Monochromatic and Bichromatic Reverse Skyline Search over 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

Reverse skyline queries over uncertain databases have many important applications such as sensor data monitoring and business planning. Due to the existence of uncertainty in many real-world data, answering reverse skyline queries accurately and efficiently over uncertain data has become increasingly important. In this paper, we model the *probabilistic reverse skyline* query on uncertain data, in both monochromatic and bichromatic cases, and propose effective pruning methods to reduce the search space of query processing. Moreover, efficient query procedures have been presented seamlessly integrating the proposed pruning methods. Extensive experiments have demonstrated the efficiency and effectiveness of our proposed approach with various experimental settings.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Management—*Database applications, Spatial databases and GIS*; H.3.3 [**Information Systems**]: Information Storage and retrieval—*Information search and retrieval, Search process*

General Terms

Algorithms, Design, Experimentation, Performance, Theory

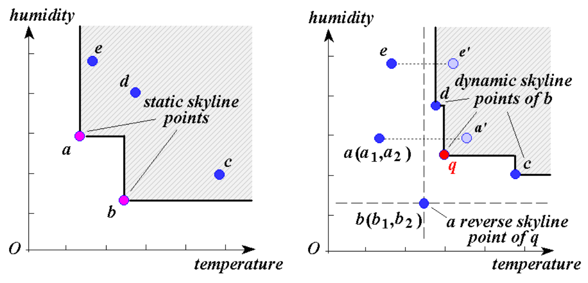
Keywords

Uncertain database, monochromatic reverse skyline, bichromatic reverse skyline

1. INTRODUCTION

Recently, the *skyline query* has played an increasingly important role in many real applications, such as multi-criteria decision making, market analysis, environmental surveillance and quantitative economics research. Given a d-dimensional database $\mathcal{D}$, object $p(p_1, p_2, ..., p_d)$ *dominates* object $o(o_1, o_2, ..., o_d)$, if it holds that: 1) $p_i \leq o_i$ for each dimension $1 \leq i \leq d$, and 2) $p_j < o_j$ for at least one dimension $1 \leq j \leq d$, where o_i and p_i are the attributes of o and p, respectively. A traditional skyline (a.k.a *static skyline query*) retrieves all the objects in the database whose attribute vectors are not dominated by others. For static skylines, once the database is given, the skylines as well as the dominant relationships are fixed. Figure 1(a) illustrates an example of static skyline over data collected sensors, where five data points, $a \sim e$, represent five $\langle temperature, humidity\rangle$-pairs, respectively. From the figure, data point a dominates points d and e, since a has lower temperature and smaller humidity than d and e. Similarly, sensor data b also dominates points c and d. Since a and b are not dominated by any other points in the space, they are *static skyline points* [2].

In addition to static skyline, *dynamic* (*relative*) *skyline queries* have been proposed [20, 10, 9], where the attributes of each object are dynamically calculated based on query predicates. Specifically, each d-dimensional object p is mapped to a new d'-dimensional point $p' = \langle f_1(p), f_2(p), ..., f_{d'}(p)\rangle$, where f_i is a dynamic function. In this paper, for the sake of simplicity, we assume that $d' = d$ and for a given query point q $f_i(p) = |q_i - p_i|$, which are also used by Dellis and Seeger [9]. Note, however, that the proposed approaches still hold for a more general class of functions [9]. We give an example with these assumptions below.



(a) Static Skyline (b) Dynamic/Reverse Skyline

Figure 1: 2D Sensor Examples of Skyline

Figure 1(b) illustrates the dynamic skyline in the example of sensor data. Assume point $b(b_1, b_2)$ is a query object. For any data point, for instance, $a(a_1, a_2)$, its dynamic attributes are computed from its (absolute) humidity and temperature differences from query object b, that is, $|a_1 - b_1|$ and $|a_2 - b_2|$, respectively. Since we consider the absolute differences, as shown in the figure, we map all the points onto the first quadrant of a new coordinate space with origin b (e.g. mapping a and e to a' and e', respectively; c and d remain the same since they are already in the first quadrant). The dynamic skyline query is issued to obtain all the sensors whose dynamic attribute vectors are not dominated by others (in that quadrant). In our example, sensor data d, q, and c are the *dynamic skyline points* of b. Semantically, these dynamic skylines are close to the query point b for all attributes and better (closer to b) than other non-skyline points along at least one attribute (e.g. temperature or humidity).

In Figure 1(b), q is one of dynamic skyline points of b, and b is called a *reverse skyline* [9] of q. Specifically, given a query pattern q, a *reverse skyline query* obtains data objects whose dynamic skyline contains q. As shown in the example, the query pattern q is closer to its reverse skyline b than other sensor data along at least one dimension. The reverse skyline query is very useful for environmental monitoring applications. For example, in an application of monitoring the forest, a number of sensors are deployed in a monitoring area to collect data such as temperature and humidity. Assume the query q represents the value thresholds of possible fire disaster on different data attributes. If we find that the number of reverse skylines of q within a monitoring area is greater than an alarm threshold, it indicates that, rather than individual malfunctioned sensor data, many sensing values are in the dangerous state (i.e. either temperature is too high or the air is too dry). In this case, we can take immediate actions to prevent the disaster.

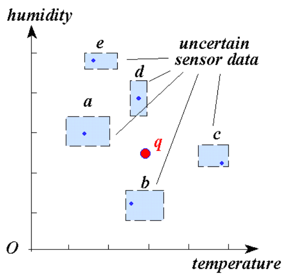


Figure 2: Monochromatic Probabilistic Reverse Skyline Query

In reality, sensor data often contain noises resulting from environmental factors, transmission delay, or device failures. In the literature of uncertain databases, each sensor data can be considered as an *uncertainty region* [6, 28]. Figure 2 shows an example of reverse skyline over uncertain database, in which each $\langle temperature, humidity \rangle$-pair locates in an uncertainty region (shaded region with rectangular shape) with any distribution. Thus, the goal of this work is to find out reverse skylines of q among uncertain objects.

Although previous work [9] has studied the reverse skyline search problem over precise data, they cannot directly handle the uncertain database. The reason is that, each object is now represented by an uncertainty region, instead of a precise point, and the dynamic attributes (with respect to query and data objects) are thus variables rather than exact values. In this paper, we define the reverse skyline query over uncertain database, namely *monochromatic probabilistic reverse skyline* (MPRS), which retrieves all the uncertain objects in a database whose dynamic skylines contain a user-specified query object with a probability greater than or equal to a user-specified threshold. Note that, the MPRS query has clear advantages that its answers are invariable to scales in different dimensions and it does not require users to specify any heuristic query parameters, which invalidate other query types like nearest neighbor or range query in our examples above.

Apart from the MPRS query over one single data set, we also study the *bichromatic probabilistic reverse skyline* (BPRS) query, involving two distinct data sets. Specifically, given two distinct uncertain databases $\mathcal{A}$ and $\mathcal{B}$ and a query object q, a BPRS query obtains those points $o \in \mathcal{A}$ such that the dynamic skyline of o in the data set B contain q. Note that, in both MPRS and BPRS, q can be either certain or uncertain object. For simplicity, we first use a certain query object q to illustrate the MPRS and BPRS definitions and retrieval methods. In fact, our proposed methods can be easily extended to the case of an uncertain query object, which will be discussed in Section 6. Figure 3(a) illustrates a set of customers' preferences to laptops, denoted as $\mathcal{A}$, which are represented by uncertainty regions (note: it is a common practice for customers to specify a price/weight range of laptop that they are interested in rather than a single value). Figure 3(b) shows a set of objects, denoted as $\mathcal{B}$, corresponding to the existing/future laptops in the market. If a company wants to produce a new laptop model q, a BPRS query with query point q can be issued to retrieve those customers in $\mathcal{A}$ that may be potentially interested in this new product among all the existing/future models ($\in \mathcal{B}$) in the market. If there are few customers who are interested in the new model, the company may need to change the specifications of the new model in order to attract more customers.

Since the retrieval of MPRS and BPRS requires examining the entire data set(s) to compute the probabilities according to the definitions listed in Section 3, it is quite inefficient and even infeasible. We propose novel pruning methods, namely *spatial* and *probabilistic pruning*, to reduce the search space. Specifically, the spatial pruning method utilizes the spatial relationship among data objects to conduct the filtering, and the probabilistic one uses the data distribution information of uncertain objects to facilitate the pruning. Most importantly, both pruning techniques can be seamlessly integrated into the MPRS and BPRS query procedures, and the filtering time can be further reduced through pre-computation.

In summary, we make the following contributions in this paper.

1. We formalize novel queries, including *monochromatic* and *bichromatic probabilistic reverse skyline* (MPRS and BPRS, respectively) queries, over uncertain databases.
2. We propose effective spatial and probabilistic pruning methods, which are not trivial, to help reduce the search space for both MPRS and BPRS queries. The proposed pruning methods can be seamlessly integrated into our query procedure and efficiently retrieve the query results.
3. We further improve the efficiency of our query procedures via offline pre-computation.
4. Last but not least, we demonstrate through extensive experiments the effectiveness of our pruning methods as well as the efficiency of MPRS and BPRS query processing under various experimental settings.

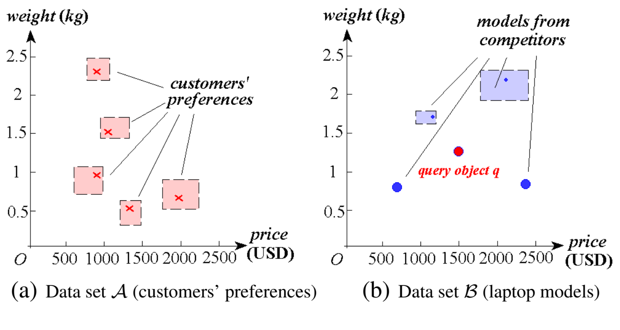


Figure 3: Bichromatic Probabilistic Reverse Skyline Query

The rest of this paper is organized as follows. Section 2 briefly reviews the (reverse) skyline query in "certain" database and query processing over uncertain databases. Section 3 formally defines our problem of probabilistic reverse skyline query in both monochromatic and bichromatic cases. Section 4 and Section 5 illustrate the pruning heuristics and query procedures for MPRS and BPRS, respectively. Section 7 presents the query performance of our proposed approaches in answering MPRS and BPRS queries over the uncertain database. Finally, Section 8 concludes this paper.

2. RELATED WORK

The skyline operator is very important in many real applications such as multi-criteria decision making, market analysis, environmental surveillance and quantitative economics research. In particular, the skyline query can be classified into two categories, static [2] and dynamic (relative) [20, 10, 9]. For the static skyline [2], the attributes of each object in the database are fixed, and skyline points are thus determined once a static database is given. For the dynamic (relative) skyline [20, 10, 9], however, the attributes of each object are dynamically calculated with respect to queries. Specifically, Papadias et al. [20] defined the dynamic attributes of each object by several *dimension functions*. Deng et al. [10] presented the *multi-source skyline query*, in which attributes of each object are defined as the shortest path lengths on road networks from this object to multiple query objects, respectively. Dellis and Seeger [9] considered a dynamic skyline where attributes of each object are dynamically computed as the absolute coordinate differences from a query object for all the dimensions. In this paper, we use the definition of dynamic attributes the same as Dellis and Seeger [9].

Dellis and Seeger [9] proposed a novel query, namely *reverse skyline*, which retrieves those objects in the database whose dynamic skylines contain a given query point. They propose an effective pruning method to reduce the search space with respect to precise points. In contrast, our work in this paper aims to effectively reduce the search space over uncertain data without introducing false dismissals. Moreover, in [9], efficient query procedure and optimization techniques have been proposed to answer the reverse skyline query through the R-tree index. Since the proposed approaches assume the query processing over the precise data points in the "certain" database, they cannot be simply extended to the uncertain scenarios, where real-world data inherently contain uncertainty in many applications such as sensor data monitoring or business planning as mentioned earlier. One straightforward way to answer reverse skyline queries over uncertain data might be to take center of each uncertain data and then apply traditional techniques on centers. However, the query result is clearly inaccurate or even erroneous due to the loss of distribution information of uncertain objects. This motivates our work in this paper to study efficient and accurate query processing of reverse skyline query, considering the unique characteristics of data uncertainty. Besides, Dellis and Seeger [9] only discuss the monochromatic reverse skyline query in a single database. Our work, however, also investigates the bichromatic case in which two uncertain databases are involved. Previous work on the bichromatic query type include the *reverse nearest neighbor* query [27, 13].

