2009

Identifying Interesting Instances for Probabilistic Skylines

Yinian Qi

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Report Number:
09-009

Identifying Interesting Instances for Probabilistic Skylines

Yinian Qi
Mikhail Atallah

CSD TR #09-009
December 2009

Identifying Interesting Instances for Probabilistic Skylines

Yinian Qi
Purdue University
305 N. University Street
West Lafayette, IN 47907, USA
yqi@cs.purdue.edu

Mikhail Atallah
Purdue University
305 N. University Street
West Lafayette, IN 47907, USA
mja@cs.purdue.edu

ABSTRACT

Uncertain data arises from various applications such as sensor networks, scientific data management, data integration, and location based applications. While significant research efforts have been dedicated to modeling, managing and querying uncertain data, advanced analysis of uncertain data is still in its early stages. In this paper, we focus on skyline analysis of uncertain data, modeled as uncertain objects with probability distributions over a set of possible values called *instances*. Computing the exact skyline probabilities of instances is expensive, and unnecessary when the user is only interested in instances with skyline probabilities over a certain threshold. We propose two filtering schemes for this case: a preliminary scheme that bounds an instance's skyline probability for filtering, and an elaborate scheme that uses an instance's bounds to filter other instances based on the dominance relationship. We identify applications where instance-level filtering is useful and desirable. Our algorithms can be easily adapted to filter at the object level if the application domain requires it. Moreover, the uncertain model we adopt in this paper allows *missing probabilities* of uncertain objects as well as arbitrary probability distributions over instances. We experimentally demonstrate the effectiveness of our filtering schemes on both the real NBA data set and the synthetic data set.

1. INTRODUCTION

Skyline analysis is widely used in multi-criteria decision making applications where different criteria often conflict with each other [3]. A classic example is to find a hotel that is close to the beach, yet not expensive. However, proximity to the beach usually suggests higher rates. If we consider hotels as two-dimensional points in the data space with one dimension being the distance to the beach and the other being the hotel rate, we say point p_1 *dominates* 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 skyline analysis then returns all points that are not dominated by any other point in the data set, known as "*skyline points*" [6].

In applications where uncertainty is inherent, such as sensor networks, scientific data management, data integration, and location based applications, the skyline analysis needs to be performed on uncertain data. [29] first proposed *probabilistic skylines* for uncertain data modeled as uncertain objects with probability distributions over a set of possible values called *instances*. This model is useful in practice when the probability density function of an uncertain object is approximated by sampling [29]. Unlike the traditional skyline analysis, with uncertain data many points can represent mutually exclusive instances of the same object, and each instance is now associated with a *skyline probability*: the probability that the instance (i) is the one that occurs among the mutually exclusive set of instances (i.e., the object) to which it belongs, and (ii) is not dominated by any occurring instance of another object. The probability of an instance belonging to the skyline is therefore not only a function of its own probability but also of "how good it is" (as captured by the dominance relationship).

In this paper, we study the problem of identifying "interesting" instances for probabilistic skylines, defined as those with skyline probabilities above a given threshold (the instances are "uninteresting" otherwise). Our goal is to design a scheme that quickly filters (un)interesting instances while avoiding the expensive computations of their exact skyline probabilities. Here by "filtering" an instance we mean that we merely do not bother computing the exact skyline probability of the filtered instance: It is called "*negative* (resp., *positive*) *filtering*" if we do it because we know the instance's skyline probability is below (resp., above) threshold. The key idea of filtering is that the determination of "above threshold" or "below threshold" is done using computations that are much less expensive than the full-blown computation of exact skyline probabilities.

Unlike [29] where thresholding happens at the object level, we filter with a threshold at the instance level. This comes naturally in many applications where the user would be more interested in instances than in the object as a whole. The following two examples give an intuitive motivation for the instance-level filtering.

Example 1 (Job hunting): *Similar to the example in [3], our protagonist Alice is looking for a job with her MBA degree. There are many companies she could consider, each offering multiple job positions for MBAs, with different salaries and job security due to different job responsibilities, work units, geographic locations, etc. Each job is associated with*

a probability that it will be offered to an MBA by the company (the probabilities of all jobs offered within a company sum up to at most 1). Since Alice can only take one job in the end, she wants to save time in job application by looking only at jobs with desirable salary as well as job security (the higher the better for both) and at the same time have a relatively high chance to be offered to her.

