Computing All Skyline Probabilities for Uncertain Data

Mikhail J. Atallah
CS, Purdue University
West Lafayette, IN 47907-2107, USA
mja@cs.purdue.edu

Yinian Qi
CS, Purdue University
West Lafayette, IN 47907-2107, USA
yqi@cs.purdue.edu

ABSTRACT

Skyline computation is widely used in multi-criteria decision making. As research in uncertain databases draws increasing attention, skyline queries with uncertain data have also been studied, e.g. probabilistic skylines. The previous work requires "thresholding" for its efficiency – the efficiency relies on the assumption that points with skyline probabilities below a certain threshold can be ignored. But there are situations where "thresholding" is not desirable – low probability events cannot be ignored when their consequences are significant. In such cases it is necessary to compute skyline probabilities of all data items. We provide the first algorithm for this problem whose worst-case time complexity is sub-quadratic. The techniques we use are interesting in their own right, as they rely on a space partitioning technique combined with using the existing dominance counting algorithm. The effectiveness of our algorithm is experimentally verified.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; G.3 [**Ma-thematics of Computing**]: Probability and statistics

General Terms

Algorithms

Keywords

Uncertain data, probabilistic skyline

1. INTRODUCTION

Skyline queries are widely used in multi-criteria decision making, where a choice that scores high in one criterion may score low in another (e.g., a hotel very close to the beach but very expensive). The query returns all data points that are not dominated by any other point in a data set, where a point p_1 *dominates* another point p_2 if p_1 is no worse than p_2 in all dimensions and better than p_2 in at least one dimension. The points returned by a skyline query are called *skyline points* in the database community [5], and *maximal points* in the computational geometry community [24].

Many algorithms have been proposed to efficiently answer skyline queries [5, 28, 1, 21, 22], all of which deal with traditional data where no uncertainty is involved. However, uncertainty in data is inherent in many applications such as sensor networks, scientific data management, data integration and location-based applications, where data can take different values with probabilities [9, 23]. The uncertainty in data is typically modeled as a *probability density function* (pdf). Little work has been done to answer skyline queries with such uncertain data. [23] first introduced *probabilistic skyline queries* that answer skyline queries for data with discrete uncertainty, i.e., each uncertain object is associated with a set of instances and corresponding probabilities to take those particular instances. Instances of the same object are mutually exclusive, i.e., at most one can exist at a time. Their skyline query is similar to other probabilistic queries for uncertain data such as ranking queries [13, 26, 16], nearest-neighbor queries [9, 11, 4] and range queries [8, 27], where a probability threshold is used to prune away objects that cannot meet the threshold. For probabilistic skyline queries, only objects with skyline probabilities greater than or equal to the threshold are returned. The skyline probability of an instance that belongs to an object is the probability that this instance occurs and is not dominated by any occurring instance of another object. The skyline probability of an object is the sum of the skyline probabilities of all its instances (because the instances are mutually exclusive).

1.1 Motivations

Example 1 (Motivation): *Alice just got an MBA degree and is looking for jobs at various companies. Each company has multiple jobs that can be offered to MBAs. These jobs vary in titles, work units (departments) and geographic locations, which are not criteria in her decision making. The two criteria in deciding which company she wants to work for are the annual salary and the job security (shown in Figure 1), both the bigger the better. The salaries vary among the available jobs in the company, and the job security of each is quantified by a numerical score (e.g., as reported by credit-rating agencies or financial analysts). A company can be considered as an uncertain object with its job openings as instances. At any time, an MBA can only take one job from*

PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

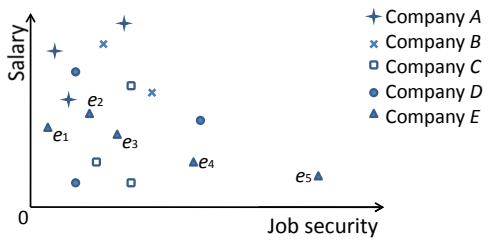

Figure 1: Companies and job openings

a company, and the job offers between different companies are independent from each other. Each job is associated with a probability that the particular job will be offered to an MBA by the company. This probability may differ from one job to another, because some positions are easier to get than others or because there are more positions of a certain kind than others. Furthermore, the probabilities over all job instances of a company may add up to less than 1, as a result of some job that might be offered by the company but is unknown to the MBA at the time of her job hunting. The relative importance of salary and job security varies among different MBAs. For some, salary is most important while others prefer steady jobs. Even for the same MBA like Alice, her interest might change from focusing on salaries to preferring secure jobs over time.

While [23] proposed a couple of algorithms to compute the probabilistic skyline, both algorithms use a probabilistic threshold to help prune objects with skyline probabilities below the threshold. Also, instances of an object are assumed to have equal probabilities that sum up to 1. There are several problems with this "thresholding" approach as well as their simplified uncertain data model.

First, the relative importance among the criteria of the skyline computation for a specific user is usually unknown to the system, hence for skyline queries with certain data, typically all skyline points are returned to the user. We use the term "*utility*" to refer to the satisfaction of a user when given a point. It is the responsibility of the user to identify the interesting points from the skyline set according to his/her own utility function. In case of uncertain objects with multiple instances, each instance of an object has a skyline probability from which the object's skyline probability is computed. If we threshold out uncertain objects (thus all its instances) based solely on their skyline probabilities like in [23], we are making assumptions on the utility of a user in the sense that the user will not be interested in any uncertain object whose skyline probability is below a certain threshold. However, it is possible that for some user, an uncertain object with a relatively low skyline probability (below the threshold) has instances whose utilities are so huge to the user as to make them non-negligible.

Example 2: *In Figure 1, Company E is very appealing to Alice for its best average job security. However, its skyline probability may be low due to the fact that all its jobs except e_5 are dominated by multiple jobs from other companies, hence could have been discarded if we had followed the thresholding approach in [23]. As a result, Alice could have missed a good opportunity to apply for Company E that has the most secure job e_5.*

As indicated in Example 1, utility is user-defined, may change over time for the same user, or simply be unknown to the system at the time of the skyline analysis. Therefore, it is unfeasible to replace the skyline probability by skyline probability*utility and threshold on this new quantity.