In literature [11, 7, 8, 28, 1, 16, 6, 4, 24, 18, 26, 17, 21, 3, 5, 19, 22, 25, 32], uncertain query processing is very important in many applications, due to the wide existence of uncertainty in real-world data. Many techniques have been designed specific for query types in the context of uncertain databases, including *range query* [7, 8, 28, 3], *nearest neighbor query* [6, 7, 17, 5], *skyline query* [21], and *similarity join* [16, 19]. Pei et al. [21] defined the probabilistic skyline query where each object contains discrete random instances. Effective pruning rules have been proposed to facilitate the search process. Moreover, in the probabilistic database with possible worlds semantics [14], *top-k query* [23, 26, 15] has recently been studied. To the best of our knowledge, this is the first work to study the probabilistic reverse skyline query on uncertain data, considering both monochromatic and bichromatic cases.

3. PROBLEM DEFINITION

In this section, we formally define the *monochromatic* and *bichromatic probabilistic reverse skyline* (MPRS and BPRS, respectively) over uncertain databases. Given a query object q, we say object u *dynamically dominates* object p with respect to q (denoted as $u \prec_q p$), if it holds that: 1) $|u_i - q_i| \leq |p_i - q_i|$, for all dimensions $1 \leq i \leq d$, and 2) there exists at least one dimension j, such that $|u_j - q_j| < |p_j - q_j|$, where u_i and p_i are the i-th coordinates of objects u and p, respectively. Based on this dominance definition, the *reverse skyline* [9] is given as follows.

DEFINITION 3.1. (*Reverse Skyline [9]*) *Given a database $\mathcal{D}$ and a query object q, object $u \in \mathcal{D}$ is a reverse skyline of q, if there does not exist any object $p \in \mathcal{D} \backslash \{u\}$ such that $p \prec_u q$.*

Given an uncertain database $\mathcal{D}$, a data object $u \in \mathcal{D}$ is represented by an *uncertainty region* $UR(u)$ where u can locate anywhere with any probabilistic distribution (u cannot appear outside $UR(u)$). In this paper, we model each uncertainty region as of hyperrectangular shape [7, 21]. Following the convention [7, 6, 21], we assume uncertain objects in the database are independent of each other, and coordinates of each object are also independent. An MPRS query over uncertain data can be defined as follows.

DEFINITION 3.2. (*Monochromatic Probabilistic Reverse Skyline, MPRS*) *Given a d-dimensional uncertain database $\mathcal{D}$, a query object q, and a probability threshold $\alpha \in (0, 1]$, a monochromatic probabilistic reverse skyline (MPRS) query retrieves those objects $u \in \mathcal{D}$ such that u is a reverse skyline point of q with probability $P_{MPRS}(u)$ greater than or equal to α, that is,*

$$\begin{aligned} & P_{MPRS}(u) \\ = & \int_{u \in UR(u)} \left(Pr\{u\} \cdot \prod_{p' \in \mathcal{D} \backslash \{u\}} (1 - Pr\{p' \prec_u q\}) \right) do \geq \alpha, \end{aligned} \tag{1}$$

where $Pr\{p' \prec_u q\} = \prod_{i=1}^{d} Pr\{|p'_i - u_i| \leq |q_i - u_i|\}$.

Inequality (1) calculates the expected probability $P_{MPRS}(u)$ that object u is a reverse skyline of q, and checks whether $P_{MPRS}(u)$ is greater than or equal to a probability threshold α. If the answer is yes, then object u is a qualified MPRS point; otherwise, it can be discarded. In order to obtain $P_{MPRS}(u)$, for each possible position of u in $UR(u)$, we need to compute the probability that q is the dynamic skyline of u, that is, the probability that q is not dynamically dominated by other objects $p' \in \mathcal{D} \backslash \{u\}$ (with respect to u). Due to the assumption of object independence, this probability can be obtained by $\prod_{p' \in \mathcal{D} \backslash \{u\}} (1 - Pr\{p' \prec_u q\}$. Similarly, due to the independence among coordinates, $Pr\{p' \prec_u q\} = \prod_{i=1}^{d} Pr\{|p'_i - u_i| \leq |q_i - u_i|\}$.

Similarly, a BPRS query over uncertain databases is given below.

DEFINITION 3.3. (*Bichromatic Probabilistic Reverse Skyline, BPRS*) *Given two d-dimensional uncertain databases $\mathcal{A}$ and $\mathcal{B}$, a query object q, and a probability threshold $\alpha \in (0, 1]$, a bichromatic probabilistic reverse skyline (BPRS) query obtains those objects $u \in \mathcal{A}$ such that u is a reverse skyline point of q in $\mathcal{B}$ with probability $P_{BPRS}(u)$ greater than or equal to α, that is,*

$$\begin{aligned} & P_{BPRS}(u) \\ = & \int_{u \in UR(u)} \left(Pr\{u\} \cdot \prod_{p' \in \mathcal{B}} (1 - Pr\{p' \prec_u q\}) \right) do \geq \alpha, \end{aligned} \tag{2}$$

Symbol	Description
$\mathcal{D}$ ($\mathcal{A}$ and $\mathcal{B}$)	the uncertain database(s)
d	the dimensionality of data objects
o (u, p, or q)	the uncertain object
$UR(o)$	the *uncertainty region* of object o
$p \prec_o q$	object p dynamically dominates object q with respect to object o
α	the user-specified probability threshold
N_p	the farthest point in $UR(p)$ from query object q
M_p	the middle point between q and N_p
m_e	the point in node e closest to q

Figure 4: Symbols and Descriptions

where $Pr\{p' \prec_u q\}$ refers to Definition 3.2.

Note that, the BPRS query is more complex than MPRS, since it involves two data sets $\mathcal{A}$ and $\mathcal{B}$ as indicated by Definition 3.3. Thus, the search of BPRS points must be able to explore both data sets efficiently, whose query processing is not trivial.

The existing approach for the reverse skyline search [9] only handles "certain" data in the monochromatic case, and cannot be directly applied to the uncertain scenarios with both MPRS and BPRS. Therefore, the only available method so far is the *linear scan*, which calculates the probability of each object by sequentially scanning all the other objects on disk. However, this method is clearly not efficient, due to the complex computation of Inequalities (1) and (2) as well as the high I/O cost. Motivated by this, in this paper, we propose effective pruning methods to facilitate answering MPRS and BPRS queries efficiently. Moreover, we further improve the query performance via pre-computation techniques.

```
Procedure PRS_Framework {
  Input: an uncertain database, a query object $q$, and a probability threshold $\alpha$
  Output: the answer to the probabilistic reverse skyline query
  (1)  construct a multidimensional index structure                    // indexing phase
  (2)  perform spatial and/or probabilistic pruning over the index // pruning phase
  (3)  refine the retrieved candidates and return the answer set      // refinement phase
}
```

Figure 5: Framework for Probabilistic Reverse Skyline Queries

Figure 5 illustrates a general framework for MPRS/BPRS query processing, which consists of three phases, *indexing*, *pruning*, and *refinement phases*. Specifically, the *indexing phase* builds up a multidimensional index over uncertain database(s). Since our proposed querying methodology is independent of the underlying index, in this paper, we simply use one of the popular indexes, R-tree [12]. In particular, we insert the uncertainty region of each object into an R-tree index, which recursively groups objects with *minimum bounding rectangles* (MBRs). After the construction of the index, the second *pruning phase* aims to prune those objects that are not qualified as query results (i.e. Inequality (1) or (2) does not hold). In this phase, we propose *spatial* and/or *probabilistic pruning* methods to significantly reduce the search space. Finally, for each remaining candidate o that cannot be pruned, the *refinement phase* checks Inequality (1) (or (2)) by computing the actual probability $P_{MPRS}(o)$ (or $P_{BPRS}(o)$), and reports the qualified objects. Figure 4 summarizes the commonly used symbols in this paper.

4. MONOCHROMATIC PROBABILISTIC REVERSE SKYLINE (MPRS)

In this section, we propose the pruning techniques for MPRS and later in Section 5 we discuss how the solutions can be applied to BPRS. Sections 4.1 and 4.2 present two pruning heuristics, *spatial* and *probabilistic pruning*, respectively, followed by a discussion of retrieving MPRS candidates in Section 4.3. Finally, Section 4.4 seamlessly integrates the pruning methods and refinement step into an MPRS query procedure.

4.1 Spatial Pruning

In this subsection, we illustrate the basic pruning heuristics of our MPRS query procedure using a 2D example in Figure 4.1. Specifically, let q be a query point and $p \in \mathcal{D}$ be an MPRS candidate to the top-right of q. We can draw a line segment from q to its farthest point, N_p, in uncertainty region $UR(p)$, and take the middle point M_p between q and N_p. The shaded region with bottom left corner M_p is called *pruning region* PR(q, p). Obviously, any object $o \in \mathcal{D}$ that is fully contained in this region PR(q, p) cannot be the reverse skyline of q, since object p is always dynamically dominating q with respect to o (i.e. $p \prec_o q$, due to $|p_i - o_i| \leq |q_i - o_i|$ for all dimensions $i = 1, 2$). Thus, object o can be safely pruned.

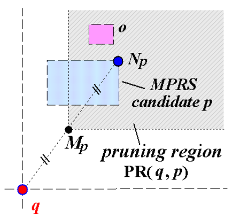


Figure 6: Heuristics of Spatial Pruning

Based on the example above, we give the formal definition of the pruning region PR(q, p):

DEFINITION 4.1. (*Pruning Region, PR(q, p)*) *Assume uncertain object $p \in \mathcal{D}$ has its uncertainty region $UR(p) = (p_1^-, p_1^+; p_2^-, p_2^+; ...; p_d^-, p_d^+)$, where $[p_i^-, p_i^+]$ is the uncertain interval of $UR(p)$ along the i-th dimension ($1 \leq i \leq d$). Given a query point q, if there exists one dimension j such that $p_j^- \leq q_j \leq p_j^+$, then the pruning region, PR(q, p), is empty; otherwise, the i-th dimension of PR(q, p) is given by $[\frac{q_i + p_i^+}{2}, +\infty]$ if $q_i < p_i^-$, or $[-\infty, \frac{q_i + p_i^-}{2}]$ if $q_i > p_i^+$, for $1 \leq i \leq d$.*

Note that, the pruning region defined in Definition 4.1 can be used not only in our MPRS query, but also the BPRS query as well, which will be discussed later. Now, based on Definition 4.1, we propose our monochromatic spatial pruning method as follows.

LEMMA 4.1. (*Monochromatic Spatial Pruning*) *Given a query object q and an MPRS candidate $p \in \mathcal{D}$, any object $o \in \mathcal{D}$ can be safely pruned if it is fully contained in the non-empty pruning region PR(q, p), that is, $UR(o) \subseteq$ PR(q, p).*

Proof. We only need to prove that $p \prec_o q$ under the assumption that o is fully covered by the pruning region PR(q, p). Recall from Definition 4.1, for object p with uncertainty region $UR(p) = (p_1^-, p_1^+; p_2^-, p_2^+; ...; p_d^-, p_d^+)$, the pruning region is defined as $[\frac{q_i + p_i^+}{2}, +\infty]$ if $q_i < p_i^-$; or $[-\infty, \frac{q_i + p_i^-}{2}]$ if $q_i > p_i^+$, where $1 \leq i \leq d$. Since object o is fully contained in the region, we have: 1) $\frac{q_i + p_i^+}{2} \leq o_i$ if $q_i < p_i^-$, or 2) $o_i \leq \frac{q_i + p_i^-}{2}$ if $q_i > p_i^+$ (for $1 \leq i \leq d$). We prove that, in these two cases, we have $p \prec_o q$.

Case 1 (**$q_i < p_i^-$ and $\frac{q_i+p_i^+}{2} \le o_i$**). From the case assumption, it holds that $q_i = \frac{q_i+q_i}{2} < \frac{q_i+p_i^-}{2} \le \frac{q_i+p_i^+}{2} \le o_i$. If $p_i - o_i \ge 0$ (Case 1.1), we have $|p_i - o_i| = p_i - o_i \le p_i^+ - o_i \le o_i - q_i = |o_i - q_i|$ (since $p_i \le p_i^+$ and $\frac{q_i+p_i^+}{2} \le o_i$); otherwise (Case 1.2, i.e., $p_i - o_i < 0$), it holds that $|p_i - o_i| = o_i - p_i \le o_i - q_i = |o_i - q_i|$ (due to $q_i < p_i^- \le p_i$).