If we model the companies as uncertain objects and the job positions available at each company as its instances of two dimensions (salary and job security), then the problem for Alice is really identifying interesting instances with relatively high skyline probabilities (e.g., above a threshold).

Example 2 (Professional Services): *Jane works for a biotech company that uses an expensive piece of equipment that periodically requires servicing (e.g., repair, maintenance, etc.) by a professional who has the specialized skills for the job. Many service providers exist, each of which employs a number of professionals who can do the job. These professionals have different quality characteristics, such as experience, education and professional certification. Moreover, the ability of a service provider to send a particular professional is uncertain and is modeled by a probability, with different professionals possibly having different probabilities due to their different availabilities at the time of the service request (depending, e.g., on whether they work full time or part time, on other customers' demand for them, etc). Since the equipment is expensive and important to the company, Jane wants to find a group of professionals who have high quality characteristics and are likely to be sent over now to examine the equipment.*

Similar to Example 1, we model the service providers as uncertain objects and the professionals who work for each service provider as its instances. Each quality characteristic is modeled as a dimension of the instance. What Jane needs then is a set of instances (i.e., professionals) with relatively high skyline probabilities.

The instance-level probabilistic skyline is especially useful in applications where there is high variability of instances within an object, i.e., some have much higher skyline probabilities than others; the low skyline probabilities of the latter could occur because an instance has a low probability to begin with, or alternatively because it is dominated by too many instances of other objects. In such situations one wants to avoid investing extensive computation time in determining the very low skyline probabilities, instead one wants to quickly identify instances whose skyline probabilities are above the threshold of interest. What to do when an object's instances all fall below the threshold is a separate matter: Our job is only to tag them as such.

1.1 Contributions

We give mechanisms for filtering instances based on approximate values that are easier to compute than these instances' exact skyline probabilities. These values fall into two categories: Upper bounds on the exact skyline probabilities, and lower bounds. If an instance's upper bound is shown to be below the threshold, its exact skyline probability is guaranteed to be below the threshold, i.e., the instance is "*uninteresting*"; we call this "*negative filtering*". Conversely, if an instance's lower bound is shown to be above or equal to the threshold, then its exact skyline probability is guaranteed to meet the threshold, i.e., the instance is "*interesting*"; we call this "*positive filtering*".

Our main contributions are as follows:

- We propose an instance-level probabilistic skyline problem for identifying instances with skyline probabilities over a given threshold. Our algorithm produces fine-grain (i.e., instance-level) information about probabilistic skylines.
- We present two instance-level filtering schemes:
  1. Preliminary filtering scheme: techniques for avoiding the expensive computation of exact skyline probabilities by bounding them with easier-to-compute values for comparing to the threshold.
  2. Elaborate filtering scheme: techniques for massive filtering through inter-instance comparisons that leverage one instance's bounds to filter other instances based on the dominance relationship.
- We adopt a more general uncertain data model than [29] (see Section 2.1). Our experimental results on both real and synthetic data sets under this model show that our algorithms are highly effective in filtering out unqualified instances.

Our preliminary filtering scheme maximizes filtering of instances by tightly bounding their skyline probabilities using indexing structures called "*probabilistic range trees*", which are range trees [35] enhanced with probabilities. Our elaborate filtering scheme further offers the opportunity to take advantage of instances whose bounds have already been computed to either massively "*kill*" (i.e., filter negatively) or "*save*" (filter positively) other instances. For instances that cannot be filtered by our schemes, we resort to the probabilistic range trees for computing exact skyline probabilities. We refer to this as the "*refinement stage*" as opposed to the filtering stage. We measure the merit of our approach by the percentage of instances filtered – the higher the better. This is equivalent to minimizing the number of the exact skyline probability computations carried out.

