Asymptotically Efficient Algorithms for Skyline Probabilities of Uncertain Data

MIKHAIL J. ATALLAH and YINIAN QI
Purdue University
and
HAO YUAN
City University of Hong Kong

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. While some earlier work focused on probabilistic skylines with a given threshold, Atallah and Qi [Atallah and Qi 2009] studied the problem to compute skyline probabilities for all instances of uncertain objects without the use of thresholds, and proposed an algorithm with sub-quadratic time complexity. In this work, we propose a new algorithm for computing all skyline probabilities that is asymptotically faster: worst-case $O(n\sqrt{n}\log n)$ time and $O(n)$ space for 2D data; $O(n^{2-\frac{1}{d}}\log^{d-1} n)$ time and $O(n\log^{d-2} n)$ space for d-dimensional data. Furthermore, we study the online version of the problem: Given any query point p (unknown until the query time), return the probability that no instance in the given data set dominates p. We propose an algorithm for answering such an online query for d-dimensional data in $O(n^{1-\frac{1}{d}}\log^{d-1} n)$ time after preprocessing the data in $O(n^{2-\frac{1}{d}}\log^{d-1})$ time and space.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*; G.3 [**Mathematics of Computing**]: Probability and statistics

General Terms: Algorithms

Additional Key Words and Phrases: 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, or a restaurant with excellent cooks but rude waiters). 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 [Börzsönyi et al. 2001], and *maximal points* in the computational geometry community [Preparata and Shamos 1985].

Many algorithms have been proposed to efficiently answer skyline queries [Börzsönyi et al. 2001; Wu et al. 2007; Vlachou et al. 2007; Morse et al. 2007; Pei et al. 2007], all of which deal with

Portions of this work were supported by National Science Foundation Grants CNS-0915436, CNS-0913875, Science and Technology Center CCF-0939370; by an NPRP grant from the Qatar National Research Fund; by Grant FA9550-09-1-0223 from the Air Force Office of Scientific Research; by sponsors of the Center for Education and Research in Information Assurance and Security; and by a grant from City University of Hong Kong (Project No. 7200218). The statements made herein are solely the responsibility of the authors.
Author's addresses: M.J. Atallah, and Y. Qi, Department of Computer Science, Purdue University, West Lafayette, IN 47907, United States; email: {mja,yqi}@cs.purdue.edu; H. Yuan, Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong; email: haoyuan@cityu.edu.hk.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

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 [Cheng et al. 2004; Pei et al. 2007]. The uncertainty in data is typically modeled as a *probability density function* (pdf). There is a growing body of work on answering skyline queries with such uncertain data. Pei et al. [Pei et al. 2007] first introduced *probabilistic skyline queries* to tackle this problem for objects with discrete uncertainty, where the pdf that describes the uncertain object is not available explicitly, and is instead approximated using a set of collected samples. Hence an uncertain object U_i is modeled as a set of multiple points $p_1, p_2, ..., p_{n_i}$ in the data space as its instances. The number of instances of an uncertain object U_i is denoted as $|U_i| = n_i$. Each instance of U_i is associated with a probability that U_i takes that instance. Uncertain objects are assumed to be independent from each other. Instances of the same object are mutually exclusive, i.e., at most one can exist at any time. This uncertain model is essentially the "x-tuple" model for probabilistic databases [Benjelloun et al. 2006], where an x-tuple consists of multiple tuple *alternatives* and confidence values are attached to alternatives and sum up to ≤ 1 – i.e., the x-tuple is the uncertain object and the tuple alternatives are the object's instances.

The probabilistic skyline query introduced in [Pei et al. 2007] is similar to other probabilistic queries for uncertain data such as ranking queries [Hua et al. 2008; Soliman et al. 2007; Lian and Chen 2008b; Li et al. 2009; Zhang and Chomicki ; Cormode et al. 2009], nearest-neighbor queries [Cheng et al. 2004; Cheng et al. 2008; Beskales et al. 2008] and range queries [Cheng et al. 2003; Tao et al. 2005], 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. Motivation

The above skyline model for uncertain data was extensively discussed and motivated in the literature, where many papers deal with the topic (e.g., [Pei et al. 2007; Zhang et al. 2009; Lian and Chen 2008a]). Paper [Pei et al. 2007] gives an example application from NBA, where game-by-game performance data of multiple players are considered. Each player is modeled as an uncertain object and the player's performance at each game is modeled as an instance of the object. Equal probabilities are assigned to different games of the same player. Below we give a few other motivating examples, together with a rationale for computing the skyline probabilities of all instances instead of thresholding out on skyline probabilities of uncertain objects as in [Pei et al. 2007].

Example 1: *The provision of a service often involves a number of different sub-services: The quality of a patient's experience at an "urgent health care" facility depends on which of the doctors is on duty, which nurse is assigned to the patient, which staff member handles the billing and insurance paperwork, etc. In effect, there is not a single patient experience at that facility, but a collection of possible experiences, one for each k-tuple of team members that the patient interacts with during a visit. Similarly, the quality of the dining experience at a restaurant depends on which of the waiters the customer gets, and which of the cooks prepares the meal. Each possible k-tuple of team members has a probability, and determining which service providers dominate is easily seen to be a skyline computation problem (one in which a service provider is an object and a k-tuple of team members is an instance of that object). But why would one want the detailed skyline probabilities of the instances, when the skyline probability of an object would seem to suffice, which allows the efficient elimination (through thresholding) of dominated objects? The reason is that a customer's valuation function of an instance is variable from customer to customer (and can vary over time for the same customer): Thresholding may eliminate a low-probability object but whose instances (or a subset thereof) are peculiarly appealing to some customers. In other words, probabilities are not all that matters, consequences matter too: Computing all instance probabilities allows for subsequent*

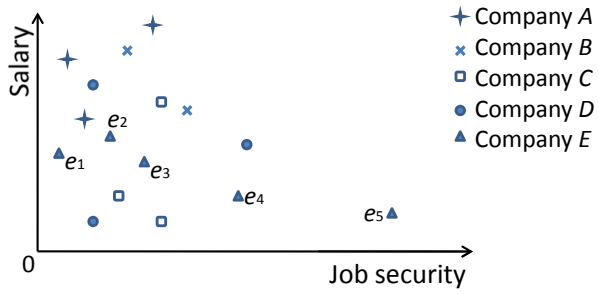

Fig. 1. Companies and job openings.

customized valuation of objects according to different sets of valuation functions, some of which may have been unforeseen at the time the instance probabilities were first computed (there is no need to recompute them when a new valuation function is used).

Example 2: *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 Fig. 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 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, which can be estimated using past statistics on success of applications. 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 [Pei et al. 2007] 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 [Pei et al. 2007], 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.

Table I. Summary of time and space complexities when d is fixed.

	Time	Space	Query Time
Naïve offline algorithm	$O(n^2)$	$O(n)$	
Offline algorithm in [Atallah and Qi 2009]	$\tilde{O}(n^{2-\frac{2}{d+1}})$	$\tilde{O}(n)$	
Our improved offline algorithm	$\tilde{O}(n^{2-\frac{1}{d}})$	$\tilde{O}(n)$	
Naïve online algorithm	$O(n)$	$O(n)$	$O(n)$
Our online algorithm	$\tilde{O}(n^{2-\frac{1}{d}})$	$\tilde{O}(n^{2-\frac{1}{d}})$	$\tilde{O}(n^{1-\frac{1}{d}})$

Example 3: *In Fig. 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 most of its jobs are dominated by multiple jobs from other companies, hence could have been discarded if we had followed the thresholding approach in [Pei et al. 2007]. 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 Examples 1 and 2, 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 [Beskales et al. 2008]. Moreover, the performance of the heuristic pruning methods (top-down, bottom-up) in [Pei et al. 2007] 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 assumption in [Pei et al. 2007] that instance probabilities of the same object always add up to 1, significantly simplifies the problem by enabling the pruning of all objects that are completely dominated by at least one object – since the existence of the dominating object is certain, the object being dominated is guaranteed to have a zero skyline probability and hence can be discarded right away. However, this will not be the case if an object's presence can be uncertain (i.e., the instance probabilities sum up to $x \leq 1$) – even if an object is completely dominated by another, the former can still have a non-zero skyline probability as long as the latter is not present (with probability $1 - x$).

The problems with the thresholding approach have motivated the study of new probabilistic skyline analysis: computing skyline probabilities of all instances [Atallah and Qi 2009]. The outcome of such analysis is useful to all users in their decision making, despite their different utilities. In this paper, we further study the problem of computing all skyline probabilities, and then extend it to the online case. Our contributions are summarized in the section below.