Case 2 (**$q_i > p_i^+$ and $o_i \le \frac{q_i+p_i^-}{2}$**). Omitted due to space limit (with proof similar to Case 1).

In summary, in both cases, we can always obtain the inequality $|p_i - o_i| \le |q_i - o_i|$ for all $1 \le i \le d$, indicating that $p \prec_o q$. □

From Lemma 4.1, we can see that the spatial pruning method only discards those objects that definitely cannot be MPRS points. Therefore, this pruning method would not introduce any *false dismissals* (i.e. the actual answers to the query which are however not in the query result).

4.2 Probabilistic Pruning

The spatial pruning method only utilizes the spatial property of uncertain objects, that is, the bounds of uncertainty regions, to reduce the MPRS search space. In case we know the distributions of objects in their uncertainty regions, we should be able to use such information to enhance the pruning ability for our MPRS search. In this subsection, we propose a novel and effective *probabilistic pruning* method, which exactly aims to utilize this distribution information to increase the pruning power.

Recall that, any object o is an MPRS point, if its expected probability $P_{MPRS}(o)$ (in Inequality (1)) to be the reverse skyline of query point q is greater than or equal to $\alpha \in (0, 1]$. The rationale of our probabilistic pruning is to discard those objects with probability $P_{MPRS}(o)$ smaller than or equal to some $\beta \in [0, \alpha)$. In particular, in order to enable such pruning, we introduce the concept of $(1-\beta)$-hyperrectangle for uncertain objects.

DEFINITION 4.2. ($(1-\beta)$*-Hyperrectangle,* $UR_{1-\beta}(\cdot)$) *As illustrated in Figure 7, assume we have an uncertain object o with its uncertainty region $UR(o)$ centered at point C_o. The $(1-\beta)$-hyperrectangle of object o is defined as a smaller hyperrectangle $UR_{1-\beta}(o)$, such that object o locates in $UR_{1-\beta}(o)$ with probability $(1-\beta)$. The complement part of $UR_{1-\beta}(o)$ in $UR(o)$ is denoted as $\overline{UR_{1-\beta}(o)}$, and object o locates in $\overline{UR_{1-\beta}(o)}$ with probability β.*

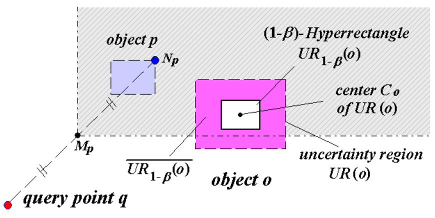


Figure 7: Illustration of $(1-\beta)$-Hyperrectangle

Note that, since we know the distribution of each object o in its uncertainty region $UR(o)$, we are able to pre-compute the $(1-\beta)$-hyperrectangle $UR_{1-\beta}(o)$ offline, for some $\beta \in [0, 1]$.

Next, we provide our monochromatic probabilistic pruning method for an MPRS query.

LEMMA 4.2. (*Monochromatic Probabilistic Pruning*) *Given a query object q, an MPRS candidate $p \in \mathcal{D}$, and a probability threshold α, any object $o \in \mathcal{D}$ can be safely pruned if it holds that the $(1-\beta)$-hyperrectangle $UR_{1-\beta}(o)$ is fully contained in PR(q, p), that is, $UR_{1-\beta}(o) \subseteq$ PR(q, p), for $\beta \in [0, \alpha)$.*

Proof. We only need to prove that, as long as $UR_{1-\beta}(o)$ is fully contained in the pruning region PR(q, p), the expected probability $P_{MPRS}(o)$ (in Inequality (1)) that object o is a reverse skyline of q is smaller than or equal to $\beta \in [0, \alpha)$. Recall from Inequality (1):

$$P_{MPRS}(o) = \int_{o \in UR(o)} \left(Pr\{o\} \cdot \prod_{p' \in \mathcal{D} \setminus \{o\}} (1 - Pr\{p' \prec_o q\}) \right) do.$$

Since the probability $Pr\{p' \prec_o q\}$ in $P_{MPRS}(o)$ is always within $[0, 1]$, we have:

$$P_{MPRS}(o) \le \int_{o \in UR(o)} (Pr\{o\} \cdot (1 - Pr\{p \prec_o q\}))\, do, \tag{3}$$

where $p \in \mathcal{D} \setminus \{o\}$.

Since uncertain object o can locate in either $UR_{1-\beta}(o)$ or $\overline{UR_{1-\beta}(o)}$ and any probability is non-negative (i.e. $Pr\{\cdot\} \ge 0$), we can rewrite $Pr\{p \prec_o q\}$ as:

$$\begin{aligned} Pr\{p \prec_o q\} &= Pr\{p \prec_o q | o \in UR_{1-\beta}(o)\} \cdot Pr\{o \in UR_{1-\beta}(o)\} \\ &\quad + Pr\{p \prec_o q | o \in \overline{UR_{1-\beta}(o)}\} \cdot Pr\{o \in \overline{UR_{1-\beta}(o)}\} \\ &\ge Pr\{p \prec_o q | o \in UR_{1-\beta}(o)\} \cdot (1-\beta) + 0 \cdot \beta \\ &= Pr\{p \prec_o q | o \in UR_{1-\beta}(o)\} \cdot (1-\beta) \end{aligned} \tag{4}$$

Note that, due to the assumption of the lemma that $UR_{1-\beta}(o)$ is fully covered by PR(q, p), we have $Pr\{p \prec_o q | o \in UR_{1-\beta}(o)\} = 1$ (with a proof similar to that of Lemma 4.1), which can be substituted into Inequality (4) and result in:

$$Pr\{p \prec_o q\} \ge 1 - \beta. \tag{5}$$

Therefore, by combining Inequalities (5) and (3), we have:

$$\begin{aligned} P_{MPRS}(o) &\le \int_{o \in UR(o)} (Pr\{o\} \cdot (1 - (1-\beta)))\, do \\ &= \beta \cdot \int_{o \in UR(o)} Pr\{o\} do = \beta. \end{aligned} \tag{6}$$

Hence, object o has the expected probability $P_{MPRS}(o)$ to be reverse skyline of q not greater than β (i.e. smaller than α), and thus it can be safely pruned. □

As shown in the example of Figure 7, the pruning region PR(q, p) with respect to q and p (shaded area) fully contains the $(1-\beta)$-hyperrectangle $UR_{1-\beta}(o)$ of object o. Thus, o can be safely pruned by our monochromatic probabilistic pruning method in Lemma 4.2. In contrast, the monochromatic spatial pruning cannot prune o since the uncertainty region of o is only partially covered by PR(q, p). According to Lemma 4.2, we know that no false dismissals are introduced after the monochromatic probabilistic pruning.

4.3 Retrieval of MPRS Candidates

Up to now, we have considered monochromatic spatial/probabilistic pruning methods with respect to one MPRS candidate in the database $\mathcal{D}$. In fact, any object p in $\mathcal{D}$ can be an MPRS candidate. However, their resulting pruning regions can be either large or small (i.e. with high or low pruning ability). Thus, we want to find those MPRS candidates with pruning regions as large as possible.

Figure 8 illustrates a query point q and two objects p and p'. We observe that the two pruning regions, PR(q, p) and PR(q, p'), have the containment relationship, that is, PR$(q, p) \supset$ PR(q, p'). In this case, the pruning region PR(q, p') is redundant, since any object o that is fully contained in PR(q, p') can be also pruned by PR(q, p). Thus, it is desired that we obtain PR(q, p), rather than PR(q, p'), during the retrieval process of MPRS candidates. We have the following lemma.

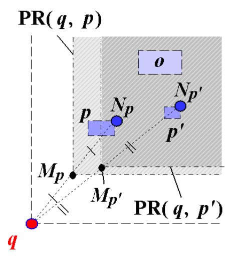


Figure 8: Containment Relationship Between Pruning Regions

Lemma 4.3. *For any two objects p and p' in $\mathcal{D}$ that have non-empty pruning regions, assume $N_p(n_1, n_2, ..., n_d)$ and $N_{p'}(n'_1, n'_2, ..., n'_d)$ are two points in $UR(p)$ and $UR(p')$, respectively, farthest from query point q, where $n_i \in \{p_i^-, p_i^+\}$ and $n'_i \in \{p_i'^-, p_i'^+\}$ for $1 \leq i \leq d$. If it holds that: 1) $(n_i - q_i) \cdot (n'_i - q_i) > 0$ for all i, 2) $|n_i - q_i| \leq |n'_i - q_i|$ for all i, and 3) there exists at least one dimension j, such that $|n_j - q_j| < |n'_j - q_j|$, then we have $PR(q,p) \supset PR(q,p')$.*

Proof. Since p (or p') has non-empty pruning region, we have either $q_i < p_i^-$ or $q_i > p_i^+$ ($q_i < p_i'^-$ or $q_i > p_i'^+$), for any $1 \leq i \leq d$. According to Condition 1) that $(p_i - q_i) \cdot (p'_i - q_i) > 0$ for all i, we consider two cases:

Case 1 ($n_i > q_i$ **and** $n'_i > q_i$ **for some** i**).** From the case assumption and the fact that PR(q,p) and PR(q,p') are not empty, we have $p_i^- > q_i$ and $p_i'^- > q_i$. Based on Definition 4.1, the pruning region PR(q, p) along the i-th dimension is $[\frac{q_i+p_i^+}{2}, +\infty]$ and PR(q,p') along the i-th dimension is $[\frac{q_i+p_i'^+}{2}, +\infty]$. Moreover, from Conditions 2) and the assumption of Case 1, we obtain $n_i - q_i = |n_i - q_i| \leq |n'_i - q_i| = n'_i - q_i$ (i.e. $n_i \leq n'_i$). Since $n_i = p_i^+$ and $n'_i = p_i'^+$, it holds that $[\frac{q_i+p_i^+}{2}, +\infty] \supseteq [\frac{q_i+p_i'^+}{2}, +\infty]$.

Case 2 ($n_i < q_i$ **and** $n'_i < q_i$ **for some** i**).** Omitted due to space limit (with proof similar to Case 1).

According to the two cases above, we have PR(q, p) $\supseteq$ PR(q, p'), since PR(q, p) contains PR(q, p') in all dimensions. Due to Condition 3), the set equality does not hold for the j-th dimension (i.e. when $i = j$) in the proof above. Hence, the lemma holds that PR(q, p) $\supset$ PR(q, p'). □

For brevity, if points N_p and $N_{p'}$ satisfy the three conditions in Lemma 4.3, we say N_p *globally dominates* $N_{p'}$. If a point N_p is not globally dominated by any other points (e.g. $N_{p'}$), we call it *global skyline point*.

Lemma 4.3 indicates that as long as point N_p globally dominates point $N_{p'}$ with respect to q, the resulting pruning region PR(q, p) would contain PR(q, p'). Let M_p and $M_{p'}$ be the middle points from q to N_p and $N_{p'}$ (i.e., corners of PR(q,p) and PR(q,p')), respectively. From Lemma 4.3, we have an immediate corollary below.

Corollary 4.1. *If M_p globally dominates $M_{p'}$, then we have $PR(q,p) \supset PR(q,p')$.* □

Therefore, in order to achieve high pruning ability (i.e. large pruning region), we need to access and obtain those objects p whose M_p are global skyline points (i.e. having pruning regions not fully covered by others).

4.4 MPRS Query Processing

In this subsection, we discuss the detailed MPRS query processing which progressively retrieves those objects p having M_p as global skyline points (i.e. with pruning regions not fully covered by others), and, meanwhile, utilizes the pruning regions to prune other objects via the spatial and/or probabilistic pruning method(s) as mentioned earlier. Specifically, we index all the uncertain objects in the database $\mathcal{D}$ with an R-tree $\mathcal{I}$ [12], on which the MPRS query is processed. In order to facilitate an efficient MPRS search, we need to prune intermediate nodes of the R-tree index as early as possible to avoid costly accessing the MBRs/objects under these nodes (in terms of computation and I/O cost). Therefore, we first design a method to prune intermediate entries of $\mathcal{I}$ in the following lemma.