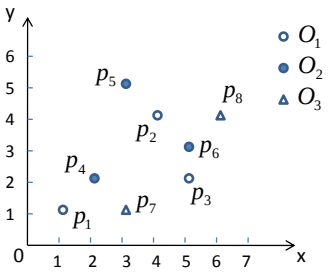
The rest of the paper is organized as follows: Section 2 formally defines our problem. Section 3 introduces the probabilistic range trees for bounding skyline probabilities. Section 4 and Section 5 present our two filtering schemes, followed by the algorithms for computing the exact skyline probabilities in Section 6. The final algorithm for probabilistic skylines is presented in Section 7. Section 8 discusses our experimental study on both real and synthetic data sets. Section 9 reviews the related work, and Section 10 concludes the paper.

2. PROBLEM DEFINITION

In this section, we first review the uncertain data model and the formal definition of the skyline probability. We then define the skyline analysis problem tackled in this paper.

2.1 Uncertain Data Model

Our uncertain data model is similar to that in [29], where uncertain data is modeled as *uncertain objects* with probability distributions over a set of possible values called *instances*. Like [29], we also assume that uncertain objects



Object	O_1			O_2			O_3	
Instance	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
Value	(1, 1)	(4, 4)	(5, 2)	(2, 2)	(3, 5)	(5, 3)	(3, 1)	(6, 4)
Probability	0.2	0.3	0.5	0.4	0.2	0.2	0.2	0.8
Skyline Probability	0.2	0.144	0.24	0.32	0.128	0.048	0.16	0

Figure 1: Probabilistic skylines with three objects and eight instances

are independent from each other to keep our model simple. However, we allow *arbitrary probability distributions* among the instances of an uncertain object, i.e., the instances can take different probabilities to occur. We further allow *missing probabilities* of uncertain objects: Specifically, if the probabilities of all instances of an uncertain object add up to s, then we call $1-s$ the missing probability of the object. This is consistent with the concept of *tuple probability* proposed in the uncertain database community [32].

Figure 1 shows an example of three uncertain objects with their corresponding instances in a two-dimensional space. Notice that O_2 has a missing probability of 0.2, since the probabilities of its three instances sum up to 0.8. This is our running example in the paper that will be referenced frequently in later sections.

2.2 Probabilistic Skylines

Generally, we consider each instance as a d-dimensional point in the data space $\mathcal{D}$. The dominance relationship "$\prec$" between such points (i.e. instances) is the same as the dominance relationship between points for certain data: Given a data set S of n data points: $p_1, \cdots, p_n$ in the data space $\mathcal{D}$ of d dimensions: $\mathcal{D}_1, \cdots, \mathcal{D}_d$, point p_i is said to *dominate* point p_j (denoted as $p_i \prec p_j$) if $\forall k \in [1,d], p_i.\mathcal{D}_k \leq p_j.\mathcal{D}_k$ and $\exists l \in [1,d], p_i.\mathcal{D}_l < p_j.\mathcal{D}_l$. Following the convention in the database community, we assume that smaller values in each dimension are preferred over larger ones. We hence use p to refer to an instance, i.e. a data point in $\mathcal{D}$. The transitivity of the dominance relationship holds between instances [29], i.e. if $p_1 \prec p_2$, $p_2 \prec p_3$, then $p_1 \prec p_3$. An uncertain object O with k instances can be denoted as $O = \{p_1, ..., p_k\}$ where $p_i (1 \leq i \leq k)$ is an instance of O.

Definition 1. The *skyline probability of an instance* p, denoted as $Pr_{sky}(p)$, is the probability that p exists and no instance of other uncertain objects that dominates p exists. Let m be the total number of uncertain objects and let $p \in O_k$. The skyline probability can be computed as:

$$Pr_{sky}(p) = Pr(p) \cdot \prod_{i=1, i \neq k}^{m} (1 - \sum_{q \in O_i, q \prec p} Pr(q)) \quad (1)$$

