scores are higher than that of object O . Thus, in Eq. (1), the probability integration (i.e. $Pr_m(O)$) of object O is always equal to zero. Hence, object O is guaranteed not to be in the k -PRank result, and thus can be safely pruned. \square

Computation of Lower and Upper Score Bounds. For spatial pruning, we need to get lower and upper score bounds for each uncertain object. We assume that the uncertainty region $UR(O)$ of an uncertain object O is centered at point C_O with radius r_O . Moreover, a query preference function is defined as $f(O) = \sum_{i=1}^d (w_i \cdot O_i)$ ($w_i > 0$). Our goal is to find a data point X in $UR(O)$, such that its score is either minimized or maximized. Formally, we want to minimize/maximize $f(X) = \sum_{i=1}^d w_i \cdot X_i$, under the constraint $\sum_{i=1}^d (X_i - C_{O_i})^2 \leq r_O^2$. We solve this optimization problem and obtain the lower and upper bounds of score $f(O)$, respectively.

$$LB_f(O) = \sum_{i=1}^d \min \left\{ w_i \cdot \left(C_{O_i} \pm \frac{w_i}{\sqrt{\sum_{j=1}^d w_j^2}} \cdot r_O \right) \right\}, \quad (3)$$

$$UB_f(O) = \sum_{i=1}^d \max \left\{ w_i \cdot \left(C_{O_i} \pm \frac{w_i}{\sqrt{\sum_{j=1}^d w_j^2}} \cdot r_O \right) \right\}. \quad (4)$$

4.3 Probabilistic Pruning

Up to now, we have discussed the *spatial pruning* method, where the location distributions of data objects can be either known or unknown in their uncertainty region. However, if we have such a distribution knowledge (i.e. the location distributions of data objects), then we can further prune more uncertain objects by utilizing this information. In this subsection, we propose a novel *probabilistic pruning* method to help answer the k -PRank query.

As indicated by Definition 3.1, any object O is in the answer set of a k -PRank query, if there exists an integer $m \in [1, k]$ such that object O has the m -th largest score in the database with the highest probability $Pr_m(O)$ (defined in Eq. (1)). However, due to the complex probability integration in Eq. (1), it is quite inefficient to calculate the actual probability $Pr_m(O)$ directly. Thus, instead, the goal of our *probabilistic pruning* method is to find a probability interval $[LB_{Pr_m}(O), UB_{Pr_m}(O)]$ that tightly bounds the complex $Pr_m(O)$, and use this interval to efficiently prune data objects. The *probabilistic pruning* method can be summarized as follows.

LEMMA 4.2. (Probabilistic Pruning) *Given an uncertain database \mathcal{D} , a user-specified preference function $f(\cdot)$ and an integer k , let P_1, P_2, \dots , and P_k be the k uncertain objects that we have obtained so far. Assume $LB_{Pr_1}, LB_{Pr_2}, \dots$, and LB_{Pr_k} are the lower bound probabilities of $Pr_1(P_1), Pr_2(P_2), \dots$, and $Pr_k(P_k)$, respectively, where $Pr_m(P_m)$ is the maximum probability in the database that an object P_m has the m -th largest score. The probabilistic pruning method can safely prune those objects O (with probability interval $[LB_{Pr_m}(O), UB_{Pr_m}(O)]$), if $UB_{Pr_m}(O) \leq LB_{Pr_m}(P_m)$ holds, for all $m \in [1, k]$.*

Proof. By contradiction. Assume object O should be included in the PRank query result. Thus, based on Definition 3.1, O must have the m -th largest score with the *highest* probability in the database, for some $m \in [1, k]$. However, according to the lemma assumption that $UB_{Pr_m}(O) \leq LB_{Pr_m}(P_m)$ for all $m \in [1, k]$, object O always has the probability smaller than P_m for any m value, which is contrary. Hence, our initial assumption is incorrect, and O can be safely pruned. \square

Lemma 4.2 indicates that we can safely prune those objects that have the m -th largest score in the database with low probability, for all $m \in [1, k]$ among the data sets.

Next, we address the remaining issue, that is, how to obtain lower and upper bound probabilities, $LB_{Pr_m}(O)$ and $UB_{Pr_m}(O)$, respectively, for probability $Pr_m(O)$ calculated in Eq. (1). In particular, within the integration of Eq. (1), we have to enumerate all possible combinations of $\{P_1, P_2, \dots, P_{m-1}\}$, which is very complex and costly. For simplicity, we denote $S(N, m)$ as:

$$S(N, m) = \sum_{\forall \{P_1, P_2, \dots, P_{m-1}\} \in \mathcal{D} \setminus \{O\}} \left(\prod_{i=1}^{m-1} Pr\{f(P_i) \geq s\} \cdot \prod_{\forall P_j \in \mathcal{D} \setminus \{O, P_1, \dots, P_{m-1}\}} Pr\{f(P_j) \leq s\} \right) \quad (5)$$

where N is the data size of $\mathcal{D} \setminus \{O\}$.

Therefore, by substitute Eq. (5) into Eq. (1), now we can rewrite Eq. (1) as:

$$Pr_m(O) = \int_{s_1}^{s_2} Pr\{f(O) = s\} \cdot S(N, m) ds. \quad (6)$$

Based on Eq. (6), in order to derive lower and upper bounds for $Pr_m(O)$, it is sufficient to obtain lower and upper bounds of $S(N, m)$, denoted as $LB_S(N, m)$ and $UB_S(N, m)$, respectively. The reason is that:

$$Pr_m(O) \geq LB_S(N, m) \cdot \int_{s_1}^{s_2} Pr\{f(O) = s\} ds = LB_S(N, m), \quad (7)$$

$$Pr_m(O) \leq UB_S(N, m) \cdot \int_{s_1}^{s_2} Pr\{f(O) = s\} ds = UB_S(N, m), \quad (8)$$

which implies that $LB_S(N, m)$ and $UB_S(N, m)$ can be exactly considered as lower and upper bounds of $Pr_m(O)$, respectively.

Thus, below, we only need to calculate $LB_S(N, m)$ and $UB_S(N, m)$ for $S(N, m)$, and simply let $LB_{Pr_m}(O) = LB_S(N, m)$ and $UB_{Pr_m}(O) = UB_S(N, m)$.

Note that, from Eq. (5), we can convert $S(N, m)$ into its recursive form. Specifically, we let $G(P_i) = Pr\{f(P_i) \geq s\}$, and assume the data set $\mathcal{D} \setminus \{O, P_1, \dots, P_{m-1}\}$ contain objects P_{m+1}, P_{m+2}, \dots , and P_N . We have the recursive function of $S(N, m)$ as follows:

$$\begin{cases} S(N, m) = S(N-1, m) \cdot (1 - G(P_N)) + S(N-1, m-1) \cdot G(P_N), \\ S(N, 1) = (1 - G(P_1)) \cdot (1 - G(P_2)) \cdot \dots \cdot (1 - G(P_N)), \\ S(m-1, m) = G(P_1) \cdot G(P_2) \cdot \dots \cdot G(P_{m-1}). \end{cases} \quad (9)$$

Obviously, in Eq. (9), if we can find the lower and upper bounds of $G(P_i)$, denoted as $LB_G(P_i)$ and $UB_G(P_i)$, respectively, then $LB_S(N, m)$ and $UB_S(N, m)$ can be easily computed. So, in the sequel, we focus on the problem of bounding $G(P_i)$, utilizing some pre-computed probabilistic information.