Lemma 4.4. (*Pruning Intermediate Entries for MPRS*) *Assume we have an intermediate entry e in the R-tree index $\mathcal{I}$ constructed over uncertain database $\mathcal{D}$, a query object q, and an MPRS candidate p. Let m_e be a point in e that is the closest to query point q, where $q_i < e_i^-$ or $q_i > e_i^+$ for all $1 \leq i \leq d$. If $PR(q, m_e) \subseteq PR(q,p)$, then e can be safely pruned.*

Proof. Derived from Corollary 4.1. Omitted due to space limit. □

MPRS Query Procedure. Figure 9 illustrates the MPRS query procedure, MPRS_Processing, in detail, which retrieves the qualified MPRS objects by traversing the R-tree index $\mathcal{I}$ in a *best-first* manner. Specifically, we maintain a minimum heap $\mathcal{H}$ with entries in the form (e, key) (line 1), where e is an MBR node of index $\mathcal{I}$ and key is defined by function $key(q,e) = min_{x \in e}\{\sum_{i=1}^{d} |q_i - x_i|/2\}$ [20]. The function $key(q,e)$ defines the L_1-norm distance from query point q to m_e in e (as given in Lemma 4.4). Intuitively, small $key(q,e)$ may result in large pruning region PR(q,p) for some point $p \in e$. We also initialize two empty sets, S_{cand} and S_{rfn} (line 2), where S_{cand} is used to store MPRS candidates, and S_{rfn} stores those pruned objects/nodes that, however, may help refine MPRS candidates in the refinement phase.

First, we insert the root $root(\mathcal{I})$ of R-tree into heap $\mathcal{H}$ (line 3). Every time we pop out an entry (e, key) from $\mathcal{H}$ with the minimum key value (line 5). Then, according to Lemma 4.4, we verify whether or not PR(q, m_e) is fully contained in PR(q,p') for some object p' in the candidate set S_{cand}. If it is true, then all the points in e would have their pruning regions fully covered by PR(q,p'), and thus e can be safely pruned by (line 6); otherwise, we process e as follows.

If e is a leaf node, we need to verify each object p in entry e (lines 7-8). In particular, if object p can be spatially pruned by some candidate p' from the candidate set S_{cand} and, moreover, M_p is globally dominated by $M_{p'}$ for some $p' \in S_{cand}$, we add it to the refinement set S_{rfn} (lines 9-11), since p is neither an MPRS candidate nor an object that has PR(q,p) not fully covered by others. Note that, we do not discard p immediately but add it to S_{rfn}, since it may be useful for helping refine MPRS candidates during the refinement phase. In case p is not an MPRS candidate but has its PR(q,p) not covered by other pruning regions, we add it to S_{cand} and mark it as false alarm (line 12, note: we can still use PR(q,p) to perform the pruning); in case p is an MPRS candidate, we simply add it to S_{cand} (line 13). Note that, if p is added to S_{cand} and moreover we know the distribution of object $p \in UR(p)$, we can load its corresponding $(1-\beta)$-hyperrectangle $UR_{1-\beta}(p)$, for a largest pre-computed β value in $[0, \alpha)$. In addition, we use object p to prune candidates $p' \in S_{cand}$ (line 14) by either spatial or probabilistic pruning (determined by whether or not we know the position distribution of p'), as given in Lemma 4.1 or Lemma 4.2, respectively. Here, we only mark the pruned candidates in S_{cand} as false alarms without removing them (since their corresponding pruning regions can still help prune more objects/nodes). When e is an intermediate node, we scan each entry e_i in e and check whether or not e_i

can be pruned (lines 16-20). Specifically, by Lemma 4.4, if it holds that PR$(q, m_{e_i}) \subseteq$ PR(q, p') for some $p' \in S_{cand}$, we insert it into S_{rfn} (lines 17-18); otherwise, we add an entry $(e_i, key(q, e_i))$ to heap $\mathcal{H}$ (lines 19-20).

```
Procedure MPRS_Processing {
  Input: R-tree $\mathcal{I}$ constructed over $\mathcal{D}$, a user-specified probability threshold $\alpha$
  Output: the MPRS query result
  (1)  initialize a min-heap $\mathcal{H}$ accepting entries in the form $(e, key)$
  (2)  $S_{cand} = \phi, S_{rfn} = \phi, rlt = \phi$;
  (3)  insert $(root(\mathcal{I}), 0)$ into heap $\mathcal{H}$
  (4)  while $\mathcal{H}$ is not empty
  (5)     $(e, key)$ = de-heap $\mathcal{H}$
  (6)     if PR$(q, m_e) \subseteq$ PR$(q, p')$ for some $p' \in S_{cand}$
                    // $m_e$ is the nearest point in entry e from q, Lemma 4.4
  (7)        if e is a leaf node
  (8)           for each uncertain object $p \in e$
  (9)              if p is fully contained in PR$(q, p')$ for some $p' \in S_{cand}$
  (10)                if $M_p$ is globally dominated by $M_{p'}$ for some $p' \in S_{cand}$
  (11)                   $S_{rfn} = S_{rfn} \cup \{p\}$
  (12)                else $S_{cand} = S_{cand} \cup \{p\}$ and mark p as false alarms
  (13)             else $S_{cand} = S_{cand} \cup \{p\}$        // Lemma 4.1, spatial pruning
  (14)             mark candidate $p' \in S_{cand}$ as false alarms if it can be pruned by p
                    // Lemmas 4.1, 4.2 and 4.3, spatial/probabilistic pruning
  (15)       else      // intermediate node
  (16)          for each entry $e_i$ in e
  (17)             if PR$(q, m_{e_i}) \subseteq$ PR$(q, p')$ $m_{e_i}$ for some $p' \in S_{cand}$
  (18)                $S_{rfn} = S_{rfn} \cup \{e_i\}$                // Lemma 4.4
  (19)             else
  (20)                insert $(e_i, key(q, e_i))$ into heap $\mathcal{H}$
  (21)    $rlt$ = Refinement$(q, S_{cand}, S_{rfn}, \alpha)$;        // refinement phase
  (22) return $rlt$
}
```

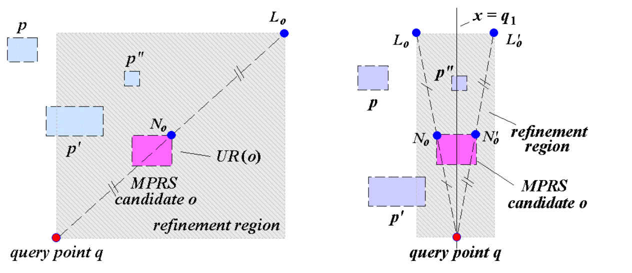
Figure 9: MPRS Query Processing

Final Refinement. The iterations of traversing the R-tree repeats until heap $\mathcal{H}$ is empty (line 4). After that, we obtain an MPRS candidate set S_{cand}, in which we filter out false alarms (line 21). That is, for each candidate $p \in S_{cand}$, we need to access objects in $\mathcal{D}$ and compute the actual probability $P_{MPRS}(p)$ using numeric method [7, 6]. Note that, since objects in $\mathcal{D}$ are either in S_{cand} or in the refinement set S_{rfn}, we can access S_{rfn} to avoid visiting tree nodes twice. If it holds that $P_{MPRS}(p) \geq \alpha$, p is a qualified MPRS point; otherwise, o is discarded. Finally, the actual answers to the MPRS query, rlt, can be returned (line 22).

Note that, from Inequality (1), the calculation of $P_{MPRS}(o)$ requires scanning all the uncertain objects in the database $\mathcal{D}$, which is quite inefficient. In order to reduce this cost, we define a *refinement region*, such that any uncertain object p that falls outside this region can be excluded from the calculation of Inequality (1). In brief, there are two types of the refinement region, depending on whether or not q_i is within $[o_i^-, o_i^+]$ for some $1 \leq i \leq d$. Figure 10 illustrates these two cases in a 2D example, where object o is an MPRS candidate and q is a query point. Figure 10(a) presents the first case, where $q_1 \notin [o_1^-, o_1^+]$ and $q_2 \notin [o_2^-, o_2^+]$ along the horizontal and vertical axes, respectively. Let N_o be the middle point of line segment qL_0, where N_o is the farthest point in $UR(o)$ from q. The hyperrectangle that has q and L_o as diagonal corners is defined as our refinement region. Furthermore, in Figure 10(b), if there exists a dimension, for example, the horizontal axis, such that $q_1 \in [o_1^-, o_1^+]$, we divide the space into two halfplanes by line $x = q_1$, and obtain the refinement region in each halfplane similar to the first case. The final refinement region is the union of these two small regions (i.e. shaded area shown in Figure 10(b)).

With regard to the refinement region, we have the lemma below.

LEMMA 4.5. *Any uncertain object p that is fully outside the refinement region would not affect the calculation of Inequality (1).*



(a) Case 1 (b) Case 2

Figure 10: Illustration of Refinement Region

Proof Sketch. From Figure 10, we can see that if object p is outside the refinement region, then q would dynamically dominate p with respect to o, that is, $q \prec_o p$. In other words, we have $Pr\{p \prec_o q\} = 0$. Thus, 1 ($= 1 - Pr\{p \prec_o q\}$) will be multiplied into the term $\prod_{p' \in \mathcal{D} \setminus \{o\}} (1 - Pr\{p' \prec_o q\})$ in Inequality (1), for object p, which however does not have any effect on the result. □

In the examples of Figure 10(a) or 10(b), object p can be safely pruned, since it is entirely outside the shaded region. Only those objects that intersect with the refinement region (e.g. objects p' and p'') are necessary to participate in calculating the probability in Inequality (1). Thus, in the refinement phase, instead of scanning the entire database $\mathcal{D}$, we only need to retrieve those objects that overlap with the refinement region of each candidate, through the index, and compute the expected probability to obtain the actual answer to the BPRS query.

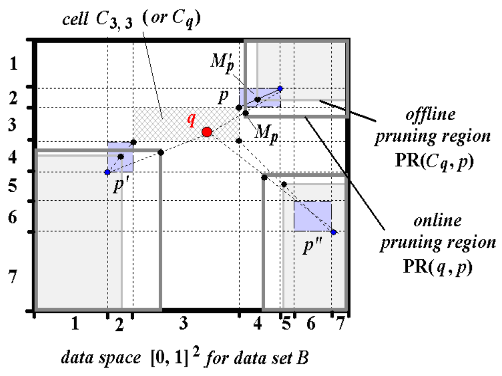
Analysis. Based on Corollary 4.1, in order to achieve the maximum pruning power, our query procedure progressively retrieves the objects whose pruning regions are not full covered by others (or equivalently, the global skyline points of M_p for $p \in \mathcal{D}$). Since M_p is the corner of the pruning region PR(q, p) with respect to q, we can finally obtain the maximum pruning region. Similar to BBS algorithm [20], procedure MPRS_Processing finds global skyline points M_p of objects p in the database, where the global dominance is used. In other words, we progressively compute the candidate set and meanwhile use the pruning regions from S_{cand} to prune the MPRS search space. We have the following lemma.

LEMMA 4.6. *The number of node accesses over $\mathcal{I}$ invoked by line 10 of procedure* MPRS_Processing *is I/O optimal to obtain all the global skyline points from the data set $\mathcal{D}$.*

Proof Sketch. Our procedure MPRS_Processing to compute the global skyline points is equivalent to applying BBS algorithm [20] to obtain skyline points within space partitions, which are obtained by dividing the space at q_i for each dimension i. Since each node in $\mathcal{I}$ is accessed at most once in our procedure and BBS algorithm is I/O optimal, procedure MPRS_Processing is I/O optimal. □

4.5 Enhanced Query Processing with Pre-Computation

In this subsection, we further propose a novel and more efficient approach, aiming to reduce the cost of MPRS query processing via offline pre-computation. Recall that, in procedure MPRS_Processing, for each newly incoming MPRS query with query point q, we need to re-calculate the pruning regions from scratch. However, if two query points are very close, they are very likely to share the same or similar pruning regions. Observing this, we can improve the MPRS query efficiency by offline pre-computing some pruning regions as well as MPRS candidates, such that the cost of searching them through the index (in terms of computation and I/O) can be significantly saved.



cell $C_{3,3}$ (or C_q)
1
2
3
4
5
6
7
M'_p
p
q
M_p
p'
p''
offline pruning region PR(C_q, p)
online pruning region PR(q, p)
1
2
3
4
5
6
7
data space $[0,1]^2$ for data set B

Figure 11: Heuristics of Enhanced Query Processing

We illustrate the rationale of our enhanced query processing with an example in Figure 11. Without loss of generality, we only show three relevant objects p, p', p'' from database $\mathcal{D}$. For the i-th dimension, the uncertain intervals of p, p', and p'' are given by $[p_i^-, p_i^+]$, $[p_i^{'-}, p_i^{'+}]$, and $[p_i^{''-}, p_i^{''+}]$, respectively. Based on the endpoints of these intervals along each dimension, we can partition the data space into a grid that contains a number of cells. As shown in the example, the horizontal dimension is split by endpoints $p_1^{'-}$, $p_1^{'+}$, p_1^-, p_1^+, $p_1^{''-}$, and $p_1^{''+}$ from left to right; and similarly, the vertical dimension is split by endpoints p_2^+ p_2^-, $p_2^{'+}$, $p_2^{'-}$, $p_2^{''+}$, and $p_2^{''-}$ from top to bottom. Denote $C_{i,j}$ as a cell that is on the i-th column and j-th row of the grid.