1.2. Our Contributions

An Improved Sub-Quadratic Algorithm for Computing All Skyline Probabilities: While the algorithm for computing all skyline probabilities in [Atallah and Qi 2009] has a worst-case complexity $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$ for objects of dimension $d = 2$, the algorithm we present in this paper has a worst-case time complexity that is $O(n\sqrt{n}\log n)$ for $d = 2$, and more generally $\tilde{O}(n^{2-\frac{1}{d}})$ 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 improved algorithm for computing skyline probabilities is based on the partitioning technique introduced in [Atallah and Qi 2009]: uncertain objects are partitioned into frequent and infrequent objects, which are handled differently.

Online Probabilistic Skyline Analysis: While the user utility is largely ignored in [Pei et al. 2007] (see the discussion in Section 1.1), we study the general probabilistic skyline analysis proposed

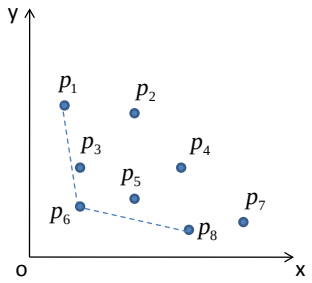

Fig. 2. Skyline computation without uncertainty.

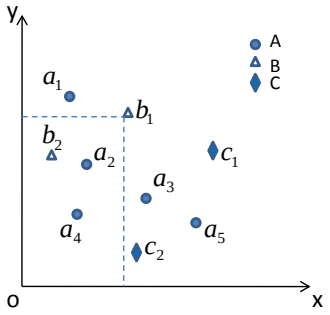

Fig. 3. Skyline computation with uncertainty.

in [Atallah and Qi 2009] that takes into account different user utilities without any restriction. We further extend the problem to the online case: Given an arbitrary query point that is unknown till the query time, compute the probability that the query point is not dominated by any instance of a given set. To distinguish the two problems, we call the first one the *offline problem* and the second one the *online problem*. We propose an algorithm that solves this online problem in $O(\sqrt{n}\log n)$ per query time for $d = 2$ using $O(\sqrt{n}\log n)$ time and space, where n is the size of the given set (i.e., the total number of instances). For d-dimensional data, the query time is $\tilde{O}(n^{1-\frac{1}{d}})$ after $\tilde{O}(n^{2-\frac{1}{d}})$ time and space preprocessing. Our query time is sub-linear at the worst case, while a naïve algorithm will require linear time to process it. The time and space complexities of the above algorithms are summarized in Table I.

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 review two basic algorithms as well as the sub-quadratic algorithm proposed in [Atallah and Qi 2009] in Section 3 and Section 4. We present our improved algorithm for the offline problem in Section 5 and the algorithm for the online problem in Section 6. The related work is given in Section 7. Finally, we conclude our paper in Section 8.

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 $\exists l \in [1,d], p_i.\mathcal{D}_l < p_j.\mathcal{D}_l$. We will use $p \prec q$ to denote that p dominates q, and $p \not\prec q$ to denote that p does not dominate q. Let $p = q$ denote that p and q have the same coordinates in all d dimensions. The notation $p \preceq q$ is used to denote that either $p \prec q$ or $p = q$.

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 4: *In Fig. 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

Our uncertain data model is similar to that in [Pei et al. 2007], 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 [Pei et al. 2007], in our model, instances of the

Table II. Instance probabilities in Fig. 3.

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

same object can have different probabilities and the sum of probabilities over all instances of an object can be less than 1 – i.e., the object's presence can be uncertain. This is consistent with the x-tuple model mentioned in Section 1 where tuple uncertainty exists [Benjelloun et al. 2006]. 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".

An instance p of an object U 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 U 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.

In the following, we formally define the probability spaces that capture the above model. Assume that there are m objects $U_1, U_2, \cdots, U_m$, where U_i is an uncertain object representing a set of instances. We define the probability space for object U_i as $(\Omega_i, \mathscr{F}_i, \Pr_i)$. The sample space Ω_i is $U_i \cup \{\perp_i\}$, where $\perp_i$ is the case of "non-presence". The event set $\mathscr{F}_i$ is 2^{Ω_i}. The probability of an outcome $p \in \Omega_i$ is measured by $\Pr_i(p)$. Note that $\Pr_i(\perp_i)$ can be 0, and in such a case, the probabilities of all instances of the object sum up to 1.

Define the probability space for the uncertain skyline problem to be $(\Omega, \mathscr{F}, \Pr)$. The sample space is the Cartesian product of $\Omega_1, \Omega_2, \cdots$, and Ω_m. The event set $\mathscr{F}$ is 2^{Ω}. For any outcome $(p_1, p_2, \cdots, p_m) \in \Omega$, the probability of the outcome is measured by

$$\Pr(p_1, p_2, \cdots, p_m) = \Pr_1(p_1) \cdot \Pr_2(p_2) \cdot \cdots \cdot \Pr_m(p_m).$$

The following set is the event that an instance p of object U_i is a skyline point:

$$\left\{(p_1, p_2, \cdots, p_m) \;\middle|\; p_i \text{ is } p, \text{ and for any } j \neq i, \text{ either } p_j = \perp_j \text{ or } p_j \not\prec p \text{ holds} \right\}.$$

When the context is clear, we will ignore the index i in the notation $\Pr_i$.

Example 5: *In Fig. 3, we have three uncertain objects A, B, C with multiple instances. The probabilities of instances are listed in Table II. Each object has a skyline probability. For example, B has two instances b_1 and b_2. Instance 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 Offline Problem

The first 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 $U_1, \ldots, U_m$: For each object U_i, a set of n_i instances of that object is given (denote the set by U_i), where the set consists of d-dimensional points with probabilities that add up to at most 1. The sets of instances for different objects are disjoint, but two instances are allowed to have the same coordinates if they are from different objects. We assume that any two instances within the same object can not have the same coordinates. We use S to denote $\cup_{i=1}^{m} U_i$ and n to denote $|S| = \sum_{i=1}^{m} n_i$ where $n_i = |U_i|$. For each point $p \in S$, we use $\Pr(p)$ to denote its instance probability.

Output: For all $p \in S$, the skyline probability of p is defined as follows (assume $p \in U_j$ where $1 \leq j \leq m$):

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

where $D_{S,i}(p)$ denotes the set of instances of object U_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 $D_{S,j}(p)$). The existence of p (with probability $\Pr(p)$) already ensures that none of the other instances of U_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 ($p \in U_j$) 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 dominate p exist. We denote the second part as $\beta(p)$, i.e.,

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

Example 6: *In Fig. 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 U_i is defined as the sum of the skyline probabilities of its instances, i.e.

$$\Pr_{\text{sky}}(U_i) \stackrel{\text{def}}{=} \sum_{p \in U_i} \Pr_{\text{sky}}(p).$$

Since we can always compute $\Pr_{\text{sky}}(p)$ from $\beta(p)$ in constant time, we henceforth 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 III for ease of reference.

2.4. The Online Problem

The second problem that we address in this paper naturally extends from the first one: Now instead of computing the skyline probabilities for a fixed set of points (i.e., instances), we want to compute the probability that no instance from the fixed set dominates a point "on the fly", where no query point is known in advance. The input and the output of an algorithm that solves the online problem are as follows:

Input: Same as that for the offline problem except that now we also have an arbitrary query point q in the data space $\mathcal{D}$ that is not part of the input data set S.

Output: For the query point q, the probability that q is not dominated by any instance in S:

$$\prod_{i=1}^{m} (1 - \sum_{p \in D_{S,i}(q)} \Pr(p)). \quad (2)$$

Note that q is simply a point in $\mathcal{D}$ – We can treat it as the only instance of an extra online object (i.e., the $(m+1)_{\text{th}}$ object), and it has an instance probability of 1. When it is clear from context, we refer to the above probability in Equation 2 as the *online skyline probability* of query point q, which is different from the skyline probability of an instance in S defined in Equation 1.

3. TWO BASIC ALGORITHMS

In this section, we review the two basic algorithms for computing $\beta(p)$ that were used in the algorithm proposed in [Atallah and Qi 2009], which inspired us in designing new algorithms for the online and the offline problems of probabilistic skylines.

Table III. Summary of notations.