We illustrate our intuition of finding $LB_G(P_i)$ and $UB_G(P_i)$ in an example of Figure 5. Assume we have an object P_i with its uncertainty region $UR(P_i)$ as shown in the figure. Given a query preference function $f(P_i) = \sum_{j=1}^d w_j \cdot P_{ij}$ with $G(P_i) = Pr\{f(P_i) \geq s\}$, our problem is to compute lower and upper bounds, $LB_G(P_i)$ and $UB_G(P_i)$, of $G(P_i)$, where $s \in [s_1, s_2]$.

The intuition of our proposed method is somewhat similar to that of PREFER [17]. However, compared to PREFER, which is designed for certain data, our proposal is more complex due to the uncertainty. In particular, we maintain a number of pre-defined preference functions (e.g., $f_v(P_i) = \sum_{j=1}^d v_j \cdot P_{ij}$ in the example). Similar to [17], preference functions, $f_v(O) = \sum_{j=1}^d v_j \cdot O_j$, are selected with a discretization of the weight domain (e.g. $(0, 1]$) into l parts of equal size, where l is the *weight resolution*. For example, when $l = 10$, we have the pre-defined preference functions $f_v(\cdot)$, whose v_j values come from 0.1, 0.2, ..., and 1.

For each $f_v(\cdot)$, we further pre-compute n probabilistic bounds with

score thresholds $t_{\beta_1}, t_{\beta_2}, \dots$, and t_{β_n} , with respect to every uncertain object P_i , such that object P_i has score greater than t_{β_i} with probability equal to β_i ($i \in [1, n]$), that is, $Pr\{f_v(P_i) \geq t_{\beta_i}\} = \beta_i \in [0, 1]$. As illustrated in Figure 5, we have 5 bounds (i.e. $n = 5$) with $t_{\beta_1} = t_{0.1}, t_{\beta_2} = t_{0.3}, t_{\beta_3} = t_{0.5}, t_{\beta_4} = t_{0.7}$, and $t_{\beta_5} = t_{0.9}$ (corresponding to 5 lines, respectively). For instance, the top line represents the function $f_v(P_i) = t_{0.1}$ (i.e. the score of P_i with $f_v(\cdot)$ is equal to $t_{0.1}$), where $t_{0.1}$ is a score threshold such that $Pr\{f_v(P_i) \geq t_{0.1}\} = 0.1$. The meanings of other bounds are similar.

Given a query preference function $f(O) = \sum_{j=1}^d w_j \cdot O_j$, we would choose one pre-computed preference function $f_v(\cdot)$ which is the most similar to $f(\cdot)$ [17]. Specifically, we pick up the pre-defined $f_v(\cdot)$ with v_j closest to w_j in $f(\cdot)$ (e.g. if $w_1 = 0.23$ and $l = 10$, then we select the one with $v_1 = 0.2$).

By utilizing the pre-computed probabilistic bounds with respect to $f_v(\cdot)$, we can obtain the upper bound $UB_G(P_i)$ of $G(P_i)$ as follows. Since $s \geq s_1$, it holds that $G(P_i) \leq Pr\{f(P_i) \geq s_1\}$. Moreover, as illustrated in Figure 5, since the line $f(P_i) = s_1$ is above line $f_v(P_i) = t_{0.7}$ within the uncertainty region $UR(P_i)$, we have $G(P_i) \leq Pr\{f(P_i) \geq s_1\} \leq Pr\{f_v(P_i) \geq t_{0.7}\} = 0.7$. That is, we can set $UB_G(P_i)$ to 0.7.

Similarly, for the lower bound $LB_G(P_i)$ of $G(P_i)$, due to $s \leq s_2$, it holds that $G(P_i) \leq Pr\{f(P_i) \geq s_1\}$. Furthermore, Figure 5 indicates that the line $f(P_i) = s_2$ is below line $f_v(P_i) = t_{0.3}$ in $UR(P_i)$. Therefore, we have $G(P_i) \geq Pr\{f(P_i) \geq s_2\} \geq Pr\{f_v(P_i) \geq t_{0.3}\} = 0.3$, resulting in $LB_G(P_i) = 0.3$.

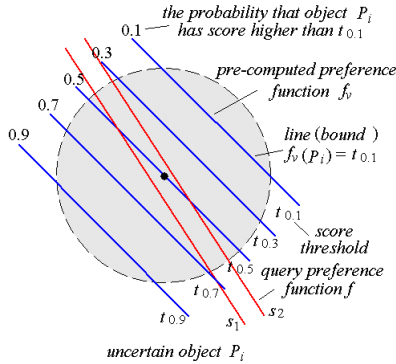


Figure 5: Lower and Upper Bounds of $G(P_i)$ for Object P_i

After introducing our intuition, now we formally give the bounds for $G(P_i)$. Specifically, given two preference functions $f_v(O) = \sum_{j=1}^d v_j \cdot O_j$ and $f(O) = \sum_{j=1}^d w_j \cdot O_j$, if $f(P_i) = s$, our goal is to find two tight (pre-computed) probabilistic bounds, $f_v(P_i^-)$ and $f_v(P_i^+)$, for $f_v(P_i)$, satisfying $f_v(P_i^-) \leq f_v(P_i) \leq f_v(P_i^+)$. Note that, the two probabilities associated with these two bounds exactly correspond to the lower or upper bound of $G(P_i)$. As in the previous example of Figure 5, the bound with score threshold $t_{0.7}$ is associated with the upper bound probability 0.7.

We have:

$$f_v(P_i) = f(P_i) - \sum_{j=1}^d (w_j - v_j) \cdot P_{ij} \quad (10)$$

where $P_i \in UR(P_i)$.

Without loss of generality, assume each coordinate P_{ij} of P_i is within an interval $[L_j, H_j]$ inferred by the uncertainty region $UR(P_i)$. Furthermore, we can obtain a even tighter bounding interval $[L'_j, H'_j]$ for P_{ij} . In particular, since it holds that $f(P_i) = \sum_{j=1}^d w_j \cdot P_{ij} = s$, we consider three cases for different values of w_j as follows.

• **Case 1** ($w_j = 0$). $L'_j = L_j$ and $H'_j = H_j$.

• **Case 2** ($w_j > 0$). Since it holds that $P_{ij} = \frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot P_{il}}{w_j}$, we have $\frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot H_l}{w_j} \leq P_{ij} \leq \frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot L_l}{w_j}$. Thus, we have $L'_j = \max\{L_j, \frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot H_l}{w_j}\}$ and $H'_j = \min\{H_j, \frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot L_l}{w_j}\}$.

• **Case 3** ($w_j < 0$). Similarly, we have $\frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot L_l}{w_j} \leq P_{ij} \leq \frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot H_l}{w_j}$. Thus, we obtain $L'_j = \max\{L_j, \frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot L_l}{w_j}\}$ and $H'_j = \min\{H_j, \frac{f(P_i) - \sum_{l=1 \wedge l \neq j}^d w_l \cdot H_l}{w_j}\}$.