The framework for our MPRS query processing with offline pre-computation is as follows. First, for each cell $C_{i,j}$, we pre-calculate the *offline pruning regions* with respect to cell $C_{i,j}$ (assuming $C_{i,j}$ is the query region). Moreover, based on these pruning regions, we obtain a superset of MPRS candidates for any query point q' in $C_{i,j}$. Whenever an MPRS query is issued with query point q, we directly access a cell C_q where q falls into, retrieve its corresponding offline MPRS candidates together with objects p that were used to define the offline pruning regions, and compute the *online pruning regions* (i.e. PR(q,p)) to filter out MPRS candidates. Finally, procedure Refinement is invoked to refine candidates by accessing the index and calculating the actual probability $P_{MPRS}(o)$ as discussed in Section 4.4.

Below, we define the offline pruning regions using the example of Figure 11. Consider the cell $C_{3,3}$ (denoted as C_q) and an object p. Obviously, for each possible position q' of q in the cell C_q, we can obtain a pruning region PR(q', p) with respect to q' and p. Regardless of q's actual position, the offline pruning region is given by the intersection of regions PR(q', p) for all $q' \in C_q$, that is, $\bigcap_{\forall q' \in C_q}$ PR(q', p). In Figure 11, point M'_p is exactly the bottom-left corner of such intersection. Formally, we have:

DEFINITION 4.3. (*Offline Pruning Region, PR(C_q, p)*) *Given a cell C_q and an object p, if there exists one dimension j such that $(C_{qj}^-, C_{qj}^+) \bigcap (p_j^-, p_j^+) \neq \phi$, then the offline pruning region, PR(C_q, p), is empty, where (C_{qj}^-, C_{qj}^+) is the side of cell C_q along the j-th dimension. Otherwise, if $C_{qi}^+ \leq p_i^-$ holds, then PR(C_q, p) along the i-th dimension is given by $[\frac{C_{qi}^+ + p_i^+}{2}, +\infty]$; if $p_i^+ \leq C_{qi}^-$ holds, then PR(C_q, p) along the i-th dimension is $[-\infty, \frac{p_i^- + C_{qi}^-}{2}]$.*

Therefore, our offline pre-computation can be accomplished as follows. For each cell C_q in the data space, we invoke the query procedure MPRS_Processing to obtain a candidate set S_{cand}, with the new pruning regions given in Definition 4.3. Here, the candidate set S_{cand} also includes those objects that are marked as false alarms (in lines 12 and 14 of procedure MPRS_Processing), however, they still contribute their pruning regions. In our example, objects p, p', and p'' determine three offline pruning regions, respectively, for cell C_q as shown in Figure 11 (shaded area).

Interestingly, regarding the offline pruning regions, we have the following lemma.

LEMMA 4.7. *Assume we have an offline pruning region PR$(C_q$, $p)$, which is not fully covered by non-empty PR(C_q, p') of any other objects $p' \in \mathcal{D}$. Then, it holds that, the online pruning region, PR(q, p) (with respect to $q \in C_q$), cannot be fully covered by any other PR(q, p') for $p' \in \mathcal{B}$ either; vice versa.*

Proof Sketch. Derived from the lemma assumption and the property of our grid partitioning. Omitted due to space limit. □

From Lemma 4.7, we can see that if we retrieve all the offline pruning regions PR(C_q, p) that cannot be fully covered by other offline ones, then their online versions would achieve large pruning area (since they are not fully contained in any other regions). Moreover, from Figure 11, we can see that the online pruning region always contains its offline version, that is, PR$(q, p) \supseteq$ PR(C_q, p). Thus, the resulting offline MPRS candidate set would be a superset of the online one with respect to query point q (i.e. no false dismissals introduced).

During our enhanced query processing via pre-computation, we directly obtain those offline MPRS candidates and perform fast filtering by online pruning regions. Therefore, in contrast to procedure MPRS_Processing, the cost of searching MPRS candidates through the index can be significantly saved, in terms of both computation and I/O.

Note that, our enhanced approach trades the space for efficiency. The space cost of the pre-computation is proportional to $|\mathcal{D}|^d$. For each cell C_q, we store uncertainty regions of objects from S_{cand}. In order to reduce the space cost, coarser cells can be used, which however may result in smaller pruning regions and thus more MPRS candidates to be refined. Moreover, compression methods can be also applied to save the space cost of storing MPRS candidates, trading the decompression cost for space efficiency. For example, *Hilbert curve* can compress a sequence of coordinates to a single value, which is also able to be reconstructed back to data series (i.e. decompression). Due to space limit, we would not discuss details in this paper.

5. BICHROMATIC PROBABILISTIC REVERSE SKYLINE (BPRS)

In this section, we first illustrate the detailed differences in computing MPRS and BPRS. Then, we discuss how to modify the proposed pruning techniques for MPRS to efficiently answer BPRS queries.

5.1 Computation Differences Between MPRS and BPRS

Different from the MPRS query, the BPRS problem involves two data sets. Therefore, the BPRS query processing is more complex than MPRS.

According to the BPRS definition, Figure 12 illustrates one straightforward approach to answer the BPRS query, where a BPRS query is issued with query point q over two data sets $\mathcal{A}$ and $\mathcal{B}$. Each time the retrieval process considers one possible query result $o \in \mathcal{A}$ individually consisting of two steps. First, we conceptually insert o into the data set $\mathcal{B}$ and obtain an updated data set $\mathcal{B}' = (\mathcal{B} \cup \{o\})$. As a second step, we find the probabilistic reverse skylines of q in the updated data set $\mathcal{B}'$. In other words, we issue an MPRS query with query point q over $\mathcal{B}'$, using the pruning methods that we mentioned in Section 4. In particular, we can obtain the pruning regions in the data set $\mathcal{B}'$, and check whether or not object $o \in \mathcal{A}$ is fully contained in any of these regions. If the answer is yes, then o can be safely pruned; otherwise, we have to refine o by computing its probability $P_{BPRS}(o)$ (in Inequality (2)) to be the probabilistic reverse skyline in $\mathcal{B}'$, and output o as the BPRS result if $P_{BPRS}(o) \geq \alpha$ holds.

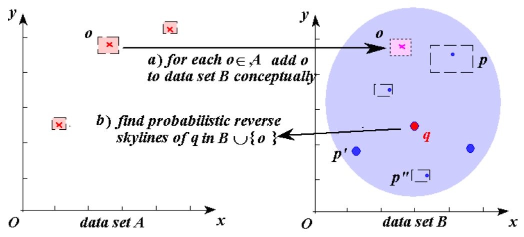

y
o
a) for each o∈A add o to data set B conceptually
b) find probabilistic reverse skylines of q in B ∪{o}
O
data set A
x
y
o
p
q
p'
p''
O
data set B
x

Figure 12: Differences Between MPRS and BPRS Queries

Therefore, in contrast to the MPRS query processing which calculates the pruning regions over one data set and prunes objects within this data set itself, BPRS requires computing the pruning regions over data set $\mathcal{B}$ and utilizing them to prune objects from the second data set $\mathcal{A}$ at the same time. Although the straightforward approach mentioned above can produce the correct answer to the BPRS query, its time complexity is quite high (i.e. proportional to the sizes of both data sets), and thus not scalable to large data sizes.

In the sequel, we assume that data sets $\mathcal{A}$ and $\mathcal{B}$ are indexed by two R-trees [12], $\mathcal{I}_\mathcal{A}$ and $\mathcal{I}_\mathcal{B}$, respectively. The goal of this work is to provide effective pruning methods to reduce the BPRS search space and efficiently retrieve BPRS objects through R-tree indexes. Section 5.2 presents the pruning heuristics of the BPRS query processing. Section 5.3 demonstrates details of our BPRS query processing as well as the offline pre-computation technique to speed up the query procedure.

5.2 Pruning Heuristics

In this subsection, we illustrate the pruning heuristics of the BPRS query using a 2D example shown in Figure 13. Given a query object q and an uncertain object $p \in \mathcal{B}$, we define the pruning region PR(q, p) as given in Definition 4.1. In particular, we obtain the farthest point N_p in $UR(p)$ from q, and take the middle point M_p between q and N_p. PR(q, p) is the shaded area having M_p as its bottom-left corner.

By utilizing pruning regions, we have bichromatic spatial and probabilistic pruning methods below, similar to the MPRS case.

LEMMA 5.1. (*Bichromatic Spatial Pruning*) *Given a query point q and two uncertain databases $\mathcal{A}$ and $\mathcal{B}$, any object $o \in \mathcal{A}$ can be safely pruned, if it is fully contained in a non-empty pruning region PR(q, p) with respect to object $p \in \mathcal{B}$.*

LEMMA 5.2. (*Bichromatic Probabilistic Pruning*) *Given a query point q, two uncertain databases $\mathcal{A}$ and $\mathcal{B}$, and a probability threshold α, any object $o \in \mathcal{A}$ can be safely pruned, if its $(1-\beta)$-hyperrectangle, $UR_{1-\beta}(o)$, is fully contained in a non-empty pruning region PR(q, p) for object $p \in \mathcal{B}$, where $\beta \in [0, \alpha)$.*

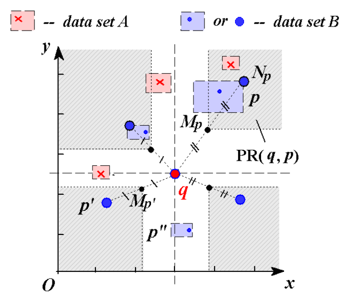


Figure 13: Pruning Heuristics of BPRS Query Processing

5.3 BPRS Query Processing

In this subsection, we integrate the bichromatic spatial and/or probabilistic pruning methods into our BPRS query procedure. Specifically, given a query point q and two data sets, $\mathcal{A}$ and $\mathcal{B}$, our BPRS query procedure progressively obtains pruning regions from data set $\mathcal{B}$, and meanwhile uses them to reduce the search space in the data set $\mathcal{A}$. Specifically, in order to achieve high pruning ability, we find all the pruning regions PR(q, p) of objects $p \in \mathcal{B}$ that are not fully covered by PR(q, p') of other objects $p' \in \mathcal{B}$, that is, those objects p in $\mathcal{B}$ whose M_p are globally skyline points, as indicated by Corollary 4.1.

BPRS Query Procedure. Figure 14 illustrates the BPRS query procedure, BPRS_Processing, in detail. In contrast to the MPRS query over a single index, BPRS query processing needs to traverse two indexes, $\mathcal{I}_\mathcal{A}$ and $\mathcal{I}_\mathcal{B}$ for databases $\mathcal{A}$ and $\mathcal{B}$, respectively, and handle two types of pruning: 1) pruning objects/nodes in $\mathcal{I}_\mathcal{A}$ (i.e. reducing BPRS candidate size), and 2) pruning objects/nodes in $\mathcal{I}_\mathcal{B}$ (i.e. filtering out the pruning region candidates). As a result, we have two pruning rules:

1. Objects $p \in \mathcal{B}$ can prune objects/nodes e_A in $\mathcal{I}_\mathcal{A}$, if e_A (or $UR_{1-\beta}(e_A)$ in case e_A is an object for some $\beta \in [0, \alpha)$) is fully contained in its pruning regions PR(q, p); and
2. Object $p \in \mathcal{B}$ can prune object p' in $\mathcal{B}$, if it holds that M_p globally dominates $M_{p'}$; object $p \in \mathcal{B}$ can prune node e_B in $\mathcal{I}_\mathcal{B}$, if it holds that PR$(q, p) \supseteq$ PR(q, p') for any point $p' \in e_B$, in other words, PR$(q, p) \supseteq$ PR(q, m_{e_B}) (since PR(q, m_{e_B}) has the largest possible area for any $p' \in e_B$), where m_{e_B} is the closest point in e_B from q.