Notation	Meaning
m	number of all uncertain objects
n	number of all instances
d	number of dimensions
U_i	the ith uncertain object; the set of instances of U_i ($n_i = \|U_i\|$)
n_i	number of instances of U_i
S	the set of all instances ($n = \|S\|$)
p	point/instance in S
$\Pr_{\text{sky}}(\cdot)$	skyline probability
$D_{S,i}(p)$	instances of U_i in S that dominate p
$D_S(p)$	instances of non-U_j objects in S that dominate p, assuming that $p \in U_j$
$\sigma_i(p)$	sum of probabilities of U_i's instances that dominate p
$\beta(p)$	the probability that p is not dominated by any instance of other object
F	the set of frequent objects
$\overline{F}$	the set of infrequent objects
G_i	a group of infrequent objects
I_i	all the instances in G_i
μ	the cutoff value for determining if an object is frequent or infrequent
$\alpha(p)$	the probability that p is not dominated by any instance of other frequent object (effect of frequent objects on p)
$\gamma(p)$	the probability that p is not dominated by any instance of other infrequent object (effect of infrequent objects on p)
$\gamma_i(p)$	the probability that p is not dominated by any instance of other infrequent object in group G_i (note that p does not need to be an instance of an object in G_i)
$\gamma'_i(p)$	for an instance outside I_i, the probability that no instance $q \in I_i$ exists such that $q \preceq p$

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 the original instances. We therefore consider the latter as *"virtual"* instances of a non-existent object U_0 and assign to each a probability of zero.

The grid can be constructed by sorting all the x-coordinates of the instances in S to obtain the sorted C_x, and sorting all the y-coordinates of the instances in S to obtain the sorted C_y. The sorting takes $O(n \log n)$ time. After that, we can allocate a 2D array to explicitly represent the grid, and put the instances into the corresponding array entries, where the position of each instance in the array (i.e., the grid) can be obtained by two binary searches on C_x and C_y. So the total time to construct the grid is $O(n^2)$, dominated by the explicit array allocation.

Example 7: *Fig. 4 shows such a grid with five instances $p_1, ..., p_5$ (from three different objects) 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.

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

$$\sigma_i(p) \stackrel{\text{def}}{=} \sum_{p' \in D_{S,i}(p)} \Pr(p'), \tag{3}$$

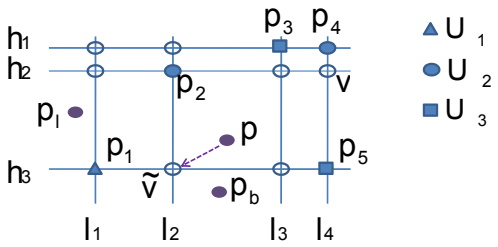

Fig. 4. Space partitioning using a grid.

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

$$\beta(p) = \prod_{i=1, i \neq j}^{m} (1 - \sigma_i(p)). \tag{4}$$

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

$$\sigma_i^*(p) \stackrel{\text{def}}{=} \sum_{p' \in U_i, p' <_h p} \Pr(p'). \tag{5}$$

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 $\hat{p} \in U_j$, add $\Pr(\hat{p})$ to $\sigma_j^*(p)$.
The above takes $O(m)$ time 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 8: *In Fig. 4, p_1 is an instance of U_1 with probability 0.8, p_2 and p_4 are instances of U_2 with probability 0.5 each, p_3 and p_5 are instances of U_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:

$$\sigma_i(p) = \sigma_i^*(p) + \sigma_i(p') + \begin{cases} \Pr(p'), & \text{if } p' \in U_i; \\ 0, & \text{otherwise;} \end{cases} \tag{6}$$

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 9: *To compute $\sigma_i(p_4)$'s from $\sigma_i^*(p_4)$'s computed in Example 8, we follow Equation 6 (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 3.1. *For any vertex p on the grid, $\sigma_i(p)$ computed by the grid method is the sum of probabilities of instances of U_i that dominate p, i.e. Equation 3 can be deduced from Equation 5 and 6.*

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 $O(mn^2)$ time 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, plays a critical part in the sub-quadratic time solution [Atallah and Qi 2009] that we will review 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 ([Bentley 1980; Preparata and Shamos 1985; Mehlhorn 1984]). 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 U_i, we assign the n_i instances of object U_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 $p \in S$, the m sums $\sum_{p' \in D_{S,i}(p)} \Pr(p')$, i.e., $\sigma_i(p)$'s (for $i = 1, \ldots, m$). From these, it is easy to use an additional $O(m)$ time per instance p in S to compute

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

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 $O(n^2(\log n)^{d-1})$. The algorithm in Section 4 (developed in [Atallah and Qi 2009]) improved this worst-case time complexity down to $\tilde{O}(n^{2-\frac{1}{d+1}})$. In this work, we further improve the worst-case time complexity down to $\tilde{O}(n^{2-\frac{1}{d}})$.

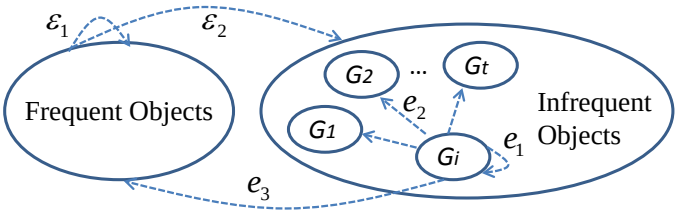

Fig. 5. Schema of the preliminary algorithm.

4. A PRELIMINARY OFFLINE ALGORITHM

4.1. Overview

Before presenting the preliminary sub-quadratic algorithm for computing all skyline probabilities in 2D and later its extension to high dimensions [Atallah and Qi 2009], we first give an intuitive overview on how this scheme works.

The 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 the algorithm is illustrated in Fig. 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 algorithm is based on a "frequent/infrequent" partition approach similar to the generalized string matching algorithm developed by Abrahamson [Abrahamson 1987]. We describe the main ideas of the preliminary algorithm as follows:

(1) 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 $O(n \log^{d-1} n)$ using the WDC algorithm. This is captured by ε_1 and ε_2 in Fig. 5.
(2) 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 Fig. 5, the set of infrequent objects is divided into t groups: $G_1, \ldots, G_t$, where $t = O(n/\mu)$.
(3) Use the grid method within each group of the infrequent objects. It is crucial here that we compute the effect of the group to virtual instances on the constructed grid as well as original instances of the group. Although this takes μ^d time, 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 Fig. 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.

The reason to do the "frequent/infrequent" partition and handle their effects differently is because:

(1) The number of frequent objects is $O(n/\mu)$, which is small and makes it affordable to run the WDC algorithm $O(n/\mu)$ times to compute their effects. The total time here is $O(n/\mu \cdot n \log^{d-1} n)$, so as long as $\mu = \omega(\log^{d-1})$, the time for this part will be $o(n^2)$.
(2) The number of infrequent groups is small (bounded by $O(n/\mu)$), and the dominating running time is to preprocess each group using the grid method in μ^{d+1} time. The total time here is $O(n\mu^d)$, therefore as long as $\mu = o(n^{1/d})$, the time for this part will be $o(n^2)$.

To balance the two running times, one can choose $\mu = (n \log^{d-1} n)^{\frac{1}{d+1}}$, and then the total running time will be $O(n^{2-\frac{1}{d+1}})(\log n)^{\frac{d(d-1)}{d+1}}$, which is $\tilde{O}(n^{2-\frac{1}{d+1}})$.

The case $d = 1$ is trivial to solve in $O(n \log n)$ time by a sweep line method as follows: First, sort the instances in $O(n \log n)$ time. Then scan through the sorted list to maintain $\sigma_i(p)$ for $1 \leq i \leq m$ where p is the current scanning instance. Although there are m such $\sigma_i(p)$'s at the current instance, but only $\sigma_j(p)$ will be updated assuming that $p' \in U_j$ where p' is the previous scanned instance. According to Equation 4, we can obtain $\beta(p)$ by $\beta(p') \cdot (1-\sigma_j(p))/(1-\sigma_j(p'))$. So each update can be implemented in $O(1)$ time, hence the total running time is bounded by the $O(n \log n)$ sorting.

We will use the case $d = 2$ to present the preliminary algorithm, and later extend it to the case $d > 2$.

4.2. Preliminary Algorithm for Two Dimensions

Algorithm 1 shows the preliminary 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.