By substituting either L'_j or H'_j into Eq. (10), we can obtain the lower and upper bounds (i.e. $f_v(P_i^-)$ and $f_v(P_i^+)$, respectively) for $f_v(P_i)$. In particular, when $w_j \geq v_j$, we let $P_{ij} = L'_j$ in order to obtain $f_v(P_i^+)$ and let $P_{ij} = H'_j$ to get $f_v(P_i^-)$. In contrast, when $w_j < v_j$, we set $P_{ij} = H'_j$ to obtain $f_v(P_i^+)$ and $P_{ij} = L'_j$ to get $f_v(P_i^-)$.

Therefore, after obtaining $f_v(P_i^-)$ and $f_v(P_i^+)$, we can find two pre-computed probabilistic bounds, and in turn obtain the associated probabilities (i.e. either $LB_G(P_i)$ or $UB_G(P_i)$). With $LB_G(P_i)$ and $UB_G(P_i)$, we can compute the bound of $S(N, m)$ by recursive function in Eq. (9), which is exactly the bound for $Pr_m(O)$. Note that, if we initially set m to k in Eq. (9), during the calculation of lower/upper bound for $S(N, k)$, we can obtain some side products from the intermediate results, that is, the lower/upper bound of $S(N, m)$ for any $m \in [1, k]$. Therefore, some redundant calculations with the same values of N and m can be significantly saved.

Finally, we present an optimization method for computing the lower / upper bound for $S(N, m)$ (i.e. that for $Pr_m(O)$). Recall in Eq. (9), that the recursion depth, N , is large, where N is the size of $\mathcal{D} \setminus \{O\}$. However, one interesting observation is that, as long as it holds that $UB_f(P_N) \leq s_1 = LB_f(O)$, we have $G(P_N) = 0$, and thus Eq. (9) can be simplified as $S(N, m) = S(N-1, m)$. In other words, object P_N would not affect the calculation of probability $S(N, m)$ if its score interval is entirely below that of object O . In this way, we can efficiently calculate $S(N, m)$ with only a small subset of objects in the database.

In summary, in order to calculate the lower/upper bound of probability $Pr_m(O)$ in Eq. (1), we can offline select some preference function $f_v(\cdot)$, with each of which n probabilistic bounds with score thresholds $t_{\beta_1}, t_{\beta_2}, \dots$, and t_{β_n} are pre-computed for each uncertain object O (like the one in Figure 5), where $Pr\{f_v(O) \geq t_{\beta_i}\} = \beta_i$. Then, upon the query's arrival, we compute $LB_G(P_i)$ and $UB_G(P_i)$ using these bounds as discussed above, and eventually obtain $LB_S(N, m)$ and $UB_S(N, m)$ (i.e. $LB_Pr_m(O)$ and $UB_Pr_m(O)$, respectively), which can be used in the *probabilistic pruning* method in Lemma 4.2.

4.4 Final Refinement

In previous subsections, we illustrate details of our pruning methods, including both *spatial* and *probabilistic pruning*. After the pruning process, the remaining data objects that cannot be pruned are called the *candidates* of the k -PRank query. Since our pruning methods can guarantee that all the discarded objects are not query results, the candidate set after pruning would contain all the query answers. However, *false positives* still exist in the candidate set (i.e. those objects that should not be query answers but are in the candidate set). Therefore, we have to refine the candidate set in order to obtain the actual query results. In particular, we need to compute the actual probability (in Eq. (1) or equivalently Eq. (6)) that each candidate is in the PRank result and remove those false positives.

Similar to the optimization method (mentioned in the last paragraph of Section 4.3), from Eq. (9), for any $s \in [s_1, s_2]$, as long as object P_N has its upper bound of score, $UB_f(P_N)$, never greater than s_1 , object P_N would not affect the calculation of $S(N, m)$ at all (since $S(N, m) = S(N - 1, m)$). Thus, we only need to calculate $S(N, m)$ (or $Pr_m(O)$) involving those objects that have their score upper bounds greater than s_1 (which are the same objects as those during *probabilistic pruning* to compute bounds).

Therefore, for the candidate set obtained after the *spatial pruning*, we can calculate the smallest score lower bound (e.g. s_1) among all candidates. Then, we further retrieve those objects that have their score upper bounds greater than s_1 , which are used first for *probabilistic pruning* and then the calculation of the actual probability (in Eq. (1) or Eq. (6)). This *conceptual* association of objects with each candidate to compute the probability (or bounds) is called the *bounding phase*, as mentioned earlier in our PRank framework (in Section 4.1).

For those remaining candidates that cannot be pruned, our final *evaluation phase* would calculate the actual probability in Eq. (1) (or Eq. (6)), applying the numerical method, similar to that used in [10, 9]. Since objects that are involved in the probability calculation are bounded by our optimization method, the resulting computation cost is expected to be much smaller, compared to the cost in the entire database using the original definition.

5. QUERY PROCESSING

In this section, we seamlessly integrate the pruning heuristics (i.e. *spatial* and *probabilistic pruning*) into our PRank query procedure. As mentioned earlier, since the only feasible method so far for answering PRank queries is the *linear scan*, which however incurs high cost, we use an R-tree [16] to index all the uncertain objects in the database and enhance the query efficiency. In particular, for each uncertain object O , we insert its uncertainty region $UR(O)$ into the R-tree \mathcal{I} , on which the PRank query is processed. Note that, in order to apply the *probabilistic pruning* method, we need to choose a number of preference functions that cover the whole space of possible queries, which has been addressed in PRE-FER [17]. Moreover, in the case of space limitations, the selection of several best preference functions under the constraint can also refer to [17], which is however not the focus of this paper. With every pre-defined preference function $f_v(\cdot)$, we pre-compute n probabilistic bounds for each uncertain object O , which can help obtain the bounds $LB_G(P_i)$ and $UB_G(P_i)$, and thus in turn $LB_S(N, m)$ and $UB_S(N, m)$.

In the sequel, Section 5.1 first illustrates how to prune intermediate entries in the R-tree index with the *spatial pruning*. Then, Section

5.2 presents the details of our PRank query procedure over the R-tree.

5.1 Pruning Intermediate Entries

In this subsection, we discuss the heuristics of pruning intermediate entries in the R-tree. Obviously, an intermediate entry of R-tree can be safely pruned if and only if all the uncertain objects under this entry can be discarded. In the context of our k -PRank query, we can safely prune an intermediate entry if the maximum score in this entry is still smaller than that of at least k uncertain objects. In particular, we summarize the rationale of pruning intermediate entries in the following lemma.

LEMMA 5.1. (*Spatial Pruning of Intermediate Entries*) Assume we have an R-tree index \mathcal{I} constructed over uncertain database \mathcal{D} , a user-specified preference function $f(\cdot)$ and an integer k . Let P_1, P_2, \dots , and P_k be any k uncertain objects that we have obtained so far, and e be an intermediate entry in \mathcal{I} with the maximum possible score $UB_f(e)$ (i.e. $\max\{f(x) \mid \forall x \in e\}$). Without loss of generality, assume $LB_f(P_k)$ is the smallest (i.e. k -th largest) lower bound of score among these k objects. Thus, entry e can be safely pruned if it holds that $UB_f(e) \leq LB_f(P_k)$.