Besides the main problem above, the thresholding approach suffers from the inherent problem of selecting a suitable threshold in search for interesting uncertain objects: A high threshold may lead to empty results, and hence the query needs to be restarted with a lower threshold; a low threshold may produce too many results and increase the query response time [4]. Moreover, the performance of the heuristic pruning methods (top-down, bottom-up) in [23] depends heavily on the characteristics of the data set being used and the value of the probability threshold. For some data and threshold, the skyline probability computation may have to be done for a large number (if not all) of the instances.

Finally, the assumptions in [23] of equal-probability among instances, and that instance probabilities add up to 1, may not hold in many practical situations (e.g., it does not hold in Example 1): An uncertain object is likely to take one instance more frequently than another; the probabilities over all its instances may add up to $x < 1$, leaving a probability $1 - x$ that the object is not present (or unknown job as in Example 1). This is known as the "tuple uncertainty" [25].

The problems with the current approach motivate us to explore a new perspective in probabilistic skyline analysis by computing skyline probabilities of all instances. The outcome of such analysis is useful to all users in their decision making, despite their different utilities.

1.2 Our Contributions

The First Sub-Quadratic Algorithm for Computing All Skyline Probabilities: While the algorithm for computing all skyline probabilities (i.e., when the threshold equals 0 in the probabilistic skyline query) in [23] has a worse-case complexity $O(n^2)$ for objects of dimension $d = 2$, the algorithm we present in this paper has a worst-case time complexity that is provably sub-quadratic: It is $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$ for $d = 2$, and more generally $\tilde{O}(n^{2-\frac{1}{d+1}})$ for arbitrary dimension d, where the notation "$\tilde{O}(\cdot)$" is similar to "$O(\cdot)$" except that it ignores factors that are polynomial in $\log n$ (the reason we use this notation for higher dimensions is to avoid unnecessarily cluttering the exposition).

Our algorithm for computing skyline probabilities makes use of two basic algorithms that are individually "bad" if either were used in isolation: A space partitioning algorithm, and one based on the known dominance counting algorithm. In our sub-quadratic algorithm these two techniques work hand in hand to provide the desired solution.

New Probabilistic Skyline Analysis: While the user utility is largely ignored in [23], we propose a general probabilistic skyline analysis that takes into account different user utilities without any restriction. We cannot afford to do the skyline analysis for each user according to his/her utility, but we can do it once for all users by computing skyline probabilities for all instances of objects. After that, each user can identify their own interesting instances (or objects) and enjoy the flexibility in making their personal decisions based on the objective analysis results. Furthermore, since we do not use any probability threshold, the typical problems with a thresholding approach are avoided (see Section 1.1).

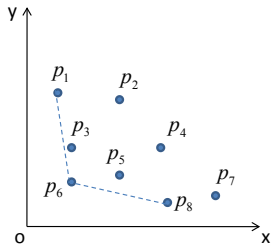

Figure 2: Skyline computation without uncertainty

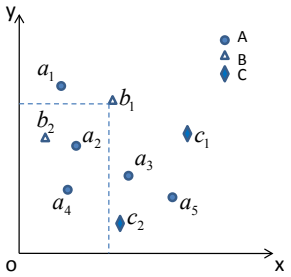

Figure 3: Skyline computation with uncertainty

More General Uncertain Data Model: We remove the two assumptions on the uncertain data model in [23]: The instances of an uncertain object can have different probabilities and the probabilities over all instances of an object may sum up to less than 1. Like [23], we also focus on the skyline computation with discrete uncertainty where uncertain objects are independent of each other.

The rest of the paper is organized as follows: We first introduce definitions and notations that are used throughout the paper in Section 2, then introduce two basic algorithms for computing skyline probabilities in Section 3. We present our sub-quadratic algorithm that makes use of the two methods in Section 4 and show our experimental results in Section 5. The related work is given in Section 6. Finally, we conclude our paper in Section 7.

2. DEFINITIONS AND NOTATIONS

In this section, we first present the preliminaries for our probabilistic skyline analysis, then define our problem, and finally summarize the notations used in the paper.

2.1 Dominance and Skyline

Given a data set S of n certain points: $p_1, \cdots, p_n$ in the data space $\mathcal{D}$ with d dimensions: $\mathcal{D}_1, \cdots, \mathcal{D}_d$, point p_i is said to *dominate* point p_j if $\forall k \in [1,d], p_i.\mathcal{D}_k \leq p_j.\mathcal{D}_k$ and $\exists l \in [1,d], p_i.\mathcal{D}_l < p_j.\mathcal{D}_l$. Without loss of generality, we assume that the smaller the value is, the better it is. This criteria is used consistently throughout the rest of the paper. A point p_i is a *skyline point* or *in the skyline* if it is not dominated by any other point in S. A *skyline query* for certain data returns all skyline points.

Example 3: *In Figure 2, $p_1, p_2, \ldots, p_{10}$ are 10 points in the two-dimensional space $\mathcal{D}$. The skyline points consist of p_1, p_6 and p_8. All the other points are dominated by at least one point, e.g., p_7 is dominated by p_8 and p_5 is dominated by p_6.*

2.2 Probabilistic Skyline

As mentioned in Section 1.2, our uncertain data model is similar to that in [23] where objects have discrete uncertainty, i.e., each uncertain object is associated with multiple instances and corresponding probabilities for the instances to occur. Uncertain objects are assumed to be independent of each other. Each object can only take one instance at a time. Unlike [23], in our model, the sum of probabilities over all instances of an object may add up to less than 1 and the probabilities may vary from one instance to another, as shown in Example 4 below. Since an instance can be considered as a d-dimensional point, we use "instance" and "point" interchangeably throughout the paper, and denote it as "p".

A					B		C	
a_1	a_2	a_3	a_4	a_5	b_1	b_2	c_1	c_2
0.3	0.2	0.1	0.1	0.2	0.2	0.4	0.5	0.5

Table 1: Instance probabilities in Figure 3

An instance p of an object O has a non-zero probability to be a skyline point, as long as no instance of any other object that dominates p exists (there is no need to consider other instances of O since they cannot occur simultaneously with p). The probability for an instance to be a skyline point is called the instance's *skyline probability*. The object's skyline probability is the sum of the skyline probabilities over all its instances.