Algorithm 1 Computing Skyline Probabilities

Input: a set S of n instances from m uncertain objects
Output: all instances with skyline probabilities

RESULT $\leftarrow \emptyset$
Partition objects into two sets: F, $\overline{F}$ // Section 4.2.1
for each U_i in F **do** // Section 4.2.2
  Obtain $\sigma_i(p)$ for all $p \in S$ by calling WDC(S)
end for
for each $p \in S$ (assume $p \in U_j$) **do**
  for each U_i in F and $U_i \neq U_j$ **do**
    $\alpha(p) \leftarrow \prod_{U_i \in F}(1 - \sigma_i(p))$
  end for
end for
Partition $\overline{F}$ into groups: $G_1, \ldots, G_t$ // Section 4.2.3
for each G_i in $\overline{F}$ **do** //
  Obtain $\gamma_i(p)$ and $\gamma'_i(p)$ for all $p \in I_i$ by calling Grid(I_i)
  for each $p \notin I_i$ **do**
    Locate p in a grid cell C of G_i. Let p' be the bottom-left corner of C.
    $\gamma_i(p) \leftarrow \begin{cases} \gamma_i(p'), & p' = p; \\ \gamma'_i(p'), & p' \prec p. \end{cases}$
  end for
end for
for each $p \in S$ **do**
  $\Pr_{\text{sky}}(p) \leftarrow \Pr(p) \cdot \alpha(p) \cdot \prod_{i=1}^{t} \gamma_i(p)$
  RESULT $\leftarrow$ RESULT $\cup (p, \Pr_{\text{sky}}(p))$ // add the pair
end for
return RESULT

4.2.1. Partitioning Objects. Partition the m objects into two sets: a set of *frequent* objects (F), consisting of every object U_i for which $n_i > (n \log n)^{\frac{1}{3}}$ (i.e., $\mu = (n \log n)^{\frac{1}{3}}$); 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 Algorithm 1 shows how to handle frequent objects. For every frequent object U_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 U_j. We compute the quantity

$$\alpha(p) \stackrel{\text{def}}{=} \prod_{U_i \in F, U_i \neq U_j} (1 - \sigma_i(p)) \tag{7}$$

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

4.2.3. Handling Infrequent Objects. Line 11 to 18 in Algorithm 1 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 $n/\mu = n/(n \log n)^{\frac{1}{3}}$. We denote these groups as $G_1, \ldots, G_t$. We use m_i to denote $|G_i|$ (i.e., 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 $|I_i| = \sum_{U_j \in G_i} n_j$).

The next step computes, for each group G_i, the effect of the group to every $p \in S$ (assume that p belongs to object U_k):

$$\gamma_i(p) \stackrel{\text{def}}{=} \prod_{U_j \in G_i, U_j \neq U_k} (1 - \sum_{p' \in D_{I_i,j}(p)} \Pr(p')).$$

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:

First, we use the grid method on G_i. This gives $\gamma_i(p)$ for every grid vertex p, which is the effect of G_i on its own instances (i.e., e_1 in Fig. 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 Fig. 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 have the following important property:

$$\gamma_i(p) = \begin{cases} \gamma_i(p'), & p' = p; \\ \gamma_i'(p'), & p' \prec p; \end{cases}$$

where p' is the bottom-left corner of the grid cell containing p, and

$$\gamma_i'(p') \stackrel{\text{def}}{=} \prod_{U_j \in G_i} \left(1 - \sum_{p'' \in U_j \text{ and } p'' \preceq p'} \Pr(p'')\right).$$

The γ_i''s in the grid can be computed by modifying the grid method accordingly, and the running time is the same as computing γ_i's. Note that if such p' cannot be found (as illustrated in Example 10) for p, then $\gamma_i(p) = 1$. 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 10: *Assume that all instances $p_1, \ldots, p_5$ in Fig. 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 $\gamma_i(p) = \gamma'_i(\tilde{v})$ 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 $\gamma_i(p_l) = \gamma_i(p_b) = 1$.*

4.2.4. Computing Skyline Probabilities. For every p in S, compute the desired $\Pr_{\text{sky}}(p)$ as

$$\Pr_{\text{sky}}(p) = \Pr(p) \cdot \alpha(p) \cdot \prod_{i=1}^{t} \gamma_i(p). \tag{8}$$

Equation 8 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 the preliminary algorithm in Algorithm 1 for two-dimensional uncertain objects.

Partitioning objects (line 2 in Algorithm 1) 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 the 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 $\gamma_i(p')$ and $\gamma'_i(p')$ 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 Algorithm 1.

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

4.3. Extending the Preliminary Algorithm to 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.

Algorithm 1 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. The total time complexity for computing all skyline probabilities is $\tilde{O}(n^{2-\frac{1}{d+1}})$ [Atallah and Qi 2009].

4.3.1. Partitioning and Grouping. Partition the m object into two sets: a set of *frequent* objects (F), consisting of every object U_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 γ_i or γ_i' value of the bottom-left vertex of the cell as p's own γ_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 Algorithm 1 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. IMPROVED OFFLINE ALGORITHM

In this section, we propose a new algorithm for computing all skyline probabilities (the offline problem) that improves the algorithm presented in Section 4 by reducing the time complexity from $O(n^{\frac{5}{3}}(\log n)^{\frac{2}{3}})$ to $O(n^{\frac{3}{2}} \log n)$ in the 2D case. The performance gain comes from a different way to compute the effects of infrequent objects, which is more efficient than the way in Section 4. Such a better infrequent object handling method allows us to increase the cutoff value μ from $(n \log n)^{1/3}$ to $\sqrt{n}$, which reduces the time complexity of computing the effects of frequent objects, because there are less frequent objects when the cutoff value raises. Note that the improved offline algorithm handles the frequent objects in the same way as in Section 4.

5.1. Overview and Preliminaries

As in the preliminary algorithm of Section 4, we partition all uncertain objects into two categories: frequent objects and infrequent objects. The cutoff value μ is now different: We define objects with at least $\sqrt{n}$ instances as frequent objects and the rest as infrequent objects (i.e., $\mu = \sqrt{n}$).

For an instance $p \in U_j$, we compute the quantity:

$$\gamma(p) \stackrel{\text{def}}{=} \prod_{U_i \in \overline{F}, U_i \neq U_j} (1 - \sigma_i(p)) \tag{9}$$

as the effect of infrequent objects on instance p.

We have defined earlier the effect of frequent objects on instance p as $\alpha(p)$ in Equation 7. With $\alpha(p)$ and $\gamma(p)$, the skyline probability of an instance p can be easily computed as:

$$\Pr_{\text{sky}}(p) = \Pr(p) \cdot \alpha(p) \cdot \gamma(p).$$

The effect of frequent objects on all instances (i.e., $\alpha(p)$ for all $p \in S$) can be computed in $O(n\sqrt{n}\log n)$ time with $O(n)$ space. This is achieved by the WDC algorithm as we did in Section 4.2.2 in $O(n\sqrt{n}\log n)$ time, because there are only $O(\sqrt{n})$ frequent objects and each WDC takes $O(n \log n)$ time. The space complexity is $O(n)$.

The effect of infrequent objects on all instances (i.e., $\gamma(p)$ for all $p \in S$) can be computed in $O(n\sqrt{n}\log n)$ time with $O(n)$ space using the *plane sweep* algorithm that will be described in Section 5.2. The sweep line algorithm requires a data structure T_y that satisfies the following properties:

— Instances in T_y are sorted according to their y-coordinates.
— Each instance p in T_y is associated with two quantities: w and σ, where w is a weight and σ is a sum of probabilities.
— Retrieving w and retrieving σ of an instance p both take $O(\log n)$ time.
— Inserting a new instance to T_y takes $O(\log n)$ time.
— Group-updating of the w's of instances whose y-coordinates are within a range takes $O(\log n)$ time. Specifically, the operation is called a *range weight update*, denoted as $\text{RWU}(a, b, c)$, with the intended effect that each instance p whose y-coordinate is between a and b ($a \leq b$) has its w multiplied by c.

A linear-space data structure T_y that achieves the above can be constructed as follows: First, sort all the input instances according to their y-coordinates. This is possible because the instances are given offline. Then construct a binary tree on top of the sorted instances. Each leaf node of the tree corresponds to an instance. Each internal node of the tree stores the range of y-coordinates of the instances of all the leaf descendants of that internal node. Each node v will maintain a value $\eta(v)$. The η's are used to compute the w quantities. The way η is used and updated is described below:

— To retrieve the w quantity of an instance p, find the path from the root to the leaf that corresponds to p in $O(\log n)$ time, and then the w quantity is the multiplication result of all the η values associated with the nodes (including the leaf) on that path.
— To insert an instance p with quantity w, first find the path from the root to the leaf that corresponds to p in $O(\log n)$ time, and denote the path by $v_1, v_2, \cdots, v_k$ where k is the length of the path. Then set the η value of the leaf v_k to be

$$w / \prod_{1 \leq i \leq k-1} \eta(v_i).$$

— To perform $\text{RWU}(a, b, c)$, locate the $O(\log n)$ tree nodes whose subtrees are disjoint and covering exactly all the instances with y-coordinates between a and b, then update the η values of those $O(\log n)$ tree nodes by multiplying c.