Proof. Similar to Lemma 4.1, since the largest possible score, $UB_f(e)$, for any point in intermediate entry e is never greater than $LB_f(P_k)$, it indicates that at least k objects P_1, P_2, \dots , and P_k have scores higher than any object in e . Thus, it is guaranteed that entry e can be safely pruned. \square

According to Lemma 5.1, given an intermediate entry e , we only need to calculate the maximum possible scores $UB_f(e)$ in this entry with respect to the preference function $f(\cdot)$, and compare it with $LB_f(P_k)$. Note that, in the lemma, P_1, P_2, \dots , and P_k can be arbitrary k uncertain objects that we have accessed so far. Obviously, from the pruning condition $UB_f(e) \leq LB_f(P_k)$, the larger $LB_f(P_k)$ is, the higher the pruning ability is. Thus, we can set $LB_f(P_k)$ to the k -th largest lower bound of scores, for all the uncertain objects that we have obtained so far.

Furthermore, in order to compute the maximum score of an entry e , with respect to preference function $f(\cdot)$, we can use a hypersphere to bound the MBR of e , and then calculate the maximum possible score within the hypersphere. Specifically, let C_e be the center point of e , and r_e be the radius of the hypersphere. Similar to the computation of the score upper bound for an uncertainty region $UR(O)$, it is sufficient to replace C_{O_i} and r_O in Eq. (4) with C_e and r_e , respectively, in order to obtain $UB_f(e)$.

5.2 Query Procedure

After discussing the heuristics of pruning intermediate entries (*spatial pruning*) in the R-tree, in this subsection, we continue to illustrate the PRank query processing over the R-tree index in details.

In particular, Figure 6 presents the detailed query procedure, namely `PRank_Processing`, for PRank. Procedure `PRank_Processing` takes three parameters as input (i.e. an R-tree index \mathcal{I} over uncertain database \mathcal{D} , a query preference function $f(\cdot)$, and an integer k), and outputs k uncertain objects that are in the k -PRank query result.

In order to facilitate the PRank query processing, we maintain a *maximum heap* \mathcal{H} accepting entries in the form (e, key) , where e

```

Procedure PRank_Processing {
  Input: R-tree  $\mathcal{T}$  constructed over  $\mathcal{D}$ , preference function  $f(\cdot)$ , integer  $k$ 
  Output:  $k$  objects in the PRank query result
  (1) initialize a max-heap  $\mathcal{H}$  accepting entries in the form  $(e, key)$ 
  (2)  $S_{cand} = \Phi, S_{rfn} = \Phi, kLB\_score = -\infty$ ;
  (3) insert  $(root(\mathcal{T}), 0)$  into heap  $\mathcal{H}$ 
  (4) while  $\mathcal{H}$  is not empty
  (5)    $(e, key) = \text{de-heap } \mathcal{H}$ 
  (6)   if  $key \leq kLB\_score$ , then break; // Lemma 5.1
  (7)   if  $e$  is a leaf node
  (8)     for each uncertain object  $O$  in  $e$  (sorted in descending order of  $LB\_f(O)$ )
  (9)       if  $|S_{cand}| < k$ 
  (10)         $S_{cand} = S_{cand} \cup \{O\}$ 
  (11)        if  $LB\_f(O) > kLB\_score$ 
  (12)           $kLB\_score = LB\_f(O)$ 
  (13)        else //  $|S_{cand}| \geq k$ 
  (14)          if  $UB\_f(O) \geq kLB\_score$  // spatial pruning, Lemma 4.1
  (15)            if  $LB\_f(O) > kLB\_score$ 
  (16)               $kLB\_score = LB\_f(O)$  // update  $kLB\_score$ 
  (17)              let  $O'$  be uncertain object(s) in  $S_{cand}$ 
  (18)                satisfying  $UB\_f(O') < kLB\_score$ 
  (19)                move object(s)  $O'$  from  $S_{cand}$  to  $S_{rfn}$ 
  (20)               $S_{cand} = S_{cand} \setminus \{O'\}$ 
  (21)            else // intermediate node
  (22)              for each entry  $e_i$  in  $e$ 
  (23)                if  $UB\_f(e_i) > kLB\_score$  // Lemma 5.1
  (24)                  insert  $(e_i, UB\_f(e_i))$  into heap  $\mathcal{H}$ 
  (25)                else
  (26)                   $S_{rfn} = S_{rfn} \cup \{e_i\}$ 
  (27)             $rlt = \text{Probabilistic\_Pruning}(S_{cand}, S_{rfn}, f(\cdot), k)$ ;
  (28)             $rlt = \text{Evaluation}(rlt, S_{cand}, S_{rfn}, f(\cdot), k)$ ;
  (29)          return  $rlt$ 
}

```

Figure 6: PRank Query Processing

is either a data object or an MBR node, and key is defined as the maximum score $UB_f(e)$ of any point in e , with respect to preference function $f(\cdot)$ (line 1). Moreover, we also keep a candidate set S_{cand} containing candidates of the k -PRank query, and a refinement set, S_{rfn} , of objects/MBRs that are involved in the *probabilistic pruning* and/or the final evaluation phase. Furthermore, a parameter kLB_score is initialized by negative infinity, representing the k -th largest score lower bound, for all the objects that have been seen so far (line 2).

Our query procedure PRank_Processing traverses the R-tree index by accessing heap entries in descending order of their keys. Specifically, we first insert the root $root(\mathcal{T})$ of R-tree \mathcal{T} into heap \mathcal{H} (line 3). Every time we pop out an entry (e, key) from heap \mathcal{H} with the largest key (lines 4-5). Recall that, the key, key , is defined as the maximum possible score of an object or any point in an MBR node. Thus, if it holds that largest key, key , in heap \mathcal{H} is never greater than kLB_score (i.e. the k -th largest score lower bound that we have seen so far), then all the remaining entries in heap \mathcal{H} can be safely pruned and the traversal of R-tree can be terminated (line 6); otherwise, we need to check entry e as shown below.

When the entry e we obtain from heap \mathcal{H} is a leaf node, we verify uncertain objects O in e in the descending order of $f(O)$ (for the sake of achieving higher pruning power, lines 7-8). In the case where the number of candidates in S_{cand} is less than k , we simply add object O to S_{cand} and moreover update the variable kLB_score (lines 9-12). Furthermore, if the size of candidate set is greater than or equal to k , then we perform the *spatial pruning* as given in Lemma 4.1 (line 14). That is, if the upper bound of score for object O is smaller than kLB_score , then there exist at least k objects in S_{cand} such that their scores are higher than $f(O)$, and object O can thus be safely pruned; otherwise (i.e. $UB_f(O) \geq kLB_score$), we have to update the variable

kLB_score (lines 14-18) as well as the candidate set S_{cand} (line 19). In particular, if it holds that $LB_f(O) > kLB_score$, then we can obtain an even higher kLB_score , and meanwhile remove those object(s) $O' \in S_{cand}$ such that $UB_f(O')$ is smaller than the newly updated kLB_score (lines 15-18). Note that, although O' is guaranteed not to be the PRank result by our pruning method, it may still be needed for calculating the probability in Eq. (1) during the *probabilistic pruning* or evaluation phase. Therefore, we do not discard object(s) O' , but insert it (them) into the refinement set S_{rfn} (line 18).