Example 4: *In Figure 3, we have three uncertain objects A, B, C with multiple instances. The probabilities of instances are listed in Table 1. Each object has a skyline probability. For example, B has two instances b_1 and b_2. b_2 is not dominated by any point, so its skyline probability is simply its own probability 0.2. For b_1 to be a skyline point, none of the points that dominate b_1 (i.e., a_2, a_4, b_2, points in the rectangle) should exist. Hence its skyline probability is 0.4 * (1 - 0.2 - 0.1) = 0.28. The skyline probability of B is 0.48.*

2.3 The Problem

The problem that we address in this paper is to compute the skyline probabilities of all instances, from which the skyline probabilities of all objects can be computed (refer to Section 1.1 and 1.2 for motivations of the problem). We propose several algorithms to solve the problem. The input and the output of the algorithms are given below.

Input: m independent uncertain objects denoted as $O_1, \ldots, O_m$ and, for each object O_i, a set S_i of n_i instances of that object: Each S_i consists of d-dimensional points with probabilities that add up to at most 1. We use S to denote $\cup_{i=1}^{m} S_i$ and n to denote $|S| = \sum_{i=1}^{m} n_i$ where $n_i = |S_i|$. For each point $p \in S$, we use $Pr(p)$ to denote its probability.

Output: For all $p \in S$, the skyline probability of p (let $p \in S_j$ where $1 \leq j \leq m$):

$$Pr_{sky}(p) = Pr(p) \cdot \prod_{i=1, i \neq j}^{m} (1 - \sum_{p' \in D_{S,i}(p)} Pr(p')) \quad (1)$$

where $D_{S,i}(p)$ denotes the set of instances of object O_i in S that dominate p. We call it *dominance set*. Note that we do not count the "effect" of instances that dominate p and come from the same object as p (i.e., instances in $D_{S,j}(p)$). This is because an uncertain object can take at most one instance at a time; the existence of p ($Pr(p)$) already ensures that none of the other instances of O_j exists. We use $D_S(p)$ to denote $\cup_{i=1}^{m} D_{S,i}(p)$ $(i \neq j)$, i.e., all instances in S that dominate p and are not from the same object as p.

From Equation 1, we can see that the skyline probability of p ($p \in S_j$) consists of two parts: p's own probability $Pr(p)$ and the probability that p is not dominated by any instance from other objects, which is computed as the product of probabilities that none of instances from other objects that

Notation	Meaning
m	number of all uncertain objects
n	number of all instances
d	number of dimensions
O_i	the ith uncertain object
n_i	number of instances of O_i
S	the set of all instances ($n = \|S\|$)
S_i	the set of instances of O_i ($n_i = \|S_i\|$)
p	point/instance in S
$Pr_{sky}(\cdot)$	skyline probability
$D_{S,i}(p)$	instances of O_i in S that dominate p
$\sigma_i(p)$	sum of probabilities of O_i's instances that dominate p
$\beta(p)$	the probability that p is not dominated by any instance of other object

Table 2: Summary of notations

dominate p exist. We denote the second part as $\beta(p)$, i.e.

$$\beta(p) = \prod_{i=1, i\neq j}^{m} (1 - \sum_{p' \in D_{S,i}(p)} Pr(p'))$$

Example 5: *In Figure 3, instance b_1 is dominated by instances a_2, a_4 and b_2. Therefore, $D_S(b_1) = D_{S,A}(b_1) = \{a_2, a_4\}$ and $\beta(b_1) = 1-(Pr(a_2) + Pr(a_4)) = 0.7$.*

The skyline probability of an uncertain object O_i is simply the sum of skyline probabilities of its instances, i.e.

$$Pr_{sky}(O_i) = \sum_{p \in S_i} Pr_{sky}(p)$$

Since we can always compute $Pr_{sky}(p)$ from $\beta(p)$, later in the paper we will focus on computing $\beta(p)$ rather than the actual skyline probability of an instance. The major notations used in the paper are summarized in Table 2 for reference.

3. TWO BASIC ALGORITHMS

In this section, we propose two basic algorithms for computing $\beta(p)$ that will be used in our sub-quadratic solution for $d = 2$ in Section 4.2 as well as the extended solution to high dimensions in Section 4.3.

3.1 The Grid Method

We first present this space partitioning method for uncertain data with two dimensions: $d = 2$. Let C_x (C_y) be the set of x (y) coordinates of instances in S. Then $|C_x| \leq n, |C_y| \leq n$, as there might be instances with the same x or y coordinates. The elements of $C_x \times C_y$ form a grid of at most n^2 vertices, consisting of the original n instances (along with their probabilities), and the rest of the vertices that do not correspond to original instances. We therefore consider the latter as *"virtual"* instances of a non-existent object O_0 and assign to each a probability of zero.

Example 6: *Figure 4 shows such a grid with five instances $p_1, ..., p_5$ marked as solid points. All the other vertices on the grid (7 in total), which are virtual instances, are marked as hollow points (e.g., vertex v).*

We seek to compute $\beta(p)$ for every grid vertex p (even the virtual ones). The reason that we also compute for virtual instances will be explained later in Section 4.1. Now we focus on the algorithm for doing this in $O(mn^2)$ time.

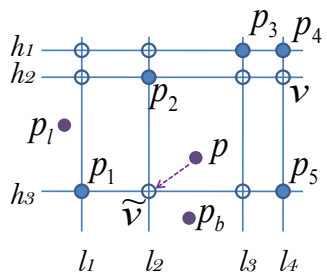

Figure 4: Space partitioning using a grid

First we observe that it suffices to compute, for each vertex p, the m sums $\sigma_1(p), \ldots, \sigma_m(p)$ where

$$\sigma_i(p) = \sum_{p' \in D_{S,i}(p)} Pr(p'), \qquad i = 1, \ldots, m \tag{2}$$

because once we have those sums we can compute the desired $\beta(p)$ values for all p's (let p's object be O_j) in the grid in $O(mn^2)$ time, following the equation below:

$$\beta(p) = \prod_{i=1, i\neq j}^{m} (1 - \sigma_i(p)) \tag{3}$$

So we focus on the computation of all the $\sigma_i(p)$'s. The algorithm for computing them is given next.