The *skyline probability of an uncertain object* $O = \{p_1, ..., p_k\}$ is the sum of the skyline probabilities of all its k instances:

$$Pr_{sky}(O) = \sum_{i=1}^{k} Pr_{sky}(p_i) \quad (2)$$

We denote the upper bound of $Pr_{sky}(p)$ as $Pr^+_{sky}(p)$, and the lower bound as $Pr^-_{sky}(p)$. Similarly, we also have $Pr^+_{sky}(O)$ and $Pr^-_{sky}(O)$ for an uncertain object O. In Equation 1, $\prod_{i=1, i \neq k}^{m} (1 - \sum_{q \in O_i, q \prec p} Pr(q))$ is the probability that no instance of other uncertain objects that dominates p exists. This can be obtained by a simple multiplication of probabilities computed from respective uncertain objects due to the independence of the objects. For example, in Figure 1, for p_6 to be a skyline point, none of the instances: p_1, p_3, p_4, p_7 should exist. Since p_6 and p_4 both belong to O_2, the existence of p_6 guarantees that p_4 does not exist. Hence $Pr_{sky}(p_6) = Pr(p_6) * (1 - Pr(p_1) - Pr(p_3)) * (1 - Pr(p_7)) = 0.2 * 0.3 * 0.8 = 0.048$.

The probabilistic skyline problem we study in this paper is defined as follows:

Definition 2. Given a data set S of n instances that belong to m uncertain objects and a probability threshold θ, the instance-level probabilistic skyline analysis returns all instances with skyline probabilities at least θ. i.e. return the skyline set S_{sky} such that:

$$S_{sky} = \{p \in S | Pr_{sky}(p) \geq \theta\}$$

We also refer to S_{sky} (the skyline results) as probabilistic skylines for threshold θ. The main notations used throughout the paper are summarized in Table 3.1.

3. PROBABILISTIC RANGE TREES

We propose two indexing structures based on the range tree [35] to facilitate bounding and computing skyline probabilities. We augment the original range trees with additional probabilistic information stored at the internal nodes, which can be leveraged when querying the trees to quickly bound the skyline probability of a given instance p (the query instance). We call such trees *probabilistic range trees* (PRT). Section 3.2 introduces a general PRT built upon all n instances with probabilistic information. A similar indexing structure is described in Section 3.3, which is built for every uncertain object and has different probabilistic information. A total of m such trees are needed for all m objects. Our algorithms for the preliminary filtering use both trees, as we will see later in Section 4.

3.1 Overview

We first give an overview of probabilistic range trees (PRT), then introduce the two PRT indices we design specifically for computing probabilistic skylines.

We explain the construction of the PRT on n d-dimensional points (representing all instances in the data set S). We begin with the base case of $d = 2$, and follow the presentation

Notation	Meaning
m	number of all uncertain objects
n	number of all instances
$D_1, \cdots, D_d$	the first, $\cdots$, the d-th dimension
O_i	the ith uncertain object
S	the set of all instances ($n = \|S\|$)
p, q	instance (point) in S
$p \prec q$	instance p dominates instance q
Pr_{sky}	skyline probability
Pr^+_{sky}/Pr^-_{sky}	upper bound/lower bound of Pr_{sky}
$L/\hat{L}$	info-list/truncated info-list
T_g/T_{c_i}	general-PRT/colored-PRT of color i
β	probability info in L of the T_g
σ	probability info in L of a T_{c_i}

Table 1: Summary of notations

of [35] modified to accommodate the probabilities. A complete binary tree T is built on top of the points sorted according to dimension D_1. Each internal node v of T points to an *info-list* L_v that contains the points at the leaves of the subtree of T rooted at v, sorted according to their D_2 dimension. Therefore, if v has children u and w, then L_v is the merge of L_u and L_w; we assume that every element of L_v stores its rank in each of the lists L_u and L_w (which implies that once a search item's position has been located in L_v it can be located in L_u and L_w in constant time). The space is obviously $O(n \log n)$. We also assume a derived probability (defined later in Section 3.2.2 and Section 3.3 respectively for the two kinds of PRTs) is associated with every element of L_v.