When the entry e is an intermediate node in the R-tree, we check the pruning condition for each entry e_i in e , applying Lemma 5.1 (lines 20-22). Specifically, for each entry e_i in e , in case the maximum possible score $UB_f(e_i)$ in e_i is greater than kLB_score , we have to insert e_i into heap \mathcal{H} , in the form $(e_i, UB_f(e_i))$, for later access (since there may exist PRank answers in e_i , lines 22-23); otherwise, we can safely prune entry e_i . However, similar to the case of data object O' (line 17), we add e_i to the refinement set S_{rfn} (lines 24-25), since objects in e_i may affect the calculation of probability for candidates during the *probabilistic pruning* or evaluation phase.

Next, we discuss the procedure Probabilistic_Pruning invoked by line 26 in procedure PRank_Processing. After the *spatial pruning*, we obtain a set, S_{cand} , of PRank candidates. Moreover, the refinement set S_{rfn} contains objects/MBRs that may help prune/refine these candidates (note that S_{rfn} does not contain any PRank candidates). Since the probability integration in Eq. (1) is very complex and costly, we perform the *probabilistic pruning* to further reduce the candidate size.

The basic rationale of procedure Probabilistic_Pruning is as follows. First, we retrieve those data objects in S_{rfn} that may affect the calculation of the lower/upper bound probability for PRank candidates in S_{cand} (according to the optimization method mentioned in Section 4.3). As a second step, we compute the lower/upper bound of probability for each candidate and apply the probabilistic pruning in Lemma 4.2.

In particular, Figure 7 shows the details of our pruning procedure over the candidate set S_{cand} . The procedure Probabilistic_Pruning first finds the smallest score lower bound, min_score , within the candidate set S_{cand} (line 1). Then, we search over the two sets S_{cand} and S_{rfn} , and retrieve all the data objects O' such that $UR_f(O') \geq min_score$ (for any intermediate node e in S_{rfn} , its subtrees are traversed in a best-first manner), which are inserted into an initially empty set R (lines 3-7). As mentioned in the last paragraph of Section 4.3 (i.e. the optimization method), only those objects in R are necessary to be checked for the probability calculation of each candidate. After that, for each candidate $O \in S_{cand}$, we compute the lower and upper bounds of $Pr_m(O)$ (for all $m \in [1, k]$) only involving objects in R (lines 8-9). Finally, we apply the *probabilistic pruning* method in Lemma 4.2 to prune candidates in S_{cand} , utilizing the computed lower/upper bound (line 10). The remaining candidates that cannot be pruned are returned as output (line 11).

In line 27 of procedure PRank_Processing, after the *probabilistic pruning*, we further refine the returned candidates (in the set rlt) in procedure Evaluation. Specifically, for each $m \in [1, k]$, we compute the actual probability of each candidate that has the m -th largest score (Eq. (1) or equivalently Eq. (6)), and select the one

Procedure Probabilistic_Pruning {
Input: candidate set S_{cand} , refinement set S_{rfn} , preference function $f(\cdot)$, integer k
Output: candidate set rll for the PRank query
(1) let min_score be the minimum score lower bound $LB_f(O)$ for all the candidates O in S_{cand}
(2) $R = \Phi$;
(3) for each object/entry e in $S_{cand} \cup S_{rfn}$
(4) if e is object and $UB_f(e) \geq min_score$
(5) $R = R \cup \{e\}$
(6) else // e is a node
(7) add all the objects O' under e to R satisfying $UB_f(O') \geq min_score$
(8) for each candidate $O \in S_{cand}$
(9) calculate the upper and lower bounds of $Pr_m(O)$ over R
// Section 4.3
// Lemma 4.2
(10) use the lower/upper bound of $Pr_m(O)$ to prune candidates in S_{cand}
(11) return the remaining candidates
}

Figure 7: Procedure for Probabilistic Pruning

with the highest probability as the final result (i.e. $PRank_m$). Note that, similar to the *probabilistic pruning*, the calculation of Eq. (1) or Eq. (6) only needs to involve those objects in R , as computed in procedure Probabilistic_Pruning, compared to that over the entire database in the original definition.

In summary, since our query procedure PRank_Processing traverses the R-tree index in a best-first manner, in which each intermediate node is accessed at most once, our query processing is efficient. Moreover, since we apply both *spatial* and *probabilistic pruning* methods, the resulting number of candidates should be small, which will be confirmed later in our experiments.

6. EXPERIMENTAL EVALUATION

In this section, we empirically evaluate the query performance of our proposed approach to answer the *probabilistic ranked query* (PRank). In particular, since there are no real data sets available, we synthetically generate uncertain objects in a d -dimensional data space $\mathcal{U} = [0, 1]^d$, like the ones in [9, 12, 33]. Specifically, for each uncertain object O , we first decide the center location C_O of its *uncertainty region* $UR(O)$, and then randomly produce the radius r_O of $UR(O)$ within $[r_{min}, r_{max}]$. In order to simulate the distribution of object position, for each uncertain object O , we generate 100 samples contained in its uncertainty region $UR(O)$, following either *Uniform* or *Gaussian* distribution (with mean C_{O_i} and variance $2r_O/5$ along the i -th dimension). For brevity, we denote lU (lS) as the data set with center *locations* C_O of *Uniform* (*Skew* with skewness 0.8) distribution, rU (rG) as that with *radius* $r_O \in [r_{min}, r_{max}]$ of *Uniform* (*Gaussian*), with mean $(r_{min} + r_{max})/2$ and variance $(r_{max} - r_{min})/5$ distribution, and pU (pG) as that with object *position* (in the uncertainty region) of *Uniform* (*Gaussian*) distribution. Thus, with different combinations, we can obtain eight types of synthetic data sets, $lUrUpU$, $lUrUpG$, $lUrGpU$, $lUrGpG$; $lSrUpU$, $lSrUpG$, $lSrGpU$, and $lSrGpG$. Note that, for data sets with other distribution parameters (e.g. mean and variance for *Gaussian*, or skewness for *Skew*), the experimental results are similar, and we would not present all of them here. After generating data sets, we index them with R-tree [16], on which PRank queries can be processed, where the page size is set to 1KB.

In order to evaluate the PRank query, we randomly generate 100 query preference functions, $f(O) = \sum_{i=1}^d w_i \cdot O_i$, as follows. For each weight w_i ($1 \leq i \leq d$), we randomly pick up a value within domain $(0, 1]$, following specific (*Uniform*, *Gaussian*, or *Skew*) distribution (note that the results with negative weights are similar and thus omitted due to space limit). To enable the *probabilis-*

Parameter	Values
$[r_{min}, r_{max}]$	[0, 0.01], [0, 0.02], [0, 0.03] , [0, 0.04], [0, 0.05]
k	2, 4, 6 , 8, 10, 20
d	2, 3, 4, 5
N	10K, 20K, 30K , 40K, 50K, 100K
l	5, 8, 10 , 20, 40
n	5, 8, 10 , 15, 20
weight distribution	Uniform , Gaussian, Skew

Table 2: The Parameter Settings

tic pruning method, we pre-select preference functions, $f_v(O) = \sum_{i=1}^d v_i \cdot O_i$ ($w \in (0, 1]$), with the *weight resolution* l [17]. Furthermore, with respect to any pre-defined preference function $f_v(\cdot)$, for each uncertain object O , we pre-compute n probabilistic bounds as discussed in Section 4.3. That is, we obtain n score thresholds $t_{\beta_1}, t_{\beta_2}, \dots, t_{\beta_n}$, such that $Pr\{f_v(O) \geq t_{\beta_i}\} = \beta_i$, for $1 \leq i \leq n$. For the sake of simplicity, here we let $\beta_i = i/n$.