1. **Process the horizontal grid lines:** For each horizontal line of $O(n)$ vertices, from left to right, we compute for every vertex p of that line the m *horizontal summations* $\sigma_i^*(p)$'s which are similar to the $\sigma_i(p)$'s except that they are defined one-dimensionally and relative only to the horizontal line that contains p (i.e., as if nothing exists other than what is on that horizontal line). Formally, let $p \in h$ where h is the horizontal line that contains p, let $p' <_h p$ denote the relationship "p' is to the left of p and is on the same horizontal line h as p", then we have

$$\sigma_i^*(p) = \sum_{p' \in S_i, p' <_h p} Pr(p') \tag{4}$$

   We compute $\sigma_i^*(p)$'s in the following way: If p is the first vertex on h, set all m $\sigma_i^*(p)$'s to zero. Otherwise, let the left neighbor of p on h be $\hat{p}$. Copy all m $\sigma_i^*(\hat{p})$'s to $\sigma_i^*(p)$'s. If $\hat{p}$ is an original (not virtual) instance and $\hat{p} \in S_j$, add $Pr(\hat{p})$ to $\sigma_j^*(p)$.

   The above takes $O(m)$ for each p on the horizontal line, hence we can compute $\sigma_i^*(p)$'s for all p's in time $O(mn)$ per horizontal line.

   Example 7: *Figure 4, p_1 is an instance of O_1 with probability 0.8, p_2 and p_4 are instances of O_2 with probability 0.5 each, p_3 and p_5 are instances of O_3 with respective probabilities 0.6 and 0.1. Then for p_4 on the horizontal line h_1, $\sigma_1^*(p_4) = \sigma_2^*(p_4) = 0$ while $\sigma_3^*(p_4) =$* 0.6.

2. **Process the vertical grid lines:** We compute $\sigma_i(p)$'s for the vertices of each vertical grid line in bottom to top order:

$$\sigma_i(p) = \sigma_i^*(p) + \sigma_i(p') + \begin{cases} Pr(p') & \text{if } p' \in S_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where p' is the grid vertex immediately below p on the vertical line l that contains p (hence its $\sigma_i(p')$ is already available because it has already been processed in the bottom-up order for l). If p is the very bottom vertex on l, then $\sigma_i(p) = \sigma_i^*(p)$. Note here we add $Pr(p')$ to $\sigma_i(p)$ to take into account probabilities of original instances on l that dominate p, which are not captured by either $\sigma_i^*(p)$ or $\sigma_i(p')$.

Example 8: *To compute $\sigma_i(p_4)$'s from $\sigma_i^*(p_4)$'s computed in Example 7, we follow Equation 5 (take $i = 3$ for example):*

$$\begin{aligned}\sigma_3(p_4) &= \sigma_3^*(p_4) + \sigma_3(v) + 0\\ &= Pr(p_3) + \sigma_3^*(v) + \sigma_3(p_5) + Pr(p_5)\\ &= 0.6 + 0 + \sigma_3^*(p_5) + 0.1 = 0.7\end{aligned}$$

Similarly, we compute $\sigma_1(p_4) = 0.8$, $\sigma_2(p_4) = 0.5$.

The next theorem states that $\sigma_i(p)$ is correctly computed by the algorithm above. The proof is straightforward, hence omitted.

THEOREM 1. *For any vertex p on the grid, $\sigma_i(p)$ computed by the grid method is the sum of probabilities of instances of O_i that dominate p, i.e. Equation 2 can be deduced from Equation 4 and 5.*

Step 1 of the algorithm takes $O(mn)$ time per horizontal line, thus $O(mn^2)$ total time for all horizontal lines. Step 2 takes $O(mn)$ time per vertical line, thus $O(mn^2)$ total time for all vertical lines. Since it takes an additional $O(mn^2)$ to compute $\beta(p)$'s from $\sigma_i(p)$'s for all p's in the grid, the overall time complexity for computing $\beta(p)$'s is also $O(mn^2)$.

The above algorithm easily generalizes to dimensions $d > 2$, with a rather daunting time complexity of $O(mn^d)$. In the worst case m is proportional to n and the time complexity is then $O(n^{d+1})$. It is interesting that such an algorithm with a discouragingly bad time complexity, will actually play a critical part in the sub-quadratic time solution that we will provide later in Section 4.

3.2 The Weighted Dominance Counting Method

The algorithm reviewed in this section, although inefficient if used as the sole method for solving the problem, will play a useful role as one of the two building blocks for the more efficient solution we give later.

The *weighted dominance counting* (WDC) problem is: Given a set S of n weighted points, compute for each point p of S the sum of the weights of all points in S that dominate p. If all weights are equal to 1, the problem is the same as counting points that dominate p. To be consistent, we use the same dominance concept here as in Section 2.1.

It is well known that the WDC problem can be solved in $O(n(\log n)^{d-1})$ time for d-dimensional points ([24, 20]). This immediately gives an $O(mn(\log n)^{d-1})$ time solution to our problem, by using the WDC algorithm m times (once for each object) on all n instances with their probabilities as weights: For object O_i, we assign the n_i instances of object O_i weights that are equal to their probabilities, and assign the other $n - n_i$ instances weight zero.

These m successive runs of the WDC algorithm give, for every instance $p \in S$ (let p be an instance of object O_j, i.e., $p \in S_j$), the m sums $\sum_{p' \in D_{S,i}(p)} Pr(p')$, i.e., $\sigma_i(p)$'s $(i = 1, \ldots, m)$. From these, it is easy to use an additional $O(m)$ time per instance p in S to compute

$$\beta(p) = \prod_{i=1, i \neq j}^{m} (1 - \sum_{p' \in D_{S,i}(p)} Pr(p'))$$

because each summation within the product is already available from one of the m runs of the WDC algorithm. In the worst case m is proportional to n and the time complexity is $O(n^2(\log n)^{d-1})$. Our paper improves this worst-case time complexity down to $\tilde{O}(n^{2-\frac{1}{d+1}})$.