5.2. Computing the Effects of Infrequent Objects

To simplify the presentation, we assume that all instances have distinct y-coordinates, regardless of which object (frequent or infrequent) it is from.

Imagine moving an infinite vertical line (called the *sweep line*) from left to right across the plane, beginning at the leftmost instance of any uncertain object. As the sweep line moves, we maintain the instances from infrequent objects that we have seen so far in a dynamic data structure T_y that was described in 5.1. During the plane sweep, whenever the sweep line encounters an instance p, we compute the effect of infrequent objects on p (i.e., $\gamma(p)$) as follows: If p is an instance from a frequent object, we simply query T_y for the relevant w and σ (whose definition will be given below), from which we can compute $\gamma(p)$; otherwise, p belongs to an infrequent object, we need to insert p

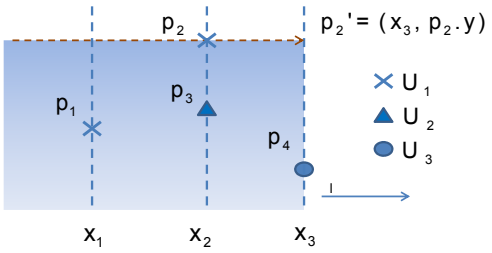

Fig. 6. Example on Computing $w(p_2.y)$.

into T_y and update T_y. When there are more than one instances on the sweep line, we process the instances ordered by their x-coordinates.

We compute the γ's for all instances by initializing T_y to be empty, then moving the sweep line l from the leftmost instance in S to the rightmost instance in S. For every position x of l, we maintain in T_y the infrequent-object instances that have been encountered by l during its sweep so far, with the weight $w(p.y)$ of such an instance p in T_y being:

$$w(p.y) \stackrel{\text{def}}{=} \prod_{U_i \in \overline{F}} (1 - \sigma_l^i(p.y))$$

where $p.y$ is the y-coordinate of instance p and $\sigma_l^i(p.y)$ is a 1-dimensional version of $\sigma_i(p)$ for the horizontal projections on the current sweep line l of the instances of U_i already encountered by l. In other words, $\sigma_l^i(y)$ is the sum of the probabilities of the instances of U_i that are either at or to the left of the current position of l and also have a smaller y-coordinate than that of instance p. Recall that in the previous section, we mentioned that an instance p in T_y has two quantities: w and σ. We now define w to be $w(p.y)$ and σ to be $\sigma_l^i(p.y)$ (assuming $p \in U_i$). Note that if instance p belongs to a frequent object and is (geometrically) on line l then $w(p.y) = \gamma(p)$, but as l moves past p and sweeps through other instances, $w(p.y)$ changes and ceases to be $\gamma(p)$. If p belongs to an infrequent object, the relationship between $w(p.y)$ and $\gamma(p)$ is not as simple. We will see why this is the case in Section 5.2.1.

Example 11: *In Fig. 6, we have 4 instances from 3 uncertain objects U_1 to U_3. U_2 and U_3 are infrequent objects while U_1 is frequent. Suppose the current sweep line l is at x_2. $\gamma(p_2) = 1 - \Pr(p_3)$, since the only instance from an infrequent object that dominates p_2 is p_3. As the sweep line moves from x_2 to x_3, $w(p_2.y)$ changes as follows:*

— *When l is at x_2: $w(p_2.y) = 1 - \Pr(p_3) = \gamma(p_2)$;*
— *When l is at x_3: $w(p_2.y) = (1 - \Pr(p_3))(1 - \Pr(p_4)) < \gamma(p_2)$.*

The way we compute $w(p.y)$ for any p given the position x of the current sweep line l is that we always horizontally project p to l and get a virtual point $p' = (x, p.y)$. Then we compute the effect of infrequent objects on this virtual point p' without considering the original p. For example, when l is at x_3, p_2' is dominated by infrequent-object instances p_3 and p_4, hence $w(p_2.y) = (1 - \Pr(p_3))(1 - \Pr(p_4))$. The shaded region in Fig. 6 indicates the region that needs to be examined when computing $w(p_2.y)$ for l at position x_3.

We next explain in detail what is done with T_y when the left-to-right sweep encounters an instance p of an object U_j (that could be either frequent or infrequent). Let the current sweep line be l.

5.2.1. Computing $\gamma(p)$. We distinguish two cases when computing $\gamma(p)$, where p is an instance of U_j:

Case 1: $U_j \in F$, i.e., p is an instance from a frequent object.

We first search for p's predecessor in T_y under the order of y-coordinates. This can be done in $O(\log n)$ time by using an auxiliary dictionary data structure to maintain the inserted instances, where y-coordinates are the search keys. Denote the predecessor by p'. Assume that p' is from object U_k. Note that $U_k \in \overline{F}$, because T_y only stores infrequent objects. Retrieve both $w(p'.y)$ and $\sigma_l^k(p'.y)$ from T_y in $O(\log n)$ time, then we can compute

$$\gamma(p) = w(p'.y) * (1 - \sigma_l^k(p'.y) - \Pr(p'))/(1 - \sigma_l^k(p'.y)). \tag{10}$$

This is because, after the projection, the instances from infrequent objects that dominate p are exactly those that dominate p', plus p' itself. In Equation 10, the division by $1 - \sigma_l^k(p'.y)$ represents the cancelation of U_k's effect and the multiplication by $1 - \sigma_l^k(p'.y) - \Pr(p')$ represents adding U_k's new effect on p after taking p' into account (where $p' \in U_k$). The total time needed to compute $\gamma(p)$ for a frequent-object instance p is just $O(\log n)$.

Case 2: $U_j \in \overline{F}$, i.e., p is an instance from an infrequent object.

We need to be more careful in computing $\gamma(p)$ in this case. First of all, $w(p'.y)$ might contain the effect from instances of U_j itself, which should be removed in computing $\gamma(p)$ (recall that in Equation 9, the instances from the same object are not considered), i.e., we should first divide $w(p'.y)$ by $1 - \sigma_l^j(p)$ to cancel the effect of instances from U_j. Then we check if $U_k = U_j$: If so, we are done; otherwise, we go on to incorporate the effect of p'.

As an example, suppose that the sweep line l in Fig. 6 now moves past the position x_3 and encounters a new instance p whose y-coordinate is between p_1 and p_4. The predecessor of p in T_y is hence p_4 (note that $p_4 \in U_3$, where U_3 is an infrequent object). If $p \notin U_3$, then $\gamma(p) = 1 - \Pr(p_4)$; otherwise, $\gamma(p) = 1$.

5.2.2. Updating T_y. If U_j is frequent, we do not need to update T_y, as T_y only contains instances from infrequent objects encountered during the sweep. However, if U_j is infrequent, we need to update T_y to reflect the change after seeing $p \in U_j$ at the current sweep line. This update is done as follows:

Let p'' be the nearest predecessor of p from the same U_j as p; this p'' can be obtained from an auxiliary dynamic dictionary structure that maintains the instances of U_j already encountered by the sweep line l, sorted by their y-coordinates. Such a step takes $O(\log \sqrt{n})$ query time, as U_j is an infrequent object with the number of instances no more than $\sqrt{n}$.

We insert instance p into T_y with $w(p.y) = \gamma(p)$ and $\sigma_l^j(p.y) = \sigma_l^j(p''.y) + \Pr(p'')$. Insertion takes $O(\log n)$ time, so does retrieving $\sigma_l^j(p''.y)$ from T_y.

Now we need to update $w(\hat{p}.y)$ for every instance $\hat{p}$ in T_y with greater y coordinate than that of p. The weight of each of such instances $\hat{p}$ has to be updated to reflect the effect of the newly inserted p that $\hat{p}$ dominates. There are two cases that we need to consider:

Case 1: $\hat{p} \in U_j$ (where $p, \hat{p}$ are from the same object):

We do the following updates

$$w(\hat{p}.y) \leftarrow w(\hat{p}.y) * (1 - \sigma_l^j(\hat{p}.y) - \Pr(p))/(1 - \sigma_l^j(\hat{p}.y)),$$
$$\sigma_l^j(\hat{p}.y) \leftarrow \sigma_l^j(\hat{p}.y) + \Pr(p).$$

Note that this step takes $O(n_j \log n)$ time, which is $O(\sqrt{n} \log n)$ because U_j is infrequent.