Throughout our experiments, we measure the *wall clock time* and the number of PRank candidates to be refined (in the *evaluation phase*), in order to study the efficiency and effectiveness of our PRank query procedure as well as the pruning methods. In particular, the *wall clock time* is the time cost of retrieving PRank candidates before the final evaluation phase, where we incorporate the cost of each page access (i.e. one I/O) by penalizing 10ms [35, 36]. Thus, the *wall clock time* includes both the CPU time and I/O cost. Moreover, the number of PRank candidates to be refined is calculated by summing up the number of candidates with rank m , for all $m \in [1, k]$ (i.e., each object might be counted multiple times).

To the best of our knowledge, no previous work on PRank problem has been studied in the context of uncertain databases. Therefore, the most naive method is the *linear scan*, which sequentially scans the entire uncertain database object by object on disk and finds out the answer to the PRank query. To give an example, assume an uncertain database contains 30K 3D data objects. If the page size is 1KB and a floating number takes up 4 bytes, the I/O cost of even one linear scan requires about 3.6 ($= \frac{10ms}{1000ms/s} \times \frac{30K \times 3 \times 4}{1K}$) seconds which is much greater than that of our method as shown below. In the sequel, we also illustrate the *speed-up ratio* of our approach compared with the linear scan, which is defined as the required *wall clock time* of the linear scan (underestimated by only considering the I/O cost) divided by that of our method. As we will see, our method can outperform the linear scan in terms of the *wall clock time* by orders of magnitude.

In each of the subsequent experiments, we vary the value of one parameter while fixing others to their default values. Table 2 summarizes the parameter settings, where the default values of parameters are in bold font. All our experiments are conducted on a Pentium IV 3.2GHz PC with 1G memory, and the reported results are the average of 100 queries.

6.1 Performance vs. Radius Range $[r_{min}, r_{max}]$

In the first set of experiments, we evaluate the efficiency and effectiveness of our proposed PRank query procedure using data sets that have different radius ranges $[r_{min}, r_{max}]$ for uncertain objects. In particular, we test the *wall clock time* and the number of PRank candidates to be refined over eight types of data sets, with ranges $[r_{min}, r_{max}]$ set to $[0, 0.01]$, $[0, 0.02]$, $[0, 0.03]$, $[0, 0.04]$, and $[0, 0.05]$, where other parameters are set to their default values in Table 2 (i.e. the data size N is 30K, the dimensionality d is 3, the number, k , of query results (for k -PRank queries) is 6, the weight

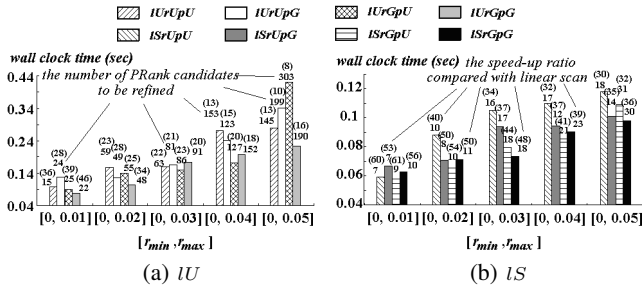


Figure 8: Performance vs. Radius Range $[r_{min}, r_{max}]$

resolution, l , is 10 and the number of probabilistic bounds, n , is set to 10). Furthermore, the query preference function is generated by selecting a random value $w_i \in (0, 1]$ as weight following *Uniform* distribution. We will show later the effect of parameters l and n on the query performance, as well as other weight distributions (e.g. *Gaussian* and *Skew*).

Figure 8 illustrates the experimental results by varying the radius range of 8 data sets. Figure 8(a) shows the results of 4 *LU* data sets and Figure 8(b) presents that of the remaining 4 *LS* ones. Note that, in all figures of experimental results, the numbers without brackets over columns indicate the numbers of PRank candidates to be refined (i.e. calculating the actual probability integration in Eq. (1)) in the evaluation phase, whereas those in brackets over columns are the *speed-up ratio* of our methods, compared with the *linear scan* (i.e. underestimated by only considering the I/O cost).

From figures, we find that, when the radius range $[r_{min}, r_{max}]$ varies from $[0, 0.01]$ to $[0, 0.05]$, the *wall clock time* of retrieving the PRank candidates increases. This is reasonable, since larger uncertainty regions of data objects would make the ranks of uncertain objects much more indistinguishable. Thus, more PRank candidates will be included in the candidate set, as confirmed by numbers in figures, which makes the pruning process more costly. In general, however, the *wall clock time* of our PRank procedure is very efficient (less than 0.44 second for *LU* data sets and 0.12 second for *LS* data sets). Observe that, *LS* data sets require much less processing time than *LU*. This is because most data in *LS* data sets tend to have small values along each dimension (i.e. locating towards the origin), and only a few (sparse) data locate in spaces with high scores, which thus has much less PRank candidates to check the pruning conditions, compared with *LU* data sets. It is interesting to see the same phenomena in all the subsequent experiments. Furthermore, for all the data sets, the number of PRank candidates to be refined remains small, compared with the total data size $30K$; moreover, the *speed-up ratio* is significant, indicating that our method can outperform the *linear scan* by orders of magnitude.

6.2 Performance vs. k

As a second step, we evaluate the query performance of our proposed approach, with respect to different k values specified by PRank queries. In particular, Figure 9 illustrates the *wall clock time* of PRank query procedure, over eight types of data sets, by varying k from 2 to 20, where other parameters are set to their default values.

From the experimental results, the pruning efficiency of our proposed method is not very sensitive to the k value, in terms of the *wall clock time*. As we can see, the required *wall clock time* is

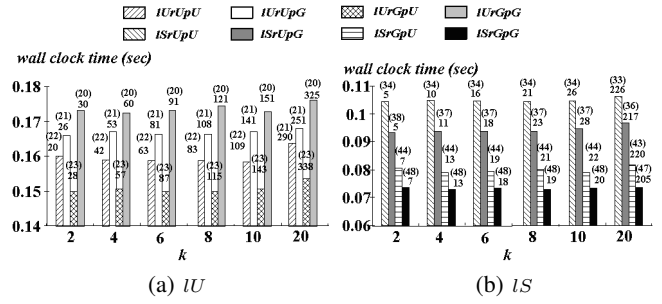


Figure 9: Performance vs. k

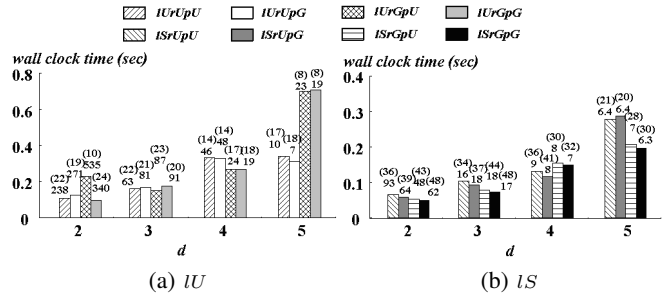


Figure 10: Performance vs. Dimensionality d

merely less than 0.18 (0.11) second for *LU* (*LS*) data sets in Figure 9(a) (Figure 9(b)). When k grows larger, the number of resulting PRank candidates to be refined increases (since we maintain candidates with rank m for each m from 1 to k). Moreover, from the *speed-up ratio* in figures, we find that our approach can perform orders of magnitude better than the *linear scan*.