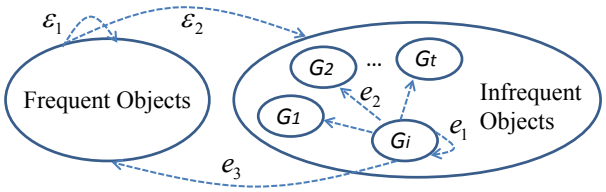

Figure 5: Schema of our algorithm

4. THE IMPROVED ALGORITHM

4.1 Overview

Before presenting our sub-quadratic algorithm for computing all skyline probabilities and later its extension to high dimensions, we first give an intuitive overview on how our scheme works in the two-dimensional case.

Our algorithm works by balancing the use of two inefficient methods: One uses weighted dominance counting (WDC), the other is based on partitioning space into grids. Using the former alone to compute skyline probabilities for all instances would result in an $O(n^2 \log n)$ time complexity, whereas using the latter alone would take $O(n^2)$ time without computing for virtual instances. The schema of our algorithm is illustrated in Figure 5, which shows the interplay between different subsets of uncertain objects. We use a dashed arrow from one set of objects S_A to another set S_B to denote the effect of instances in S_A on the skyline probabilities of instances in S_B. Specifically, by "effect of $p \in S_A$ on $p' \in S_B$" we mean the contribution of p to the summation $\sigma_i(p')$. We use ε and e to denote the effect computed during WDC and the effect computed during the grid method.

The main ideas of our algorithm are as follows:

1. We use WDC for objects that are "frequent" with a number of instances that exceeds a special value μ. Objects that are not frequent are called "infrequent" objects. Specifically, we compute the effect of each frequent object on the skyline probabilities of all n instances in time $O(n \log n)$ using the WDC algorithm. This is captured by ε_1 and ε_2 in Figure 5.

2. We merge the infrequent objects into groups such that each group contains between μ and 2μ instances (except that the last group could contain fewer than μ instances). In Figure 5, the set of infrequent objects is divided into t groups: $G_1, \ldots, G_t$.

3. We use the grid method for each group of the infrequent objects. It is crucial here that we compute $\beta(p)$'s

Input: a set S of n instances from m uncertain objects
Output: all instances with skyline probabilities
1. $Result = \emptyset$
2. partition objects into two sets: F, $\overline{F}$ // Section 4.2.1
3. **for each** O_i in F **do** // Section 3.2
4. obtain $\sigma_i(p)$ for all $p \in S$ by calling WDC(S)
5. **end for**
6. **for each** $p \in S$ (let $p \in S_j$) **do**
7. **for each** O_i in F and $O_i \neq O_j$ **do**
8. $\alpha(p) = \prod_{O_i \in F}(1 - \sigma_i(p))$
9. **end for**
10. **end for**
11. partition $\overline{F}$ into groups: $G_1, \ldots, G_t$ // Section 4.2.3
12. **for each** G_i in $\overline{F}$ **do** // Section 3.1
13. obtain $\beta_i(p)$ for all $p \in I_i$ by calling Grid(I_i)
14. **for each** $p \notin I_i$ **do**
15. locate p in a grid cell C of G_i
16. $\beta_i(p) = \beta_i(p')$ // p': bottom-left corner of C
17. **end for**
18. **end for**
19. **for each** $p \in S$ **do**
20. $Pr_{sky}(p) = Pr(p) \cdot \alpha(p) \cdot \prod_{i=1}^{t} \beta_i(p)$
21. $Result = Result \cup (p, Pr_{sky}(p))$ // add the pair
22. **end for**
23. **return** *Result*

Figure 6: Computing skyline probabilities

for virtual instances on the grid as well as original instances. Although this takes cubic time with regard to the number of instances in the group, it lends itself to the efficient computation of the effects of the group on all the other instances outside of the group through the use of a "bucketing" technique that processes together all the non-group instances that fall within a cell of the grid (a "bucket"); the details are given later in Section 4.2.3. In Figure 5, e_1 captures the effect of a group G_i on its own instances while e_2 and e_3 capture the effect of G_i on all the other instances.

We use the case $d = 2$ to present our algorithm, and later extend it to the case $d > 2$. The case $d = 1$ is trivial to solve in $O(n)$ time and hence we omit it.

4.2 Our Algorithm for Two Dimensions

Figure 6 shows our algorithm for computing all skyline probabilities that follows the schema in Section 4.1. A more detailed explanation for two-dimensional data is given below.

4.2.1 Partitioning Objects

Partition the m objects into two sets: a set of *frequent* objects (F), consisting of every object O_i for which $n_i > (n \log n)^{\frac{1}{3}}$; the other objects (in $\overline{F}$) are said to be *infrequent*. We shall process the frequent objects differently from the infrequent ones. The value $(n \log n)^{\frac{1}{3}}$ is the partitioning point μ we mentioned in Section 4.1.

4.2.2 Handling Frequent Objects

Line 3 to 10 in Figure 6 shows how to handle frequent objects. For every frequent object O_i, we use WDC once (as explained in Section 3.2) to obtain $\sigma_i(p)$'s for every $p \in S$. Let p be an instance of object O_j. We compute the quantity

$$\alpha(p) = \prod_{O_i \in F, O_i \neq O_j} (1 - \sigma_i(p))$$

as the effect of frequent objects on any instance p, which is illustrated by ε_1 and ε_2 in Figure 5.

4.2.3 Handling Infrequent Objects

Line 11 to 18 in Figure 6 shows how to handle infrequent objects. We first go through the infrequent objects and, while doing so, partition the infrequent objects into groups, as follows: A new group is initially created, with no objects in it – we refer to it as the current group. The next encountered object is included in the current group: If this causes the number of object instances in the current group to exceed $(n \log n)^{\frac{1}{3}}$ (i.e., μ), then that group is considered done (i.e., no longer current) and a new (initially empty) current group is started (to which the next object encountered is added, etc).