Therefore, in order to retrieve the BPRS query results, we access $\mathcal{I}_\mathcal{A}$ and $\mathcal{I}_\mathcal{B}$ in a way similar to MPRS, while performing the pruning only if one of pruning rules above is satisfied. In particular, procedure BPRS_Processing (Figure 14) maintains a minimum heap for each R-tree (i.e. $\mathcal{H}_\mathcal{A}$ and $\mathcal{H}_\mathcal{B}$ for $\mathcal{I}_\mathcal{A}$ and $\mathcal{I}_\mathcal{B}$, respectively) with entries in the form (e, key) (line 1), where e is an MBR node and key is defined by function $key_A(q, e)$ or $key_B(q, e)$. Specifically, when e is a node from $\mathcal{I}_\mathcal{A}$, we have $key_A(q, e) = \sum_{i=1}^{d} min\{|q_i - e_i^-|, |q_i - e_i^+|\}$, which is the minimum L_1-norm distance from q to e; when e is a node from $\mathcal{I}_\mathcal{B}$, we have $key_B(q, e) = \sum_{i=1}^{d} min\{|q_i - e_i^-|/2, |q_i - e_i^+|/2\}$, which is the minimum possible L_1-norm distance from q to the corner of the pruning region PR(q, m_e), where m_e is a point in e closest to q. In addition, we keep three sets S_A, S_B, and S_{rfn}, which are initialized to empty (line 2). Set S_A is used to store BPRS query results; S_B stores objects $p \in \mathcal{B}$ with global skyline points M_p; and S_{rfn}

contains MBRs/objects from $\mathcal{I}_\mathcal{B}$ that are pruned, however, might be useful for refining BPRS candidates during the refinement step.

Procedure BPRS_Processing accesses nodes in both $\mathcal{I}_\mathcal{A}$ and $\mathcal{I}_\mathcal{B}$ in ascending order of their keys. First, the roots of $\mathcal{I}_\mathcal{A}$ and $\mathcal{I}_\mathcal{B}$ are inserted into heaps $\mathcal{H}_\mathcal{A}$ and $\mathcal{H}_\mathcal{B}$, respectively (line 3). Then, each time we pop out a heap entry from one of the heaps (selecting the one with minimum key) (line 6 or 9). If the current popped entry e is from heap $\mathcal{H}_\mathcal{A}$, we invoke procedure Node_Handler_A (lines 6-7), which checks whether e can be pruned by any pruning regions PR(q, p) (as given by the first pruning rule); otherwise, we invoke procedure Node_Handler_B (lines 8-10), which pops out top entry (e_B, key_B) from heap $\mathcal{H}_\mathcal{B}$ and applies the second pruning rule given above. The iteration stops when one of heaps is empty (line 4). After that, if heap $\mathcal{H}_\mathcal{A}$ is not empty (i.e. $\mathcal{H}_\mathcal{B}$ is empty), we invoke procedure Node_Handler_A to handle each remaining entry e_A in $\mathcal{H}_\mathcal{A}$ (line 11). In case heap $\mathcal{H}_\mathcal{B}$ is not empty, we simply add all the heap entries in $\mathcal{H}_\mathcal{B}$ to the refinement set S_{rfn} (line 12), which may be used for refining BPRS candidates. Finally, procedure Refinement is called to refine the BPRS candidates stored in S_A (line 13). In particular, for each candidate o, we access those objects/nodes in S_{rfn} and S_B that intersect with its refinement region, and calculate the expected probability. If this probability is greater than or equal to α, then o is output as the query results (line 14); otherwise, it is discarded. Detailed BPRS query procedure is described in Figures 14, 15, and 16.

```
Procedure BPRS_Processing {
  Input: two R-trees $\mathcal{I}_\mathcal{A}$ and $\mathcal{I}_\mathcal{B}$ for data sets $\mathcal{A}$ and $\mathcal{B}$, respectively, a query point $q$,
         and a user-specified probability threshold $\alpha$
  Output: the BPRS query result $S_A$
  (1)  initialize two min-heaps $\mathcal{H}_\mathcal{A}$ and $\mathcal{H}_\mathcal{B}$ accepting entries in the form $(e, key)$
  (2)  $S_A = S_B = S_{rfn} = \phi$;
  (3)  insert $(root(\mathcal{I}_\mathcal{A}), key_A(q, root(\mathcal{I}_\mathcal{A})))$ and $(root(\mathcal{I}_\mathcal{B}), key_B(q, root(\mathcal{I}_\mathcal{B})))$
         into heaps $\mathcal{H}_\mathcal{A}$ and $\mathcal{H}_\mathcal{B}$, respectively
  (4)  while both $\mathcal{H}_\mathcal{A}$ and $\mathcal{H}_\mathcal{B}$ are not empty
  (5)    if $top(\mathcal{H}_\mathcal{A}).key_A < top(\mathcal{H}_\mathcal{B}).key_B$
  (6)      $(e_A, key_A)$ = de-heap($\mathcal{H}_\mathcal{A}$)      // access index $\mathcal{I}_\mathcal{A}$
  (7)      Node_Handler_A $(q, e_A, S_A, S_B, \mathcal{H}_\mathcal{A}, \alpha)$      // pruning rule 1
  (8)    else
  (9)      $(e_B, key_B)$ = de-heap($\mathcal{H}_\mathcal{B}$)      // access index $\mathcal{I}_\mathcal{B}$
  (10)     Node_Handler_B $(q, e_B, S_B, S_{rfn}, \mathcal{H}_\mathcal{B})$      // pruning rule 2
  (11) if $\mathcal{H}_\mathcal{A}$ is not empty, invoke procedure Node_Handler_A for each $e_A$ in $\mathcal{H}_\mathcal{A}$
  (12) if $\mathcal{H}_\mathcal{B}$ is not empty, add all the remaining entries to $S_{rfn}$
  (13) Refinement $(S_A, S_B, S_{rfn}, \alpha)$
  (14) return $S_A$;
}
```

Figure 14: BPRS Query Processing

```
Procedure Node_Handler_A {
  Input: a query point $q$, entry $e_A$ from $\mathcal{I}_\mathcal{A}$, sets $S_A$ and $S_B$, heap $\mathcal{H}_\mathcal{A}$, and a
         user-specified probability threshold $\alpha$
  (1)  if $e_A$ is a leaf node
  (2)    for each object $o \in e_A$
  (3)      if $UR(o)$ (or $UR_{1-\beta}(o)$) is not fully contained in pruning regions
             obtained from $S_B$ // bichromatic spatial (or probabilistic) pruning
  (4)        $S_A = S_A \cup \{o\}$
  (5)  else  // intermediate node of $\mathcal{I}_\mathcal{A}$
  (6)    for each entry $e_A^j \in e_A$
  (7)      if $e_A^j$ is not fully contained in pruning regions obtained from $S_B$
                                                      // bichromatic spatial pruning
  (8)        insert $(e_A^j, key_A(q, e_A^j))$ into heap $\mathcal{H}_\mathcal{A}$
}
```

Figure 15: Handling Nodes from Data Set $\mathcal{A}$

Analysis. While ignoring lines 6-7 of procedure BPRS_Processing, our query procedure is performed over data set $\mathcal{B}$, analogue to BBS algorithm [20] that retrieves skyline points (in our case, the corner points of pruning regions). Thus, similar to Lemma 4.6, we have:

```
Procedure Node_Handler_B {
  Input: a query point $q$, entry $e_B$ from $\mathcal{I}_\mathcal{B}$, sets $S_B$ and $S_{rfn}$, and heap $\mathcal{H}_\mathcal{B}$
  (1)  if PR$(q, m_{e_B}) \subseteq$ PR$(q, p')$ for some $p' \in S_B$
                                   // $m_{e_B}$ is the nearest point in entry $e_B$ from $q$
  (2)    $S_{rfn} = S_{rfn} \cup \{p\}$ and return;
  (3)  if $e_B$ is a leaf node
  (4)    for each object $p \in e_B$
  (5)      if $M_p$ is globally dominated by $M_{p'}$ for any $p' \in S_B$ // Lemma 4.3
  (6)        $S_{rfn} = S_{rfn} \cup \{p\}$
  (7)      else
  (8)        $S_B = S_B \cup \{p\}$
  (9)  else  // intermediate node of $\mathcal{I}_\mathcal{B}$
  (10)   for each entry $e_B^j \in e_B$
  (11)     if PR$(q, m_{e_B^j}) \subseteq$ PR$(q, p')$ for some $p' \in S_B$
                                   // $m_{e_B^j}$ is the nearest point in entry $e_B^j$ from $q$
  (12)       $S_{rfn} = S_{rfn} \cup \{e_B^j\}$
  (13)     else
  (14)       insert $(e_B^j, key_B(q, e_B^j))$ into heap $\mathcal{H}_\mathcal{B}$
}
```

Figure 16: Handling Nodes from Data Set $\mathcal{B}$

LEMMA 5.3. *The number of node accesses over $\mathcal{I}_\mathcal{B}$ invoked by line 10 of procedure* BPRS_Processing *is I/O optimal to obtain S_B such that, for any $p \in \mathcal{S}_\mathcal{B}$, the pruning region PR$(q, p)$ are not fully covered by others.* □

Furthermore, with regard to the candidate set S_A, we have the following lemma.

LEMMA 5.4. *In procedure* Node_Handler_A *(line 3), if $UR(o)$ is not fully contained in pruning regions, then those remaining MBRs/objects e_B in heap $\mathcal{H}_\mathcal{B}$ cannot prune object o either, that is, the pruning region PR(q, e_B) of e_B will not fully cover $UR(o)$.*

Proof. By contradiction. Assume there exists a subsequent MBR /object e_B from $\mathcal{H}_\mathcal{B}$, such that $UR(o)$ is fully contained in its pruning region PR(q, e_B). However, as guaranteed by line 5 of procedure BPRS_Processing, for any subsequent MBR/object $e_B \in \mathcal{H}_\mathcal{B}$ that we access, we have $key_A(q, o) < key_B(q, e_B)$, which is equivalent to $\sum_{i=1}^{d} min\{|q_i - o_i^-|, |q_i - o_i^+|\} < \sum_{i=1}^{d} min\{|q_i - e_{Bi}^-|/2, |q_i - e_{Bi}^+|/2\}$. This indicates that, there must exist an integer $1 \le j \le d$, such that $min\{|q_j - o_j^-|, |q_j - o_j^+|\} < min\{|q_j - e_{Bj}^-|/2, |q_j - e_{Bj}^+|/2\}$ holds. Thus, $UR(o)$ cannot be fully covered by PR(q, e_B) at least on the j-th dimension, which is contrary to our initial assumption. Hence, the lemma holds. □

From Lemma 5.4, we can infer that if we only apply the spatial pruning (in case the position distribution of objects is unknown), every object added to S_A (in line 4 of procedure Node_Handler_A) must be a BPRS candidate, which cannot be pruned by any pruning regions from $\mathcal{B}$.

Enhanced Query Processing with Pre-computation. Similar to the MPRS case, we can improve the query efficiency by some offline pre-computations. Recall that, our proposed approach in previous subsections first compute the pruning regions online, with respect to query point q, via index $\mathcal{I}_\mathcal{B}$, then use them to prune objects/nodes in index $\mathcal{I}_\mathcal{A}$, and finally refine the retrieved candidates through index $\mathcal{I}_\mathcal{B}$ again. Although our query procedure is I/O efficient for index $\mathcal{I}_\mathcal{B}$ (as guaranteed by Lemma 5.3) upon the arrival of each new BPRS query, we have to re-calculate the pruning regions from scratch, which requires much computation and I/O costs. Motivated by this, we pre-compute the BPRS candidates as well as offline pruning regions (as defined in Definition 4.3), and speed up the BPRS query efficiency.