6.3 Performance vs. Dimensionality d

Next, we study the effect of dimensionality d for data sets on the PRank query performance. Specifically, we present the experimental results over eight data sets in Figure 10, where other parameters are set to their default values.

Note that, since the data size N in our experiments is fixed to $30K$, data in 2D space will be much more dense than that in higher dimensional (e.g. 5D) space. Thus, the number of resulting PRank candidates would decrease with the increasing dimensionality, as confirmed by the results in figures. Moreover, when $d = 2$, data set $IUrGpG$ (in Figure 10(a)) incurs higher *wall clock time* than that of 3D case, due to the high cost of retrieving more candidates in the 2D space.

For the reason of the dimensionality problem with multidimensional index (e.g. the query efficiency of R-tree [16] degrades with the increasing dimensionality [38]), the *wall clock time* over eight data sets basically increases when the dimensionality varies from 2 to 5. However, the required time is still very low, that is, at most 0.7 and 0.3 seconds for *LU* and *LS* data sets, respectively. Compared with the *linear scan*, the *speed-up ratio* is by order(s) of magnitude.

6.4 Performance vs. Data Size N

In this subsection, we test the scalability of our proposed PRank query procedure, as a function of the data size, N . Specifically, we vary the data size N from $10K$ to $100K$, and test the query performance over eight data sets, where other parameters are set to their default values. From figures, we find that the *wall clock time* increases when the data size N becomes large. The reason

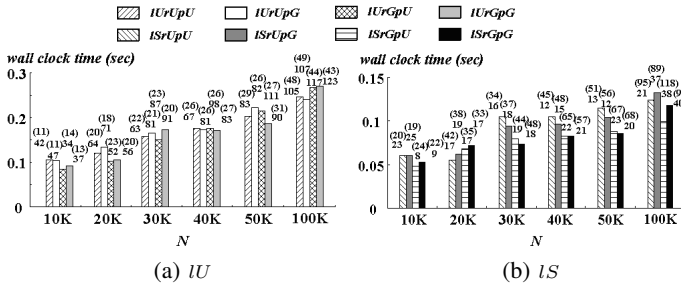


Figure 11: Performance vs. Data Size N

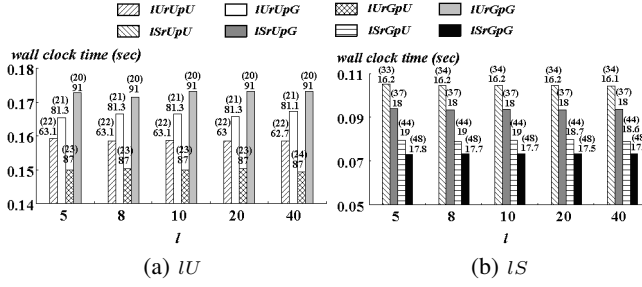


Figure 12: Performance vs. Weight Resolution l

is that, large data size results in high density of data objects in the space, which incurs high cost to filter out unqualified objects. The total required *wall clock time*, however, is less than 0.3 and 0.15 for *IU* and *IS* data sets, as illustrated in Figures 11(a) and 11(b), respectively.

In general, the number of PRank candidates increases smoothly with respect to the increasing data size; moreover, the *speed-up ratio* of our method compared with the linear scan also increases when N grows. These facts indicate the good scalability of our proposed method in answering PRank queries, with respect to the data size N .

6.5 Performance vs. Weight Resolution l

In this subsection, we study the effect of weight resolution l on the PRank query performance. Recall that, we divide the weight domain $(0, 1]$ into l intervals of equal size, and preference functions $f_v(\cdot)$ are pre-selected accordingly [17]. Then, given a query preference function $f(\cdot)$, we always choose one pre-computed preference function $f_v(\cdot)$ with weights the most similar to that of $f(\cdot)$. Intuitively, the larger l is, the more similar the chosen $f_v(\cdot)$ is to $f(\cdot)$.

Figure 12 illustrates the experimental results by varying the weight resolution l from 5 to 40 over 8 data sets, where other parameters are set to their default values. As expected, for some data sets, the number of PRank candidates to be refined decreases when l grows. However, the increasing trend is not very fast with respect to l , which indicates that our query procedure is not sensitive to l very much.

With the increasing l values, the *wall clock time* of all the 8 data sets remains approximately the same, which is only less than 0.18 (0.11) second for *IU* (*IS*) data sets in Figure 12(a) (Figure 12(b)). Similarly, the *speed-up ratio* is nearly constant with respect to l , which shows the good performance of our method compared with the linear scan.

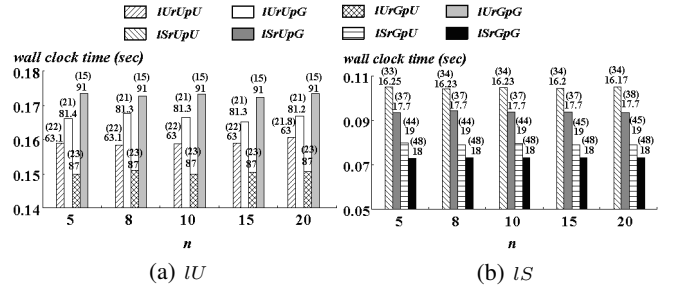


Figure 13: Performance vs. The Number of Probabilistic Bounds n

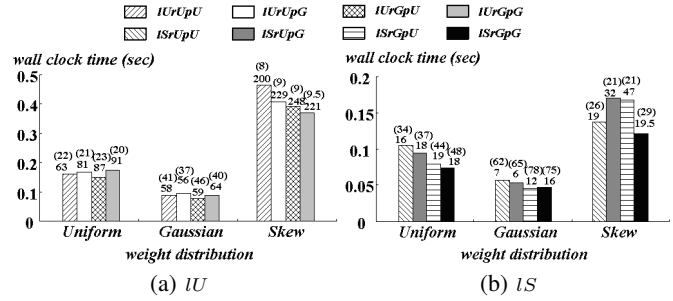


Figure 14: Performance vs. Weight Distribution

6.6 Performance vs. Number of Probabilistic Bounds n

In this set of experiments, we test the PRank query performance of our approach over 8 data sets in Figure 13, by varying the number of probabilistic bounds, n , from 5 to 20, where other parameters are set to their default values.