Comment and notation. Note that the number of instances in a completed group is between μ and 2μ, because an infrequent object does not add more than μ to the current group it joins. The number of completed groups (call it t) is obviously no more than $n/(n \log n)^{\frac{1}{3}}$. We denote these groups as $G_1, \ldots, G_t$. We use m_i to denote $|G_i|$ (= the number of objects in G_i), I_i to denote the set of all instances of the m_i objects in G_i (hence $|I_i| = \sum_{O_j \in G_i} n_j$).

The next step computes, for each group G_i, the quantity

$$\beta_i(p) = \prod_{O_j \in G_i, O_j \neq O_k} (1 - \sum_{p' \in D_{I_i,j}(p)} Pr(p'))$$

for every $p \in S$ (let p belong to object O_k). The challenge is how to do this efficiently — we can no longer afford to use WDC because there are many objects in a G_i. We do the following instead:

For each of the groups $G_1, \ldots, G_t$ in turn, compute for every instance p in S the quantity $\beta_i(p)$, $i = 1, \ldots, t$. This is done as follows for G_i:

First, we use the grid method on G_i. This gives $\beta_i(p)$ for every grid vertex p, which is the effect of G_i on its own instances (i.e., e_1 in Figure 5) as well as virtual ones. This takes cubic time with regard to the number of original instances in the grid.

Second, we compute the effect of G_i on all instances in S that are not on the grid of G_i, i.e., instances from other groups of infrequent objects and instances from frequent objects, as illustrated by e_2 and e_3 respectively in Figure 5. The grid for G_i partitions the plane into $O(|I_i|^2)$ regions (called cells). We use binary search to first locate each point p of $S - I_i$ in the grid cell in which it lies (two binary searches per point – one to locate the vertical slab in which it lies and the other to locate the cell within the vertical slab). Then for each such point p we set $\beta_i(p) = \beta_i(p')$ where p' is the bottom-left corner of the grid cell containing p. Note that if such p' cannot be found (as illustrated in Example 9) for p, then $\beta_i(p) = 1$. Since p' can be a virtual instance, it is crucial that in the grid method we compute for all vertices including the virtual instances so that the effect of G_i on p can be obtained instantaneously.

Example 9: *Assume that all instances $p_1, \ldots, p_5$ in Figure 4 come from a group G_i of the infrequent objects. Let p, p_l, p_b be instances that either come from other groups or*

belong to frequent objects. Then $\beta_i(p) = \beta_i(\tilde{v})$ where $\tilde{v}$ is the bottom-left vertex of the cell p resides in. For p_l and p_b, the regions they are located in are not closed regions and there are no corresponding bottom-left vertices. Therefore, we set $\beta_i(p_l) = \beta_i(p_b) = 1$.

4.2.4 Computing Skyline Probabilities

For every p in S, compute the desired $Pr_{sky}(p)$ as

$$Pr_{sky}(p) = Pr(p) \cdot \alpha(p) \cdot \prod_{i=1}^{t} \beta_i(p) \quad (6)$$

Equation 6 is equivalent to Equation 1 because the set of frequent objects and the groups of infrequent objects are partitions of all uncertain objects; also, the uncertain objects are all independent from each other.

4.2.5 Complexity Analysis

In this section, we analyze the time complexity of our algorithm in Figure 6 for two-dimensional uncertain objects.

Partitioning objects (line 2 in Figure 6) takes $O(m)$ time by going through all m objects. Handling frequent objects consists of calling WDC algorithm to obtain $\sigma_i(p)$'s for all $p \in S$ (line 3 to 5) and then further computing $\alpha(p)$'s (line 6 to 10). The first part takes a total of $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$ time, as the number of frequent objects is no more than $n/(n \log n)^{\frac{1}{3}}$, and the WDC for each of these takes $O(n \log n)$ time. The second part takes a total of $O(n^{\frac{5}{3}}/(\log n)^{\frac{1}{3}})$ time, as it takes $O(n/(n \log n)^{\frac{1}{3}})$ time to compute $\alpha(p)$ for each p and there are altogether n such p's in S. Therefore, the total time cost to handle frequent objects is $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$.

Handling infrequent objects is more complicated. Grouping at line 11 takes $O(m)$ time. For each group G_i, calling the grid method at line 13 takes time $O(|I_i|^3)$ where I_i is the set of all instances in G_i. This is $O(n \log n)$ because $|I_i| \leq 2(n \log n)^{\frac{1}{3}}$ is ensured by our grouping method (see Section 4.2.3). For each p in $S - G_i$, we can locate it in a grid cell of G_i in $O(\log |I_i|) = O(\log n)$ time by two binary searches. Line 16 takes constant time since $\beta_i(p')$ is already available from line 13. Thus the above takes a total of $O(tn \log n)$ time, where t is the number of the groups of infrequent objects. Since $|I_i| > (n \log n)^{\frac{1}{3}}$ for any group G_i except the last one, $t = O(n/(n \log n)^{\frac{1}{3}})$, resulting in a time complexity of $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$ for handling infrequent objects from line 11 to line 18 in Figure 6.

Finally, computing the desired skyline probabilities for all p in S takes $O(tn)$ time, which is $O(n^{\frac{5}{3}}/(\log n)^{\frac{1}{3}})$.

The overall time complexity of our algorithm is, as argued in the analysis of each step, $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$.

4.3 Extended Algorithm for Higher Dimensions

We use the same notation as in the previous section except that, because the case of $d > 2$ is more complicated, our analysis uses the notation "$\tilde{O}(\cdot)$" which is similar to the "$O(\cdot)$" notation except that it ignores polylog factors (whereas the former ignores only constant factors).

This section shows that for $d > 2$ we can achieve a worst-case performance of $\tilde{O}(n^{2-\frac{1}{d+1}})$ time to compute skyline probabilities for all instances. In the following we make no attempt to minimize the polylogarithmic factor hiding behind the $\tilde{O}(\cdot)$ notation — doing so would unnecessarily clutter the exposition, as we would have to define what constitutes a "frequent" object in a more cumbersome manner.

Figure 6 is the algorithm for computing all skyline probabilities regardless of the dimensions of the data. Hence we only address parts that are different from the previous section where data is two-dimensional.