Figure 2 illustrates a two-dimensional PRT built on top of the eight instances in the example of Figure 1. The leaves of the PRT are the instances by the first dimension. Each of the internal nodes v_1 to v_7 points to an info-list that contains instances sorted by the second dimension. For example, v_2's info-list (L_{v_2}) has four instances p_1, p_7, p_4, p_5 with ascending values in the second dimension. They are instances at the leaf level of the subtree rooted at v_2. Since v_2 has two children: v_4 and v_5, L_{v_2} can be obtained by simply merging L_{v_4} and L_{v_5}.

In the hotel example we gave earlier the dimension d was 2, but in the NBA data set used in the experiment of [29] d was 3, and in most other examples d is a fairly small integer but often exceeds 2. We now sketch how the PRT we gave for $d = 2$ extends to d dimensions within $O(n(\log n)^{d-1})$ performance; note that d is independent of n (i.e., d is $O(1)$) and hence there is always a large enough n that makes the quantity $n(\log n)^{d-1}$ less than dn^2 (if the dataset size n is not truly massive then there is no need for sophistication and even the brute-force dn^2 is practical enough).

A d-dimensional PRT is built inductively using $d-1$ dimensional PRTs: A complete tree T is built whose leaves are the n points sorted according to dimension D_1, and each internal node v of T points to a $d-1$ dimensional PRT that contains the elements at the leaves of the subtree of T rooted at v, organized according to the remaining $d-1$ dimensions (i.e., ignoring D_1). The space complexity is $O(n(\log n)^{d-1})$. Note that our construction ensures that the points in the info-lists are always sorted according to the last dimension. Figure 3 illustrates such a d-dimensional PRT with $d = 3$. A

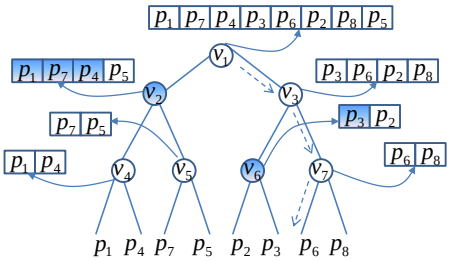


Figure 2: A 2-dimensional probabilistic range tree

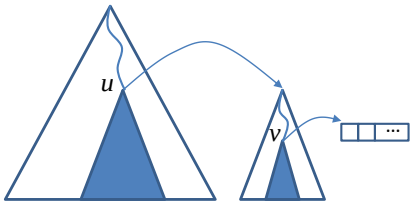


Figure 3: A 3-dimensional probabilistic range tree

node u in this PRT points to a two-dimensional PRT where a node v points to an info-list.

3.2 General Probabilistic Range Tree

To compute probabilistic skylines, we build the *general probabilistic range tree* (*general-PRT*) on all n instances in the data set S.

3.2.1 Probabilistic Information

In the overview of probabilistic range trees, we did not explain what probabilistic information an info-list contains. Here we take a closer look at the info-lists in the general-PRT.

Figure 2 gives us a preliminary idea of info-lists in a PRT. The probabilities associated with the instances in the list are not shown in the figure, since they differ between the general-PRT and colored-PRTs. For the general-PRT, such probabilities are defined as follows:

Definition 3. Let p be the k-th instance in an info-list L. Let p belong to an uncertain object $O_i (1 \le i \le m)$, let $\hat{L}$ be the list of the first k instances in L, then the probability associated with p (denoted as β_p) in L is computed as:

$$\beta_p = \prod_{i=1}^{m} (1 - \sum_{q \in \hat{L}, q \in O_i} Pr(q)) \quad (3)$$

In other words, the probability β_p is the probability that no instance in $\hat{L}$ exists, i.e., the probability that p does not exist and no instance before p in the info-list L exists.

3.2.2 Creating Info-Lists

Given a set of instances, we can create an info-list L by adding each instance to L and then sort all instances by their d-th dimension values. After this, we need to compute the probabilistic information associated with each instance in

Goal: compute β_p for each instance p in info-list L
Algorithm:
1. **for each** $p \in L$ **do**
2. **if** $p \in O_i$ **then**
3. $s_i = 0$ // initialize probability sums
4. **end if**
5. **end for each**
6. $\beta = 1$ // initialize the current β
7. **for each** $p \in L$ **do**
8. **if** $p \in O_i$ **then**
9. $s = s_i$ // back up the old s_i
10. $s_i = s_i + Pr(p)$ // update s_i
11. $\beta_p = \beta/(1-s) * (1-s_i)$ // compute β_p
12. $\beta = \beta_p$ // update the current β
13. **end if**
14. **end for each**

Figure 4: Computing β's for an info-list

the info-list. Based on Equation 3, we compute β_p for each p in L as shown in Figure 4. We use s_i to record the current probability sum for object O_i that has appeared in L. As we go through the instances in L, we update the corresponding probability sum (line 10), and compute β_p based on the β of the instance immediately before p in L (line 11).