Recall that, in order to perform the *probabilistic pruning* method, we need to pre-compute n probabilistic bounds so that the lower /upper bounds of probabilities can be calculated. Intuitively, a larger number of pre-computed bounds would give tighter bounds and higher pruning ability. From figures, we can see that, when n increases, some data sets indeed show slightly fewer PRank candidates, which result in lower refinement cost (for other data sets, the decrease is not that significant). Moreover, the *wall clock time* and *speed-up ratio* are not very sensitive to the n values. Note that, since a large n would also lead to large space to store pre-computed information, the experimental results shown in Figure 13 indicate that we can choose a small n value (e.g. 10) to make a trade-off between space and query efficiency.

6.7 Performance vs. Weight Distribution

Finally, we present the query performance of our proposed approach with different weight distributions. Apart from the *Uniform* weight distribution that we used in previous experiments, we also compare it with *Gaussian* and *Skew* weight distributions. In particular, Figure 14 illustrates the *wall clock time* of PRank query processing over 8 types of data sets, using *Uniform*, *Gaussian*, and *Skew* weight distributions, where other parameters are set to their default values.

From figures, we can see that our method is efficient with different weight distributions, that is, less than 0.5 and 0.2 seconds are required for *IU* and *IS* data sets, as shown in Figures 14(a) and 14(b), respectively. The results with *Skew* distribution require more *wall clock time* (thus lower *speed-up ratio*), since more PRank candidates are retrieved from the index.

7. CONCLUSIONS

Due to the inherent uncertainty of data in many real-world applications, query processing over these uncertain data becomes more and more important. In the traditional “certain” database, the ranked query has many applications like decision making, recommendation raising, and data mining tasks. Previous query processing methods, however, are inapplicable to the handle uncertain data. Motivated by this, in this paper, we propose a formal definition of the ranked query over uncertain databases, namely the *probabilistic ranked query* (PRank), and design two effective pruning methods, *spatial* and *probabilistic*, to facilitate reducing the PRank search space. Moreover, we seamlessly integrate these two pruning heuristics into our PRank query procedure. Extensive experiments have verified the efficiency and effectiveness of our proposed approach, in terms of the *wall clock time* and the number of PRank candidates to be refined. Since our work assumes linear preference functions, in future, it would be interesting to consider arbitrary monotonic functions. Furthermore, another interesting direction is to study discrete PRank queries with uncertain categorical data.

8. ACKNOWLEDGMENT

Funding for this work was provided by Hong Kong RGC Grant No. 611907, National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303000, and the NSFC Key Project Grant No. 60533110.

9. REFERENCES

- [1] R. Akbarinia, E. Pacitti, and P. Valduriez. Best position algorithms for top- k queries. In *VLDB*, pages 495–506, 2007.
- [2] L. Antova, C. Koch, and D. Olteanu. Query language support for incomplete information in the MayBMS system. In *VLDB*, pages 1422–1425, 2007.
- [3] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top- k algorithms. In *VLDB*, pages 914–925, 2007.
- [4] C. Böhm, A. Pryakhin, and M. Schubert. The Gauss-tree: efficient object identification in databases of probabilistic feature vectors. In *ICDE*, page 9, 2006.
- [5] N. Bruno, S. Chaudhuri, and L. Gravano. Top- k selection queries over relational databases: Mapping strategies and performance evaluation. *TODS*, 2002.
- [6] K. C.-C. Chang and S.-W. Hwang. Minimal probing: supporting expensive predicates for top- k queries. In *SIGMOD*, pages 346–357, 2002.
- [7] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The Onion technique: indexing for linear optimization queries. In *SIGMOD*, pages 391–402, 2000.
- [8] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.
- [9] R. Cheng, D. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. In *TKDE*, volume 16, pages 1112–1127, 2004.
- [10] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.
- [11] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *VLDB*, pages 1271–1274, 2005.
- [12] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, pages 876–887, 2004.
- [13] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top- k queries using views. In *VLDB*, 2006.
- [14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.
- [15] A. Faradjian, J. Gehrke, and P. Bonnet. Gadt: A probability space ADT for representing and querying the physical world. In *ICDE*, pages 201–211, 2002.
- [16] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.
- [17] V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, 2001.
- [18] V. Hristidis and Y. Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDBJ*, 13(1):49–70, 2004.
- [19] M. Hua, J. Pei, A. W.-C. Fu, X. Lin, and H.-F. Leung. Efficiently answering top- k typicality queries on large databases. In *VLDB*, pages 890–901, 2007.
- [20] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top- k join queries in relational databases. *VLDBJ*, 13(3):207–221, 2004.
- [21] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *DASFAA*, 2006.
- [22] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, 2007.
- [23] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. RankSQL: Query algebra and optimization for relational top- k queries. In *SIGMOD*, pages 131–142, 2005.
- [24] C. Li, M. Wang, L. Lim, H. Wang, and K. C.-C. Chang. Supporting ranking and clustering as generalized order-by and group-by. In *SIGMOD*, pages 127–138, 2007.
- [25] V. Ljosa and A. K. Singh. APLA: indexing arbitrary probability distributions. In *ICDE*, pages 247–258, 2007.
- [26] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: Top- k keyword query in relational databases. In *SIGMOD*, pages 115–126, 2007.
- [27] A. Marian, N. Bruno, and L. Gravano. Evaluating top- k queries over web-accessible databases. *TODS*, 29(2):319–362, 2004.
- [28] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.
- [29] C. Re, N. Dalvi, and D. Suciu. Efficient top- k query evaluation on probabilistic data. In *ICDE*, 2007.
- [30] R. Ross, V. S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, 2005.
- [31] A. D. Sarma, O. B., A. Y. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, page 7, 2006.
- [32] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top- k query processing in uncertain databases. In *ICDE*, 2007.
- [33] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. K., and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, pages 922–933, 2005.
- [34] Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Inf. Syst.*, 32(3):424–445, 2007.
- [35] Y. Tao, D. Papadias, and X. Lian. Reverse k NN search in arbitrary dimensionality. In *VLDB*, pages 744–755, 2004.
- [36] Y. Tao, D. Papadias, X. Lian, and X. Xiao. Multidimensional reverse k NN search. In *VLDBJ*, 2005.
- [37] M. Theobald, G. Weikum, and R. Schenkel. Top- k query evaluation with probabilistic guarantees. In *VLDB*, pages 648–659, 2004.
- [38] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *PODS*, pages 161–171, 1996.
- [39] D. Xin, C. Chen, and J. Han. Towards robust indexing for ranked queries. In *VLDB*, 2006.
- [40] D. Xin, J. Han, and K. C.-C. Chang. Progressive and selective merge: computing top- k with ad-hoc ranking functions. In *SIGMOD*, pages 103–114, 2007.
- [41] K. Yi, F. Li, D. Srivastava, and G. Kollis. Efficient processing of top- k queries in uncertain databases. In *ICDE*, pages 385–394, 2000.
- [42] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis. Top- k spatial preference queries. In *ICDE*, pages 1076–1085, 2007.
- [43] M. L. Yiu and N. Mamoulis. Efficient processing of top- k dominating queries on multi-dimensional data. In *VLDB*, pages 483–494, 2007.