In particular, we construct a grid over data set $\mathcal{B}$ in the same way as that mentioned in Section 4.5. Then, for each cell C_q, issue the query procedure BPRS_Processing to obtain set $\mathcal{S}_\mathcal{B}$ in which objects are used to define offline pruning regions (as defined in Definition 4.3), together with BPRS candidates in S_A. S_A and S_B are sequentially stored on disk with respect to C_q. If a BPRS query with query point q arrives, we directly access cell C_q that q is in, which points to its corresponding sets S_A and S_B. Next, we refine the candidate set S_A by using online pruning regions from q and S_B. Finally, procedure Refinement is invoked to refine the remaining candidates through index $\mathcal{I}_\mathcal{B}$ (mentioned in Section 4.4).

6. DISCUSSIONS

Our proposed approaches for MPRS and BPRS can be easily extended to the case where the query object q is also uncertain locating anywhere within a hyperrectangle $UR(q)$. In this case, with respect to $UR(q)$ and an object o, the pruning region PR$(UR(q), o)$ can be defined similar to the offline pruning region (replacing C_q with $UR(q)$) in Definition 4.3. Thus, we can invoke the MPRS or BPRS query procedure via the pruning region PR$(UR(q), o)$ for each candidate o, and obtain an MPRS/BPRS candidate set. Finally, for each candidate, we retrieve objects that overlap with its refinement region and calculate its expected probability to be MPRS/BPRS answer. If the expectation is not smaller than α, it is reported as an MPRS/BPRS result; otherwise, it is discarded.

Furthermore, in the discrete case of MPRS or BPRS, suppose each uncertain object o contains $l(o)$ random samples, $s_1(o)$, $s_2(o)$, ..., $s_{l(o)}(o)$ (assuming samples appear with equal probability which can be easily extended to that with weighted probabilities). Given a query point q, the probability $P_{MPRS}(o)$ (or $P_{BPRS}(o)$) in Definition 3.2 (Definition 3.3) can be rewritten by replacing integration with summation on $s_j(o) \in o$, $Pr\{o\}$ with $1/l(o)$, and $Pr\{p' \prec_o q\}$ with $\frac{|s_i(p')|s_i(p') \prec_{s_j(o)} q|}{l(p')}$. Note that, our previously discussed pruning methods can be still used for the discrete scenario, since we consider every possible position of objects in their uncertainty regions. In particular, we bound all the samples of each object with a hyperrectangle and insert it into an index (e.g. R-tree). Further, we offline pre-compute $(1-\beta)$-hyperrectangle $UR_{1-\beta}(o)$ from discrete samples (such that $(1-\beta)$ of $l(o)$ samples appear in $UR_{1-\beta}(o)$), which is used for probabilistic pruning during MPRS /BPRS query processing discussed in previous sections.

7. EXPERIMENTAL EVALUATION

In this section, we test the efficiency and effectiveness of our proposed pruning methods for both MPRS and BPRS query processing with and without pre-computations. Since real uncertain data sets are not available, in our experiments, we synthetically generate uncertain objects with various parameter values, similar to [6, 8, 28]. In particular, in order to produce an uncertain object o, we first pick up the center C_o of o in a d-dimensional data space with domain $[0, 1000]$ in each dimension. Then, we select a radius r_o within $[r_{min}, r_{max}]$, which indicates the maximum deviation of object position from center C_o. Finally, we randomly generated a hyperrectangle $UR(o)$ that is tightly bounded by the sphere centered at C_o and with radius r_o. We consider two types of location distributions for C_o, *Uniform* and *Skew* (skewness of *Zipf* distribution is set to 0.8); and moreover two types of radius distributions for r_o, *Uniform* and *Gaussian* (with mean $(r_{min}+r_{max})/2$ and variance $(r_{max}-r_{min})/5$). Within the uncertainty region of

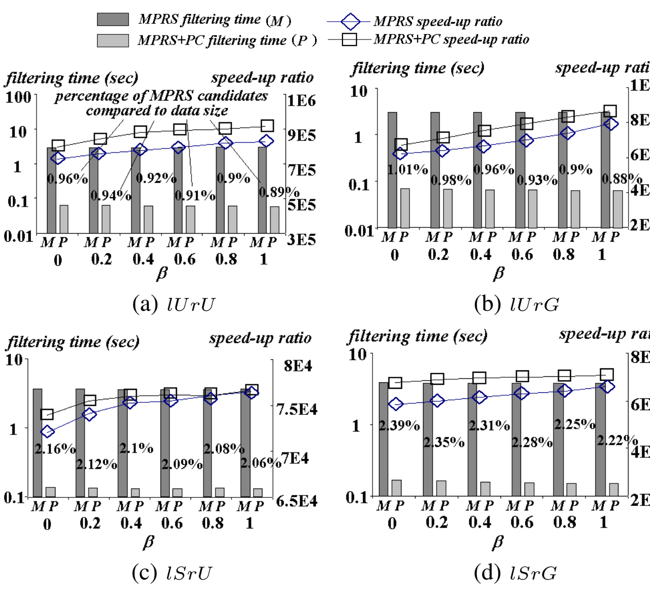


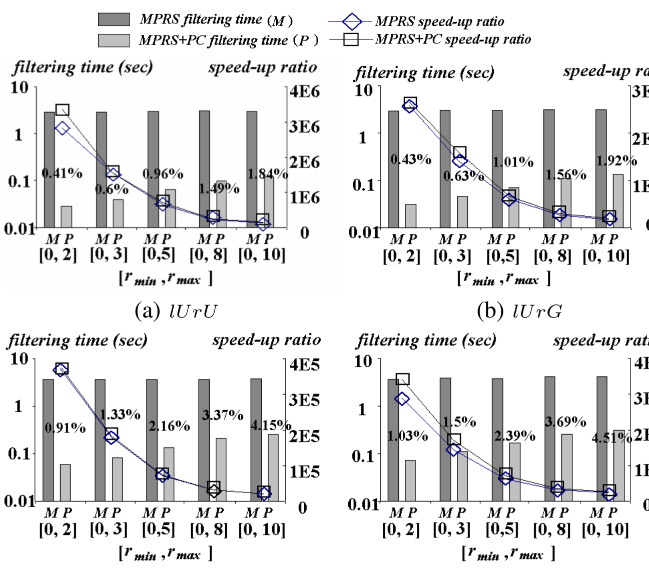
Figure 17: Performance vs. β (MPRS)

each object, we generate random samples following uniform distribution. Note that, due to space limit, we do not present the experimental results for data sets with other parameter settings (e.g. skewness, mean, and variance) or sample distributions, which have similar query performance. For brevity, we denote lU (lS) as the data set with center location C_o of Uniform (Skew) distribution, whereas rU (rG) as the one with radius r_o of Uniform (Gaussian) distribution within $[r_{min}, r_{max}]$. Therefore, we have four types of data sets (with parameter combinations), denoted as $lUrU$, $lUrG$, $lSrU$, and $lSrG$. We index the uncertainty regions of objects with R-tree [12], where the page size is set to 4K.

In order to evaluate the performance of MPRS and BPRS queries, we randomly generate 50 query points following the same distribution as (center locations of) objects in data sets $\mathcal{D}$ and $\mathcal{B}$, respectively. For either MPRS or BPRS, we compare two of our proposed approaches, with and without pre-computations, in terms of the *filtering time* during the pruning phase. In particular, the filtering time consists of two parts, CPU time and I/O cost, where each page access is incorporated by penalizing $10ms$ [29, 30]. Moreover, to the best of our knowledge, this is the first work to study MPRS and BPRS query processing in uncertain databases. The only available method is the *linear scan*, which calculates the probability $P_{MPRS}(o)$ ($P_{BPRS}(o)$) of each object o in $\mathcal{D}$ ($\mathcal{A}$) by sequentially scanning other objects (samples) on the disk for MPRS (BPRS) queries. Furthermore, we also show the *speed-up ratio* of our approaches, which is defined as the total running time of the linear scan method divided by that of our approaches. All our experiments are conducted on a Pentium IV 3.2GHz PC with 1G memory. The reported results are the average of 50 queries.

7.1 The Performance of MPRS Queries

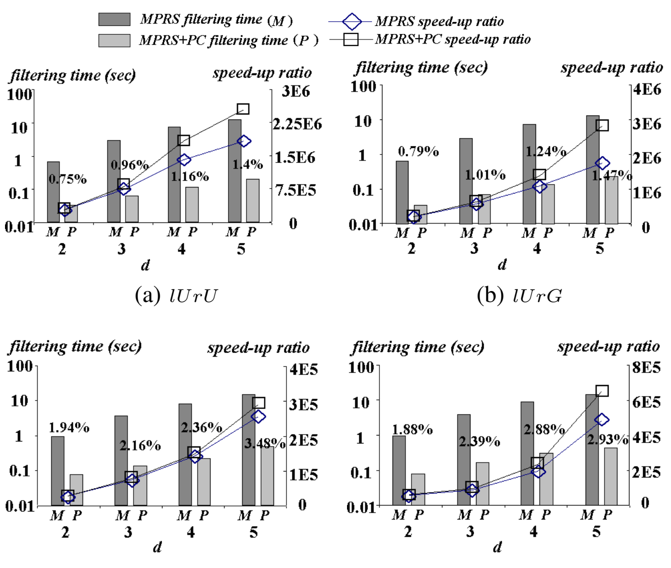
For brevity, we denote the MPRS query processing without precomputation as MPRS (or M in figures), and the one with precomputation as MPRS+PC (or P in figures). Specifically, during the pruning phase, MPRS traverses the R-tree index to obtain MPRS candidates, whereas MPRS+PC retrieves a superset of pre-computed MPRS candidates (and objects that define offline pruning regions as well) from disk and perform the filtering by online pruning regions with respect to query point. After the pruning phase, both



(a) $lUrU$ (b) $lUrG$

(c) $lSrU$ (d) $lSrG$

Figure 18: Performance vs. $[r_{min}, r_{max}]$ (MPRS)



(a) $lUrU$ (b) $lUrG$

(c) $lSrU$ (d) $lSrG$

Figure 19: Performance vs. Dimensionality d (MPRS)

MPRS and MPRS+PC access the index to obtain objects that intersect with the refinement regions and refine MPRS candidates by calculating the actual probability (discussed in Section 4.4).

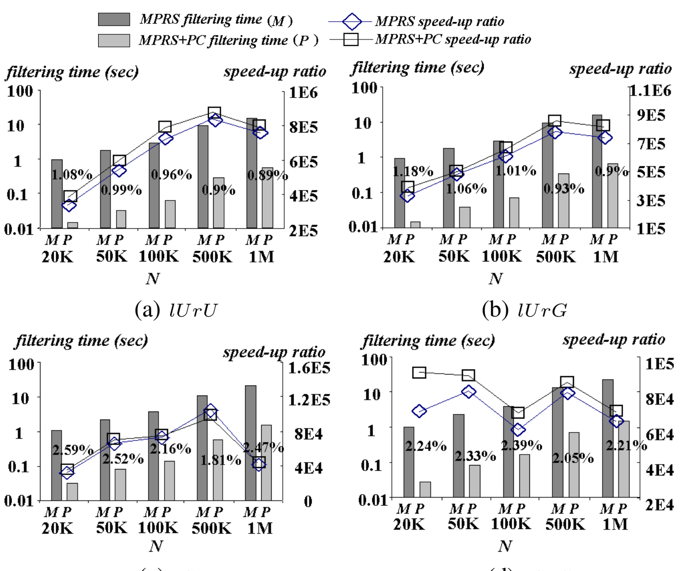
Effect of β on MPRS. In the first set of experiments, we test the effect of β on the MPRS query performance, over four types of data sets $lUrU$, $lUrG$, $lSrU$, and $lSrG$. Figure 17 illustrates the experimental results by setting $\alpha = 1$ in order to test the performance in case the maximum pre-computed β value is 0, 0.2, 0.4, 0.6, 0.8, or 1, where the data size $N = 100K$, the dimensionality $d = 3$, the range $[r_{min}, r_{max}]$ of radius r_o is $[0, 5]$, and 100 samples are used for each object in the probability integration. In particular, the columns in figures indicate the filter time of MPRS and MPRS+PC, whereas the curves represent the speed-up ratio of our approach compared with the linear scan. The numbers on columns show the percentage of MPRS candidates compared with the total data size (i.e. N), after the pruning phase. From figures, we can see that our approaches outperform the linear scan by about 4-5 orders of magnitude. When the largest used $\beta \in [0, \alpha)$ increases, the speed-up ratio becomes higher over all the four data sets, with fewer MPRS candidates, which confirms the efficiency and effectiveness of our monochromatic spatial/probabilistic pruning methods. In general, data sets with skew center locations (i.e. lS) have higher filtering cost than those with uniform locations (i.e. lU), since more MPRS candidates are required to be refined. All the subsequent experimental results show similar trends. Moreover, the filtering time of MPRS+PC is by more than one order of magnitude smaller than that of MPRS, which confirms the efficiency of our MPRS+PC method.