As a concrete example on computing β's, let us look back at Figure 2 where info-lists of a PRT is shown. Since the instances in this figure are from the example in Figure 1, they are from different objects. Hence the PRT is actually the general-PRT. Therefore, for the info-list of the node v_2 (L_{v_2}), we can compute the probabilities associated with each instance using the algorithm in Figure 4. For example, β_{p_1} in L_{v_2} is $1 - Pr(p_1) = 1 - 0.2 = 0.8$, and β_{p_7} is $0.8 * (1 - Pr(p_7)) = 0.8 * (1 - 0.2) = 0.64$.

The time needed to compute $\beta's$ for an info-list L is $O(|L|)$, since we only scan the list twice. Note that the list of probability sums is only used for computing $\beta's$ of the instances when creating the info-list. It is not stored along with the info-list. Therefore, it will not bring an additional worst-case $O(m)$ space complexity for each info-list in the general-PRT.

3.3 Colored Probabilistic Range Trees

Besides the general-PRT built upon all n instances in S, we also have m specific PRTs, each built upon the instances of the corresponding object. If we render each instance with a color that indicates the source of the instance and match color i to object O_i, then each of these specific PRTs has only one color. Hence we call these trees *colored-PRTs* as opposed to the general-PRT. For the rest of the paper, whenever we say "instance p of color i", we mean "instance p that belongs to object O_i".

Unlike the info-lists of the general-PRT, an info-list of a colored-PRT is associated with probability sums for each instance in the list. For the k-th instance p in an info-list L of a colored-PRT, its probability sum σ_p is computed as follows:

$$\sigma_p = \sum_{q \in \hat{L}} Pr(q) \quad (4)$$

where $\hat{L}$ is the list of first k instances in L. That is, σ_p is the sum of probabilities of all instances up until p in L. For computing all σ's, we simply need to go through each instance in L and accumulate the probability sum. The time complexity is $O(|L|)$. As an example, if we build a colored-PRT upon the instances of O_3 in Figure 1, the colored-PRT has p_7 and p_8 as the leaves and an internal node that is also the root. The info-list pointed to by the root contain the two instances: p_7 and p_8 sorted by the second dimension values. We compute their σ's as follows: $\sigma_{p_7} = Pr(p_7) = 0.2$, $\sigma_{p_8} = \sigma_{p_7} + Pr(p_8) = 0.2 + 0.8 = 1$.

4. A PRELIMINARY FILTERING SCHEME

Now that we have introduced both the general-PRT and the m colored-PRTs, we can use them to compute the bounds of the skyline probabilities that are used for filtering. The scheme presented in this section works at the individual instance level without comparing instances; the next section will present a more refined scheme where more savings are achieved through a mechanism of inter-instance comparisons whereby one instance's elimination (it is not a skyline result) implies a mass extinction of other instances that dominate it, and one instance's survival (it is a skyline result) implies a mass survival of other instances that are dominated by it.

4.1 Obtaining an Upper Bound

Given a query instance p, we can obtain $Pr^+_{sky}(p)$ by querying the general-PRT T_g as follows:

We begin with the base case of $d = 2$, as shown in Figure 5. Given the two-dimensional query $p = (p.D_1, p.D_2)$, we first locate the search path (call it $\mathcal{P}$) in T_g from the root to the position of the value $p.D_1$ among the leaves, then do one binary search for $p.D_2$ in the info-list L_{root} of the root of T_g. We record the position (rank) k of $p.D_2$ in L_{root} and call it the *search position* in L_{root}. We use $L_v[k]$ to denote the k-th instance (let it be q) in the info-list of the node v and $L_v[k].rankL$, $L_v[k].rankR$ to denote the rank of q in the info-lists of v's left child and right child respectively. These ranks are stored so that given the position of q in L_v, we can locate its position in info-lists of v's children in constant time. Since v is initially the root and k is initially the search position in L_{root}, we can obtain the search positions in the successive nodes as we walk down the search path $\mathcal{P}$.