Case 2: $\hat{p} \notin U_j$ ($p, \hat{p}$ are from different objects):

Although there can be $O(n)$ such non-U_j instances to be updated, this massive update can be done in $O(n_j \log n)$ time (i.e., in $O(\sqrt{n} \log n)$ time) because these non-U_j instances can be partitioned into $O(n_j)$ contiguous groups separated by instances from U_j: We update the weights of each such non-U_j group in logarithmic time using $\mathrm{RWU}(a, b, c)$ where a (resp., b) is the y coordinate of the instance p_u (p_l) of U_j that is at the upper (lower) boundary of that contiguous group of non-U_j instances, and

$$c = (1 - \sigma_l^j(p_l.y))/(1 - \sigma_l^j(p_l.y) + \Pr(p)).$$

Note that for non-U_j instances, the old effect of instances from U_j is $1 - \sigma_l^j(p_l.y) + \Pr(p)$ and the new effect is $1 - \sigma_l^j(p_l.y)$. This is because $p_l \in U_j$ and its $\sigma_l^j(p_l.y)$ has already been updated to include the effect of the newly-inserted p in Case 1.

The time for processing an instance encountered by the sweep line l is $O(\sqrt{n} \log n)$, and therefore the total time is $O(n\sqrt{n} \log n)$.

5.3. High Dimensional Cases

The extension to d-dimensional case is based on the following idea (assuming d is a fixed constant): Assume that the cutoff value to do the frequent/infrequent partition is μ. The effects of the frequent objects can be done using WDC (like we did in the 2D case), in $O(\frac{n}{\mu} \cdot n \log^{d-1} n)$ time. For the effects of infrequent objects, the algorithm uses a $(d-1)$-dimensional hyperplane to sweep through the instances one by one according to their sorted order. The sorted order is obtained by sorting the instances, where the k-th coordinate serves as the k-th most important key, i.e., the sort order is lexicographic. During the sweeping, the algorithm maintains the w and σ_l^i quantities like our 2D case, using an $O(n \log^{d-2} n)$-space data structure that support insertion, retrieval and range weight update in $O(\log^{d-1} n)$ time. Each time an instance is inserted into the data structure, it needs to do μ^{d-1} range weight updates. The dominating time comes from these range weight updates, so the total time complexity during the sweep is $O(n\mu^{d-1} \log^{d-1} n)$ time.

Based on the above analysis, the total space complexity is $O(n \log^{d-2} n)$, and the total time complexity is

$$O(n\mu^{d-1} \log^{d-1} n + \frac{n}{\mu} \cdot n \log^{d-1} n).$$

Choosing the cutoff value to be $\mu = n^{1/d}$ yields Theorem 5.1.

THEOREM 5.1. *Offline skyline probabilities for d-dimensional uncertain data can be computed in $O(n^{2-\frac{1}{d}} \log^{d-1} n)$ time and $O(n \log^{d-2} n)$ space.*

6. THE ONLINE ALGORITHM

In this section, we will present a data structure that can be preprocessed in $O(n\sqrt{n} \log n)$ time and space, so that the *online* skyline probability of a query point can be computed in $O(\sqrt{n} \log n)$ time.

Throughout this section, we assume that the x-coordinates of all the instances in S are distinct; the y-coordinates of all the instances in S are distinct; any online query point does not share the same x-coordinate or y-coordinate with any instance in S. This assumption is done without loss of generality, and dropping it would result in a more cluttered exposition but would not cause any change in our asymptotic complexity bounds.

6.1. Basic Idea

We partition all uncertain objects into two categories: frequent objects and infrequent objects. The cutoff value μ is set to $\sqrt{n}$ as in the algorithm of Section 5.

The effect of frequent objects on a query point q,

$$\hat{\alpha}(q) = \prod_{U_i \in F} (1 - \sigma_i(q)),$$

can be computed in $O(\sum_{U_i \in F} \log n_i) \leq O(\sqrt{n} \log n)$ time if a weighted dominance counting data structure [Willard 1985] is built to compute $\sigma_i(q)$ for each $U_i \in F$. The total time and space for all such data structures are $O(\sum_{U_i \in F} n_i \log n_i) \leq O(n \log n)$.

It remains to show how to efficiently compute the effect of infrequent objects. For a query point $q \in \mathcal{D}$, define $\hat{\gamma}(q)$ to be the effect of infrequent objects on q, i.e.,

$$\hat{\gamma}(q) = \prod_{U_i \in \overline{F}} (1 - \sigma_i(q)). \tag{11}$$

We will show that computing $\hat{\gamma}(q)$ for any point $q \in \mathcal{D}$ takes $O(\log n)$ time with a data structure that can be built in $O(n\sqrt{n} \log n)$ time and space.

The key idea to compute Equation 11 efficiently is to transform the computation of $1 - \sigma_i(q)$ to a range product problem. More specifically, for an object U_i, we will construct a set of dummy points U_i' and assign a dummy value $v(p)$ for each $p \in U_i'$ (the construction algorithm will be given in Section 6.2), such that computing $1 - \sigma_i(q)$ for any online query $q \in \mathcal{D}$ is equivalent to a range product query in U_i' over the dummy values, i.e.

$$1 - \sigma_i(q) = \prod_{p' \in U_i', p' \prec q} v(p'), \tag{12}$$

where $p' \prec q$ means that p' dominates q. We assume that the range product is 1 if there is no point dominating p (i.e., no point in the range).

Under this transformation, Equation 11 becomes a range product in the point set $U_{\overline{F}}' = \bigcup_{U_i \in \overline{F}} U_i'$ over the dummy values, i.e.,

$$\hat{\gamma}(q) = \prod_{U_i \in \overline{F}} \prod_{p' \in U_i', p' \prec q} v(p') = \prod_{p' \in U_{\overline{F}}', p' \prec q} v(p').$$

This range product problem can be solved in $O(\log |U_{\overline{F}}'|)$ time with an $O(|U_{\overline{F}}'| \log |U_{\overline{F}}'|)$ time and space preprocessing using a standard range query data structure [Willard 1985].

We will provide an algorithm in Section 6.2 to construct the point set U_i' and their dummy values. Based on our construction, the following lemma holds:

LEMMA 6.1. *For any $U_i \in \overline{F}$, there exists an algorithm to construct U_i' and the dummy values in $O(n_i^2)$ time and space. The number of dummy points in U_i' is n_i^2. The total number of dummy points in $U_{\overline{F}}'$ is bounded by*

$$\sum_{U_i \in \overline{F}} n_i^2 \leq \sum_{U_i \in \overline{F}} n_i \sqrt{n} = \sqrt{n} \sum_{U_i \in \overline{F}} n_i \leq \sqrt{n} \cdot n.$$

Therefore, the total time and space to build the range product data structure for $U_{\overline{F}}'$ is $O(n\sqrt{n} \log n)$, and a range product query takes $O(\log(n\sqrt{n})) = O(\log n)$ time.

Algorithm 2 and 3 summarize the data structures for computing the online skyline probability. Theorem 6.2 summarizes the above analysis for Algorithm 2 and 3. Note that The data structures can also be used to compute all skyline probabilities in $O(n\sqrt{n} \log n)$ time with slight and straightforward modifications.

THEOREM 6.2. *Online skyline probability for a 2D query point can be computed in $O(\sqrt{n} \log n)$ time with data structures that can be built in $O(n\sqrt{n} \log n)$ time and space.*

Algorithm 2 Preprocessing for Computing Online Skyline Probability

1: **for** $U_i \in F$ **do**
2: Build a data structure for computing $\sigma_i(q)$.
3: **end for**
4: **for each** $U_i \in \overline{F}$ **do**
5: Compute the dummy point set U_i' and their dummy values.
6: **end for**
7: Put together all the dummy points to form $U'_{\overline{F}}$, i.e., $U'_{\overline{F}} = \bigcup_{U_i \in \overline{F}} U_i'$.
The dummy values are unchanged.
8: Build a range product data structure over the dummy values of $U'_{\overline{F}}$.
Ignore those dummy points whose dummy values are 1.

Algorithm 3 Computing Online Skyline Probability

Input: a query point $q \in \mathcal{D}$

1: Initialize $\mathrm{RESULT} \leftarrow 1$.
2: **for each** $U_i \in F$ **do**
3: $\mathrm{RESULT} \leftarrow \mathrm{RESULT} \times (1 - \sigma_i(q))$,
where $\sigma_i(q)$ is computed by a weighted dominance counting query in U_i.
4: **end for**
5: $\mathrm{RESULT} \leftarrow \mathrm{RESULT} \times \hat{\gamma}(q)$,
where $\hat{\gamma}(q)$ is computed by a range product query in $U'_{\overline{F}}$.
6: Return RESULT.