4.3.1 Partitioning and Grouping

Partition the m object into two sets: a set of *frequent* objects (F), consisting of every object O_i for which $n_i > n^{\frac{1}{d+1}}$; the other objects (in $\overline{F}$) are said to be *infrequent*. When grouping infrequent objects, we keep adding objects into groups until the number of instances in a group exceeds $n^{\frac{1}{d+1}}$. Then we start a new empty group and repeat the process until all objects have been put into groups.

4.3.2 Locating Points in the Grid

As we know from Section 4.2.3, the effect of a group G_i on an instance p that is not in G_i can be evaluated by first locating p in the grid of G_i and then taking the β_i value of the bottom-left vertex of the cell as p's own β_i. Now that we have higher dimensions ($d > 2$), the grid of G_i consists of $O(|I_i|^d)$ cells. When locating p, we have to do the binary search d times – one in each dimension to locate the slab in which it lies for that dimension.

4.3.3 Complexity Analysis

Partitioning objects takes $O(m)$ time as before. Calling WDC at line 4 in Figure 6 now takes a total of $\tilde{O}(n^{2-\frac{1}{d+1}})$ time, as the number of frequent objects is no more than $n^{1-\frac{1}{d+1}}$, and the WDC for each of these takes $\tilde{O}(n)$ time (see Section 3.2). Computing $\alpha(p)$'s takes a total of $\tilde{O}(n^{2-\frac{1}{d+1}})$ time, as it takes $\tilde{O}(n^{1-\frac{1}{d+1}})$ time for each p in S (because there are at most $n^{1-\frac{1}{d+1}}$ frequent objects). Therefore, the total time cost to handle frequent objects is $\tilde{O}(n^{2-\frac{1}{d+1}})$.

To handle infrequent objects, we first partition them into groups in $O(m)$ time. For each group, calling the grid method at line 13 takes time $\tilde{O}(|I_i|^{d+1})$, which is $\tilde{O}(n)$ because $|I_i| \leq 2n^{\frac{1}{d+1}}$. For each p in $S - G_i$, we locate it in a grid cell of G_i in $O(\log |I_i|) = O(\log n)$ time by d binary searches. The total time complexity of the above is $\tilde{O}(tn) = \tilde{O}(n^{2-\frac{1}{d+1}})$, where $t < n^{1-\frac{1}{d+1}}$ is the number of groups of infrequent objects.

Finally, skyline probabilities for all p's can be computed in $O(tn)$ time, i.e. $O(n^{2-\frac{1}{d+1}})$. Hence the overall time complexity of our algorithm is $\tilde{O}(n^{2-\frac{1}{d+1}})$. Since the use of higher-dimensional WDC implies that a factor exponential in d is hiding behind $\tilde{O}$, the scheme is not practical for large d's and the straightforward $O(dn^2)$ algorithm is better.

5. EXPERIMENTAL EVALUATION

Algorithms with good asymptotic complexity can often be impractical unless n is huge. Our algorithm is not one of those, and is in fact practical even for moderate values of n, as the following brief experimental evaluation demonstrates.

We performed our experiments on synthetic data sets of two dimensions ($d = 2$) and compared it with alternative algorithms for computing all skyline probabilities. We implemented all algorithms in C# and the experiments were performed on a MAC with Intel T2500 2.0GHz CPU and

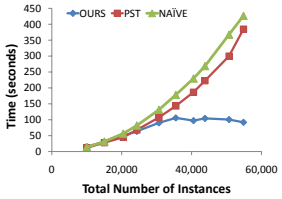

Figure 7: Running time on different algorithms

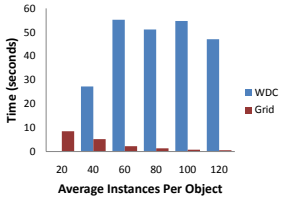

Figure 8: Effect of the average instance count

2GB main memory. In the rest of the section, we first describe the data sets used in our experiments and then discuss the experimental results.

5.1 Data Sets

We generated our synthetic data sets similarly to [23, 5] as follows: We first randomly generated the centers c of each uncertain object. Let $p.x$ and $p.y$ be the values of the first and the second dimension of an instance p, i.e. the x and y coordinates of p if we think of the instance as a point in a 2-d space. Both dimensions have a domain [1, 1000]. The default number of uncertain objects m is 1000. The number of instances for an object is uniformly distributed in the range [1, 40] by default. Therefore, by default the expected number of instances n is around 20,000. The x and y values of an instance are randomly generated in the rectangle centered at c with the edge size uniformly distributed in the range [1, 200].

5.2 Efficiency and Scalability

We compared our algorithm (`OURS`) against the priority search tree based algorithm (`PST`) [19], which queries the priority search tree built upon the data set S to find all instances that dominate a given instance p, i.e. the dominance set $D_S(p)$. The skyline probability of p then can be computed from Equation 1. The reason that we can leverage `PST` for finding $D_S(p)$ is that this problem can be converted to a two-dimensional range query: Retrieve all instances p' such that $p'.x \leq p.x$ and $p'.y \leq p.y$ (we make sure that if either is "=", then the other must be "<"). It is well known that this can be done for query point p in time $O(\log |S| + |D_S(p)|)$. Let $n = |S|$, then the tree takes $O(n)$ space and can be built in $O(n \log n)$ time. We also implemented the naïve algorithm (`NAIVE`) with an $O(n^2)$ time complexity for benchmarking. It checks all n instances to determine the dominance set of p before computing the skyline probability of p.

Figure 7 shows the running time of the three algorithms: `OURS`, `PST` and `NAIVE` on data sets with different total number of instances (i.e. n's) ranging from 10k to 60k. We used the default number of instances per object (in [1, 40]). As expected, `NAIVE` has the worst performance overall followed by `PST`, both of which have a worst-case time complexity of $O(n^2)$. Since our algorithm has a worst-case time complexity of $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$, it performs better with larger n's, as shown in Figure 7. `OURS` outperforms both alternatives even for moderate values of n such as $n = 30k$, while for smaller n's, its running time is slightly longer than the other two. As n increases, the advantage of our algorithm over the other two becomes more salient.