In the sequel, we test the robustness of our MPRS query procedure over data sets with various parameter settings (e.g. N, d, $[r_{min}, r_{max}]$, and so on). In particular, since when β is set to 0, the MPRS query performance is the worst for all possible β (i.e. with the lowest pruning ability and speed-up ratio). Thus, in the subsequent experiments, we would test the worst-case performance by setting β to 0, which can be used for either case where the position distributions of objects are known or unknown.

Effect of $[r_{min}, r_{max}]$ on MPRS. Figure 18 illustrates the worst-case query performance of MPRS and MPRS+PC over the four types of data sets, by varying the range $[r_{min}, r_{max}]$ of r_o from $[0, 2]$ to $[0, 10]$, where $\beta = 0$, the data size $N = 100K$, the dimensionality $d = 3$, and 100 samples are used for each object in the probability integration. From the results, we can see that the filtering time increases when the range $[r_{min}, r_{max}]$ becomes large. This is mainly because more MPRS candidates are required to be refined when the uncertainty regions of objects become large. Correspondingly, the speed-up ratio decreases with the large range, which, however, remains high (i.e. by 4-5 orders of magnitude better than linear scan). Moreover, MPRS+PC has lower filtering time and higher speed-up ratio than MPRS, for all the tested data sets.

Effect of Dimensionality d on MPRS. Figure 19 presents the worst-case query performance, by setting the dimensionality d of data sets to $2, 3, 4$, and 5, where $\beta = 0$, the data size $N = 100K$, $[r_{min}, r_{max}] = [0, 5]$, and 100 samples are used for each object in the probability integration. The results show that our proposed approaches outperform the linear scan method by 4-5 orders of magnitude. Furthermore, we find that both filtering time and the speed-up ratio increase when d is varied from 2 to 5. The filtering cost goes up for higher dimension, since the query performance of R-tree index usually degrades with the increasing dimensionality [31]. On the other hand, the high speed-up ratio in higher dimension indicates the nice scalability of our approaches with respect to dimensionality. Moreover, our MPRS+PC always shows better performance than MPRS in terms of both filtering time and speed-up ratio (especially when the dimensionality is high).

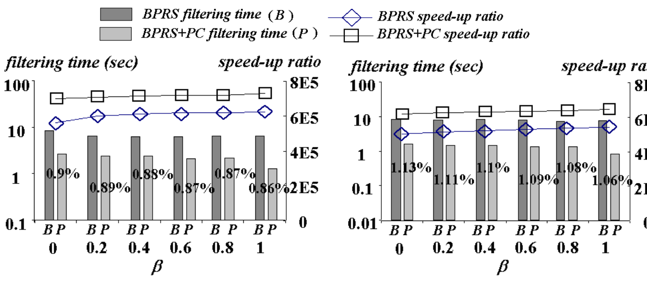
Effect of Data Size N on MPRS. In this set of experiments, we demonstrate the scalability of our approaches, MPRS and MPRS+PC, with respect to the data size N. In particular, we vary N from $20K$ to $1M$ in Figure 20, where $\beta = 0$, the dimensionality $d = 3$, $[r_{min}, r_{max}] = [0, 5]$, and 100 samples are used for each object in the probability integration. The experimental results indicate that the filtering time slightly increases when the data size N goes up while the speed-up ratio remains high (i.e. by about 4-6 orders of magnitude better than the linear scan), which confirms the nice scalability of our approaches on data size.



(a) $lUrU$ (b) $lUrG$

(c) $lSrU$ (d) $lSrG$

Figure 20: **Performance vs. Data Size** N **(MPRS)**



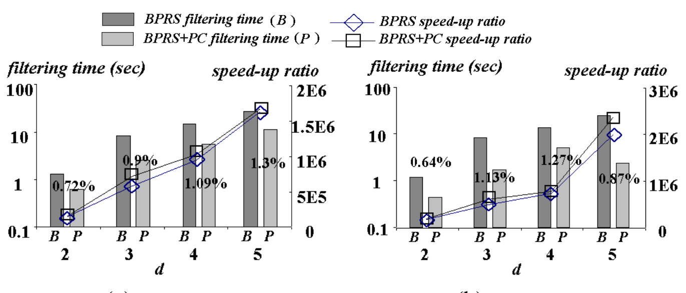
(a) $lUrG - lUrG$ (b) $lSrG - lUrG$

Figure 21: **Performance vs.** β **(BPRS)**

7.2 The Performance of BPRS Queries

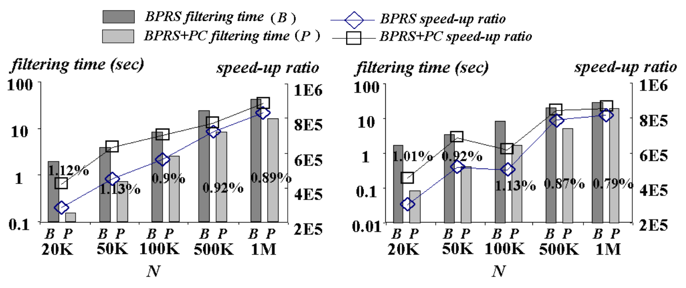
In this subsection, we study the query performance of BPRS query processing, in terms of the filtering time and speed-up ratio compared with the linear scan method. In particular, we denote the BPRS query processing without pre-computation as BPRS (or B in figures), and the one with pre-computation as BPRS+PC (or P in figures). In particular, during the pruning phase, BPRS traverses both indexes of data sets $\mathcal{A}$ and $\mathcal{B}$ and obtains BPRS candidates, whereas BPRS+PC retrieves offline pre-computed candidates and objects that define offline pruning regions from the disk. The refinement step of both approaches accesses the index of $\mathcal{B}$ to obtain those objects involved in probability calculations. Due to space limit, here we only present the experimental results over two pairs of data sets, $lUrG - lUrG$ and $lSrG - lUrG$ (in the form $\mathcal{A} - \mathcal{B}$), with the same data size. The trends of query performance for other data sets or data size combinations are similar.

Effect of β **on BPRS.** Figure 21 illustrates the query performance of BPRS and BPRS+PC, by setting $\alpha = 1$ and varying the largest pre-computed β from 0 to 1 (similar to the MPRS case), where $N = 100K$ (for both data sets), $d = 3$, $[r_{min}, r_{max}] = [0, 5]$, and 100 samples are used for each object in the probability integration. From figures, we can see that when β increases, BPRS and BPRS+PC require less filtering time and have higher speed-up



(a) $lUrG - lUrG$ (b) $lSrG - lUrG$

Figure 22: **Performance vs. Dimensionality** d **(BPRS)**



(a) $lUrG - lUrG$ (b) $lSrG - lUrG$

Figure 23: **Performance vs. Data Size** N **(BPRS)**

ratio, compared with the linear scan method, which indicates the effectiveness of our bichromatic spatial/probabilistic pruning methods. The percentage of BPRS candidates (numbers on columns) in the data set $\mathcal{A}$ is decreasing for large β. Moreover, BPRS+PC performs much better than BPRS over both data set pairs.

Effect of Dimensionality d **on BPRS.** Figure 22 illustrates the experimental results by varying dimensionality d from 2 to 5, where $\beta = 0$, $N = 100K$ (for both data sets), $[r_{min}, r_{max}] = [0, 5]$, and 100 samples are used for each object in the probability integration. The query performance is similar to the MPRS case. Note that, for data set pair $lSrG - lUrG$, the filtering time of MPRS+PC for $d = 5$ is smaller than that of MPRS+PC for $d = 4$, since fewer BPRS candidates (i.e. 0.87%) are needed to load from disk (compared with 1.27%) and perform the filtering. In general, our approaches perform much better than the linear scan.

Effect of Data Size N **on BPRS.** Finally, we evaluate the worst-case BPRS performance of our approaches by letting $\beta = 0$. Specifically, we study the effect of data size on our BPRS query processing in Figure 23, where $N = 20K, 50K, 100K, 500K, 1M$ (for both data sets), $d = 3$, $[r_{min}, r_{max}] = [0, 5]$, and 100 samples are used for each object in the probability integration. The experimental results show that, when the data size increases, the required filtering time smoothly goes up. Moreover, the speed-up ratio remains high (i.e. by 5-6 orders of magnitude compared with the linear scan method), which implies a good scalability of our approaches. Note that, for the data set pair $lSrG - lUrG$ in Figure 23(b), the speed-up ratio for $N = 100K$ is lower than that for $N = 50K$, since more candidates (i.e. 1.13% compared with 0.92%) are needed to be refined. In general, BPRS+PC consistently outperforms BPRS in terms of the speed-up ratio.

We also did experiments with other parameters (e.g. $[r_{min}, r_{max}]$) whose results are similar to the MPRS case and omitted due to space limit. In summary, our proposed approaches can efficiently answer both MPRS and BPRS queries, in terms of the filtering time and speed-up ratio compared with the linear scan method, under various types of data sets and with different settings.

8. CONCLUSIONS

Reverse skyline query has many important applications, which retrieves data objects whose dynamic skyline points contain a given query point. Due to the inherent uncertainty in many real-world data, the query processing techniques that are designed for precise data cannot be directly applied to handle uncertain data. In this paper, we focus on the reverse skyline query processing over uncertain data, namely *probabilistic reverse skyline*, in both monochromatic and bichromatic cases (i.e. MPRS and BPRS, respectively). In particular, we propose effective pruning methods to reduce the search space of MPRS and BPRS queries, and seamlessly integrate them into efficient query procedures. Moreover, the enhanced query processing methods are also proposed via pre-computation techniques. Extensive experiments have demonstrated the efficiency and effectiveness of our proposed approaches in answering MPRS and BPRS queries, under various experimental settings.

Acknowledgments

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

Repeatability Assessment Result

All the results in this paper were verified by the SIGMOD repeatability committee.

9. REFERENCES

[1] C. Böhm, A. Pryakhin, and M. Schubert. The Gauss-tree: efficient object identification in databases of probabilistic feature vectors. In *ICDE*, page 9, 2006.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[3] J. Chen and R. Cheng. Efficient evaluation of imprecise location-dependent queries. In *ICDE*, pages 586–595, 2007.

[4] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[5] R. Cheng and J. Chen. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, 2008.

[6] R. Cheng, D. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. In *TKDE*, volume 16, pages 1112–1127, 2004.

[7] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

[8] 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.

[9] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.

[10] K. Deng, X. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *ICDE*, pages 796–805, 2007.

[11] 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.

[12] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[13] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, pages 806–815, 2007.

[14] C. Koch, D. Olteanu, L. Antova, and T. Jansen. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.

[15] G. Kollios, K. Yi, F. Li, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *ICDE*, 2008.

[16] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *DASFAA*, 2006.

[17] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, 2007.

[18] V. Ljosa and A. K. Singh. APLA: indexing arbitrary probability distributions. In *ICDE*, pages 247–258, 2007.

[19] V. Ljosa and A. K. Singh. Top-k spatial joins of probabilistic objects. In *ICDE*, 2008.

[20] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[21] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.

[22] S. Prabhakar, C. Mayfield, R. Cheng, S. Singh, R. Shah, J. Neville, and S. Hambrusch. Database support for pdf attributes. In *ICDE*, 2008.

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

[24] A. D. Sarma, O. B., A. Y. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, page 7, 2006.

[25] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database support for pdf attributes. In *ICDE*, 2008.

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

[27] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*, pages 99–108, 2001.

[28] 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.

[29] Y. Tao, D. Papadias, and X. Lian. Reverse kNN search in arbitrary dimensionality. In *VLDB*, pages 744–755, 2004.

[30] Y. Tao, D. Papadias, X. Lian, and X. Xiao. Multidimensional reverse kNN search. In *VLDBJ*, 2005.

[31] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *PODS*, pages 161–171, 1996.

[32] P. S. Yu and C. Aggarwal. On high dimensional indexing of uncertain data. In *ICDE*, 2008.