6.2. Dummy Points and Dummy Values

For each infrequent object U_i, construct a grid using a method similar to the one described in Section 3.1 except that only one object is considered here (i.e., no grouping). There are n_i^2 grid points, and these grid points form the set U_i'. So it remains to assign the dummy values to the grid points so that Equation 12 holds. The dummy value assignment scheme is similar to the scheme for extension for the more general ULDB model in [Sarma et al. 2008].

To assign dummy values to the grid points in U_i', we first process the grid to compute $\hat{\sigma}_i(p)$ for each grid point $p \in U_i'$, where

$$\hat{\sigma}_i(p) = \sigma_i(p) + Pr(p).$$

$\Pr(p)$ is the probability of an instance of U_i located in p. Note that $\Pr(p) = 0$ if p does not coincide with any instance of U_i. This processing is straightforward using two "FOR" loops, and it takes $O(n_i^2)$ time.

We then assign the dummy value for a grid point $p \in U_i'$ as follows:

— If p is the bottom-left grid point (the one with the smallest coordinates in all dimensions), the dummy value is set to $1 - \hat{\sigma}_i(p)$.
— If p is on the leftmost vertical line of the grid but not the bottom-left point, we assign value $\frac{1-\hat{\sigma}_i(p)}{1-\hat{\sigma}_i(p_b)}$ to p where p_b is the point immediately below p (see Fig. 7).
— If p is on the bottom horizontal line of the grid but not the bottom-left point, we assign value $\frac{1-\hat{\sigma}_i(p)}{1-\hat{\sigma}_i(p_l)}$ to p where p_l is the point immediately to the left of p.
— Otherwise, the dummy value $v(p)$ can be computed as:

$$v(p) = \frac{(1 - \hat{\sigma}_i(p))(1 - \hat{\sigma}_i(p_{lb}))}{(1 - \hat{\sigma}_i(p_l))(1 - \hat{\sigma}_i(p_b))}$$

where p_{lb} is the point immediately to the left of p_b (or equivalently, the point immediately below p_l).

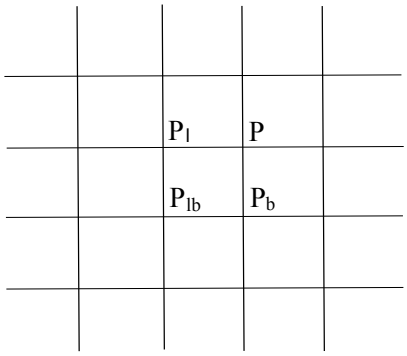

Fig. 7. Illustration of P_b, P_l and P_{lb}.

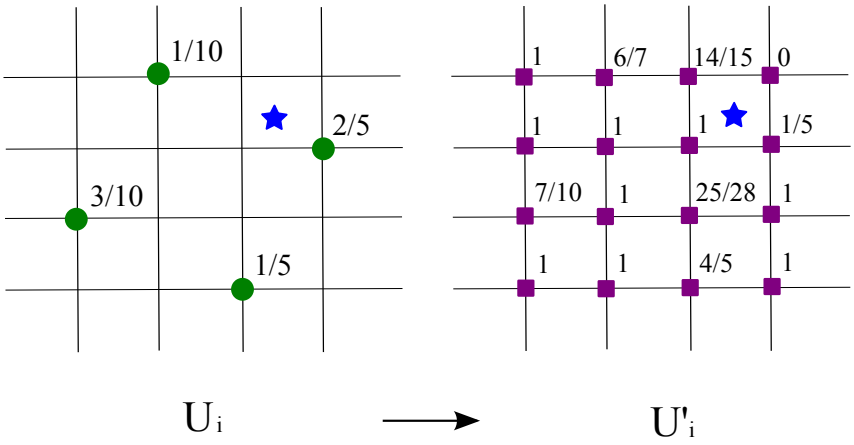

Fig. 8. Illustration of a transformation from U_i to U_i'.

The total time to assign the dummy values is clearly $O(n_i^2)$. Equation 12 holds, because we can verify that by assigning grid points such dummy values, we make sure that the product of dummy values of all grid points that dominate a query point $q \in \mathcal{D}$ is $1 - \sigma_i(q)$.

An example of the above transformation is illustrated in Fig. 8. The circles are the instances of U_i. The squares are the corresponding dummy points in U_i'. If an online query point q is located in the $\star$ position, then $1 - \sigma_i(q) = 1 - (0.3 + 0.2) = 0.5$ which is equal to the product of the dummy values $\frac{4}{5}$, $\frac{7}{10}$, $\frac{25}{28}$ and six 1's.

6.3. High Dimensional Cases

The generalization to d-dimensional cases is straightforward: Algorithm 2 and Algorithm 3 still apply when a d-dimensional data structure is used. We analyze the time and space complexity below.

Assume that the frequent/infrequent cutoff value is μ, and the dimension d is a fixed constant. The d-dimensional range sum data structures in line 2 of Algorithm 2 can be constructed in time $O(n_i \log^{d-1} n_i)$ time and space, such that $\sigma_i(q)$ can be computed in $O(\log^{d-1} n_i)$ time [Willard 1985]. So the total time and space complexity of line 1 to 3 of Algorithm 2 is bounded by the order

of

$$\sum_{U_i \in F} n_i \log^{d-1} n_i \leq \sum_{1 \leq i \leq m} n_i \log^{d-1} n = n \log^{d-1} n.$$

The dummy points at line 5 can be constructed in $O(n_i^d)$ time on a d-dimensional grid. So both the total number of dummy points (i.e., $|U'_{\overline{F}}|$) at line 7 and the total time and space of line 4 to 6 are bounded by the order of

$$\sum_{U_i \in \overline{F}} n_i^d \leq \sum_{U_i \in \overline{F}} n_i \mu^{d-1} = n\mu^{d-1}.$$

The range product data structure in line 8 can be constructed in time and space proportional to

$$|U'_{\overline{F}}| \log^{d-1} |U'_{\overline{F}}| \leq n\mu^{d-1} \log^{d-1}(n\mu^{d-1}) \leq n\mu^{d-1} \log^{d-1}(n^d) = O(n\mu^{d-1} \log^{d-1} n),$$

such that a range product query can be done in $O(\log^{d-1}(n\mu^{d-1})) = O(\log^{d-1} n)$ time [Willard 1985]. Therefore, the total preprocessing time and space of Algorithm 2 is

$$O\left(n \log^{d-1} n + n\mu^{d-1} + n\mu^{d-1} \log^{d-1} n\right) = O(n\mu^{d-1} \log^{d-1} n).$$

At the query stage, the running time for line 2 to 4 of Algorithm 3 is in the order of

$$\sum_{U_i \in \overline{F}} \log^{d-1} n_i \leq \sum_{U_i \in \overline{F}} \log^{d-1} n = \left|\overline{F}\right| \log^{d-1} n \leq \frac{n}{\mu} \log^{d-1} n.$$

The range product query at line 5 is $O(\log^{d-1} n)$. So the total query time is $O(\frac{n}{\mu} \log^{d-1} n)$, dominated by line 2 to 4.

Based on the above analysis, different cutoff value μ's can lead to different preprocess/query time/space trade-offs. If we set $\mu = n^{1/d}$, then we get Theorem 6.3.

THEOREM 6.3. *Given a query point q, its online probability for d-dimensional data can be computed in $O(n^{1-\frac{1}{d}} \log^{d-1} n)$ time with data structures that can be built in $O(n^{2-\frac{1}{d}} \log^{d-1} n)$ time and space.*

7. 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 [Börzsönyi et al. 2001], although efficient algorithms for finding maxima of vectors exist as early as in [Kung et al. 1975] in computational geometry. Various skyline algorithms have been designed for different settings, such as computing skylines in a distributed setting [Vlachou et al. 2007; Zhu et al. 2007], computing subspace skylines [Morse et al. 2007; Pei et al. 2007], maintaining incremental skylines for dynamic data sets [Wu et al. 2007], etc. In addition, many interesting variations of skyline computation have been explored recently, including k most representative skyline points [Lin et al. 2007], reverse skyline queries [Dellis and Seeger 2007; Lian and Chen 2008a] and privacy-preserving skylines [Chen et al. 2007].