5.3 Effect of Instance Count per Object

Since we use the cutoff value $\mu = (n \log n)^{\frac{1}{3}}$ to partition objects into frequent and infrequent objects, different numbers of instances per object can affect the number of frequent objects versus the number of infrequent ones, resulting in different effects on the time cost of the `WDC` and the grid method (`Grid`) within our algorithm. For example, with $n = 20k$, $\mu = 65$. If the instance count per object generated in our synthetic data set is between [1, 80], there can be many more infrequent objects than frequent ones, since the expected instance count is $40 < \mu$. As a result, the time cost of `Grid` in our algorithm will be bigger.

We evaluated the effect of instance count per object on the time cost of `WDC` and `Grid` in Figure 8. The average instance count per object varies from 20 to 120 and n is fixed to be around 20k. We can see that `WDC` generally takes much longer than `Grid`. Given $\mu = 65$, as instance count increases, the number of infrequent objects decreases, hence the time cost of `Grid` decreases. The time cost of `WDC` first increases as a result of more frequent objects, then maintains at a level.

6. RELATED WORK

Many research efforts have been devoted to skyline computation and its variations. The skyline was first introduced in the context of databases in [5], although efficient algorithms for finding maxima of vectors exist as early as in [14] in computational geometry. Various skyline algorithms have been designed for different settings, such as computing skylines in a distributed setting [1, 30], computing subspace skylines [21, 22], maintaining incremental skylines for dynamic data sets [28], etc. In addition, many interesting variations of skyline computation have been explored recently, including k most representative skyline points [17], reverse skyline queries [12, 15] and privacy-preserving skylines [7].

As research in uncertain databases draws more and more attention in the database community, much work has been done to query uncertain data, such as range queries [10, 27], probabilistic nearest-neighbor queries [9, 11, 4], ranking queries [26, 16, 13] and so on. Most of work assumes some uncertain database models, e.g., attribute-level uncertainty model [25, 2] or tuple-level uncertainty model [3, 6]. Skyline queries with uncertain data have also been studied recently [29, 23, 15], among which [23] is closest to our work. While [23] designed algorithms for computing probabilistic skylines, [15] further studied the problem of probabilistic reverse skyline computation in both monochromatic and bichromatic cases. [29], on the other hand, proposed algorithms for continuous probabilistic skyline queries on uncertain data streams. Although different from each other, all three papers leveraged a probability threshold for pruning the result set. We take a different approach by abandoning the use of thresholds and computing the skyline probabilities for all instances instead of returning the p-skyline as in [23]. This allows more flexibility for users to utilize the skyline results according to their own utilities rather than focus on the probability alone. [18] also looks beyond the probability and uses a score based on both the probability and the distance in the spatial join of probabilistic data, and returns top-k results ranked according to the score. Since the relative importance (i.e. utility) varies from user to user and may even change over time, [18] has to recompute the results for each user or whenever a user's utility changes. However, with our

approach to the probabilistic skyline analysis, we only need to compute the skyline probabilities once for all users, leaving users the largest flexibility to make their own decisions based on their current utilities and the skyline probabilities returned by the algorithm.

7. CONCLUSIONS

To the best of our knowledge, our paper is the first to study the problem of computing all skyline probabilities for data with discrete uncertainty. We designed an efficient algorithm based on space partitioning and weighted dominance counting. We gave strict complexity analysis of our sub-quadratic algorithm for $d = 2$ and showed how to extend it to higher dimensions. Our algorithm provides the user with the biggest flexibility in identifying their own interesting skyline instances by returning skyline probabilities of all instances and making no assumptions on how the user will use the skyline results (i.e. the user utility is not restricted in any way). Such skyline analysis only needs to be done once for all users, and the results are useful for all users regardless of their different utilities. Our future work includes extending the current algorithm to also work for the probabilistic skyline queries with thresholds, and studying the problem of computing all skyline probabilities for data with continuous uncertainty.

Acknowledgments

Portions of this work were supported by Grant CNS-0627488 from the National Science Foundation, by Grant FA9550-09-1-0223 from the Air Force Office of Scientific Research, and by sponsors of the Center for Education and Research in Information Assurance and Security.

8. REFERENCES

[1] V. Akrivi, A. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. Skypeer: Efficient subspace skyline computation over distributed data. In *ICDE*, 2007.

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

[3] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: databases with uncertainty and lineage. In *VLDB*, 2006.

[4] G. Beskales, M. A. Solima, and I. F. Ilyasu. Efficient search for the top-k probable nearest neighbors in uncertain databases. In *VLDB*, 2008.

[5] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[6] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD*, 2005.

[7] B.-C. Chen, K. LeFevre, and R. Ramakrishnan. Privacy skyline: privacy with multidimensional adversarial knowledge. In *VLDB*, 2007.

[8] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[9] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *TKDE*, 2004.

[10] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, 2004.

[11] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, 2008.

[12] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, 2007.

[13] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *SIGMOD*, 2008.

[14] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. of ACM*, 1975.

[15] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, 2008.

[16] X. Lian and L. Chen. Probabilistic ranked queries in uncertain databases. In *EDBT*, 2008.

[17] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. *ICDE*, 2007.

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

[19] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 1985.

[20] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Springer-Verlag New York, Inc., 1984.

[21] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, 2007.

[22] J. Pei, A. W.-C. Fu, X. Lin, and H. Wang. Computing compressed multidimensional skyline cubes efficiently. *ICDE*, 2007.

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

[24] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

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

[26] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Urank: formulation and efficient evaluation of top-k queries in uncertain databases. In *SIGMOD*, 2007.

[27] Y. Tao, R. Cheng, X. Xiao, W. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, 2005.

[28] P. Wu, D. Agrawal, O. Egecioglu, and A. El Abbadi. Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation. *ICDE*, 2007.

[29] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. Yu. Probabilistic skyline operator over sliding windows. In *ICDE*, 2009.

[30] L. Zhu, S. Zhou, and J. Guan. Efficient skyline retrieval on peer-to-peer networks. *Future Generation Communication and Networking*, 2007.