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 [Cheng et al. 2004; Tao et al. 2005], probabilistic nearest-neighbor queries [Cheng et al. 2004; Cheng et al. 2008; Beskales et al. 2008], ranking queries [Soliman et al. 2007; Lian and Chen 2008b; Hua et al. 2008] and so on. Most of work assumes some uncertain database models, e.g., attribute-level uncertainty model [Singh et al. 2008; **?**] or tuple-level uncertainty model [Benjelloun et al. 2006; Boulos et al. 2005]. Skyline queries with uncertain data have also been studied recently [Zhang et al. 2009; Pei et al. 2007; Lian and Chen 2008a], among which [Pei et al. 2007] is closest to our work. While [Pei et al. 2007] designed algorithms for computing probabilistic skylines, [Lian and Chen 2008a] further studied the problem of probabilistic reverse skyline computation in both monochromatic and bichromatic cases.

[Zhang et al. 2009], 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. [Atallah and Qi 2009] takes 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 [Pei et al. 2007]. This allows more flexibility for users to utilize the skyline results according to their own utilities rather than focus on the probability alone. [Ljosa and Singh 2008] 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, [Ljosa and Singh 2008] has to recompute the results for each user or whenever a user's utility changes. However, with the approach to the probabilistic skyline analysis in [Atallah and Qi 2009], the skyline probabilities only need to be computed 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.

Notes: Recently, Afshani et al. [Afshani et al. 2011] independently established similar bounds to the ones we give here, and gave fast approximate algorithms

8. CONCLUSIONS

To the best of our knowledge, our paper is the first to asymptotically improve the time complexity of the algorithm proposed in [Atallah and Qi 2009] for computing all skyline probabilities for data with discrete uncertainty. Similar to the algorithm in [Atallah and Qi 2009], the improved algorithm is also based on the partitioning of uncertain objects into frequent and infrequent objects and handling the two categories differently. Our improvement comes from a new approach that can handle the infrequent objects more efficiently than the previous work. We also propose an algorithm for solving the online problem of computing skyline probabilities that has the same time complexity as the improved algorithm for the offline problem, but with a slightly larger space complexity.

This work focuses on the worst-case time complexity to compute all skyline probabilities. It would be interesting to see if further improvement is possible, not only in the worst-case, but also in the average-case when the geometric positions of instances follow some distribution.

No non-trivial lower bound result is known for computing all skyline probabilities, other than the lower bound for computing the skyline points [Kung et al. 1975], which is a special case when each object only has one instance with a probability of one. So there is still a gap between our result and the $\Omega(n \log n)$ lower bound result of Kung et al. [Kung et al. 1975] even in the two-dimensional case. Therefore, one open problem is to derive a better lower bound for the all skyline probabilities problem.

Acknowledgments

We would like to thank the referees for their helpful comments and suggestions.

REFERENCES

ABRAHAMSON, K. 1987. Generalized string matching. *SIAM Journal on Computing 16*, 1039–1051.

AFSHANI, P., AGARWAL, P. K., ARGE, L., LARSEN, K. G., AND PHILLIPS, J. M. 2011. (approximate) uncertain skylines. In *Proceedings of The 14th International Conference on Database Theory*.

ATALLAH, M. J. AND QI, Y. 2009. Computing all skyline probabilities for uncertain data. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. PODS '09. ACM, New York, NY, USA, 279–287.

BENJELLOUN, O., SARMA, A. D., HALEVY, A., AND WIDOM, J. 2006. ULDBs: databases with uncertainty and lineage. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB '06. VLDB Endowment, 953–964.

BENTLEY, J. L. 1980. Multidimensional divide-and-conquer. *Communications of the ACM 23*, 214–229.

BESKALES, G., SOLIMAN, M. A., AND IIYAS, I. F. 2008. Efficient search for the top-k probable nearest neighbors in uncertain databases. *Proceedings of the VLDB Endowment 1*, 326–339.

BÖRZSÖNYI, S., KOSSMANN, D., AND STOCKER, K. 2001. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 421–430.

BOULOS, J., DALVI, N., MANDHANI, B., MATHUR, S., RE, C., AND SUCIU, D. 2005. MYSTIQ: a system for finding more answers by using probabilities. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. SIGMOD '05. ACM, New York, NY, USA, 891–893.

CHEN, B.-C., LEFEVRE, K., AND RAMAKRISHNAN, R. 2007. Privacy skyline: privacy with multidimensional adversarial knowledge. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB '07. VLDB Endowment, 770–781.

CHENG, R., CHEN, J., MOKBEL, M., AND CHOW, C.-Y. 2008. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 973–982.

CHENG, R., KALASHNIKOV, D. V., AND PRABHAKAR, S. 2003. Evaluating probabilistic queries over imprecise data. In *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*.

CHENG, R., KALASHNIKOV, D. V., AND PRABHAKAR, S. 2004. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering 16*, 1112–1127.

CHENG, R., XIA, Y., PRABHAKAR, S., SHAH, R., AND VITTER, J. S. 2004. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*. VLDB '04. VLDB Endowment, 876–887.

CORMODE, G., LI, F., AND YI, K. 2009. Semantics of ranking queries for probabilistic data and expected ranks. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 305–316.

DELLIS, E. AND SEEGER, B. 2007. Efficient computation of reverse skyline queries. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB '07. VLDB Endowment, 291–302.

HUA, M., PEI, J., ZHANG, W., AND LIN, X. 2008. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD '08. ACM, New York, NY, USA, 673–686.

KUNG, H. T., LUCCIO, F., AND PREPARATA, F. P. 1975. On finding the maxima of a set of vectors. *Journal of the ACM 22,* 4, 469–476.

LI, J., SAHA, B., AND DESHPANDE, A. 2009. A unified approach to ranking in probabilistic databases. *Proceedings of the VLDB Endowment 2*, 502–513.

LIAN, X. AND CHEN, L. 2008a. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD '08. ACM, New York, NY, USA, 213–226.

LIAN, X. AND CHEN, L. 2008b. Probabilistic ranked queries in uncertain databases. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. EDBT '08. ACM, New York, NY, USA, 511–522.

LIN, X., YUAN, Y., ZHANG, Q., AND ZHANG, Y. 2007. Selecting stars: The k most representative skyline operator. 86 –95.

LJOSA, V. AND SINGH, A. K. 2008. Top-k spatial joins of probabilistic objects. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 566–575.

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

MORSE, M., PATEL, J. M., AND JAGADISH, H. V. 2007. Efficient skyline computation over low-cardinality domains. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB '07. VLDB Endowment, 267–278.

PEI, J., FU, A. W.-C., LIN, X., AND WANG, H. 2007. Computing compressed multidimensional skyline cubes efficiently. 96 –105.

PEI, J., JIANG, B., LIN, X., AND YUAN, Y. 2007. Probabilistic skylines on uncertain data. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB '07. VLDB Endowment, 15–26.

PREPARATA, F. AND SHAMOS, M. 1985. *Computational Geometry: An Introduction*. Springer-Verlag.

SARMA, A. D., THEOBALD, M., AND WIDOM, J. 2008. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 1023–1032.

SINGH, S., MAYFIELD, C., SHAH, R., PRABHAKAR, S., HAMBRUSCH, S., NEVILLE, J., AND CHENG, R. 2008. Database support for probabilistic attributes and tuples. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 1053–1061.

SOLIMAN, M. A., ILYAS, I. F., AND CHANG, K. C.-C. 2007. Urank: formulation and efficient evaluation of top-k queries in uncertain databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. SIGMOD '07. ACM, New York, NY, USA, 1082–1084.

TAO, Y., CHENG, R., XIAO, X., NGAI, W., KAO, B., AND PRABHAKAR, S. 2005. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*.

VLACHOU, A., DOULKERIDIS, C., KOTIDIS, Y., AND VAZIRGIANNIS, M. 2007. SKYPEER: Efficient subspace skyline computation over distributed data. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. 416 –425.

WILLARD, D. E. 1985. New data structures for orthogonal range queries. *SIAM Journal on Computing 14*, 232–253.

WU, P., AGRAWAL, D., EGECIOGLU, O., AND EL ABBADI, A. 2007. DeltaSky: Optimal maintenance of skyline deletions without exclusive dominance region generation. 486 –495.

ZHANG, W., LIN, X., ZHANG, Y., WANG, W., AND YU, J. X. 2009. Probabilistic skyline operator over sliding windows. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 1060–1071.

ZHANG, X. AND CHOMICKI, J. Semantics and evaluation of top-k queries in probabilistic databases. *Distributed and Parallel Databases 26*, 67–126.

ZHU, L., ZHOU, S., AND GUAN, J. 2007. Efficient skyline retrieval on peer-to-peer networks. In *Proceedings of the Future Generation Communication and Networking - Volume 02*. FGCN '07. IEEE Computer Society, Washington, DC, USA, 309–314.