We define the *left fringe nodes* of the PRT given the query instance p as the left children of the nodes on the search path $\mathcal{P}$ and are not nodes on $\mathcal{P}$ themselves. For example, in Figure 2, the search path $\mathcal{P}$ for p_6 is $v_1 -> v_3 -> v_7 -> p_6$ (rendered with dashed arrows). The corresponding left fringe nodes are v_2 and v_6 (highlighted), who are left children of v_1 and v_3 respectively. The leaf p_6 is on $\mathcal{P}$, so it is not a left fringe node despite the fact that it is a left child of v_7 on $\mathcal{P}$.

We use $\hat{L}_v$ to denote the truncated info-list of node v (L_v) with instances up till the search position in L_v. If v is a left fringe node, we call such $\hat{L}_v$ a *qualified info-list* . Figure 2 highlights two qualified info-lists for the query p_6: One contains the first three instances of L_{v_2} and the other contains the first instance of L_{v_6} (refer to Figure 1 for the values of instances).

When we reach the leaf at the end of the query, the variable *upperBound* in Figure 5 is the product of all β's we read along $\mathcal{P}$. It is indeed an upper bound of $Pr_{sky}(p)$, as we will see shortly. The time complexity for such a query is $O(\log n)$. In the example of Figure 2, the upper bound that we get for $Pr_{sky}(p_6)$ is $\beta_{p_4} * \beta_{p3}$, where p_4 and p_3 are the last instances of the two qualified info-lists.

Input: the general PRT T_g and a query instance p
Output: an upper bound of $Pr_{sky}(p)$

```
1.  binary search for p.D_1 to find the search path P
2.  binary search for p.D_2 in L_root, let the position be k
3.  upperBound = 1
4.  v = root                 //walk along P starting from root
5.  while current node v is not a leaf do
6.    if the next node w ∈ P is v.rightChild then
7.      k' = L_v[k].rankL
8.      u = v.leftChild
9.      β = L_u[k'].beta       // read β from v's left child
10.     upperBound = upperBound * β
11.     k = L_v[k].rankR       // locate the position in L_w
12.   end if
13.   else                     // w is a left child of v
14.     k = L_v[k].rankL       // locate the position in L_w
15.   v = w                    // go one level down
16. end while
17. return upperBound
```

Figure 5: Compute an upper bound of $Pr_{sky}(p)$

When $d > 2$, we obtain the upper bound $Pr^+_{sky}(p)$ by querying inductively on $d-1$ dimensional PRTs: Given the query $p = (p.D_1, \cdots, p.D_d)$, we first locate the path $\mathcal{P}$ in T_g from the root to the position of the value $p.D_1$ among the leaves. Then we walk along $\mathcal{P}$ and issue the query of $p' = (p.D_2, \cdots, p.D_d)$ for every $d-1$ dimensional PRT associated with every qualified node. The final $Pr^+_{sky}(p)$ is obtained by multiplying the values returned by all sub-queries. Such a query takes altogether $O((\log n)^{d-1})$ time, as we cannot avoid doing at most $O(\log n)$ binary searches in PRTs of the left fringe nodes for the first $d-1$ dimensions in order to find the qualified info-lists for reading β's.

We can obtain the qualified info-lists ($\hat{L}$'s) by modifying the algorithm in Figure 5: Instead of reading β's in line 9 and multiplying them along the path in line 10, we create an info-list $\hat{L}_u$ containing the first k' instances of L_u and add it to the result. The lemma below states that the set of all instances in $\hat{L}$'s is the set of all instances in S that dominate p, which can be easily proved from the search process and the definition of the general-PRT. Note that the notation $\hat{L}_i$ in the lemma is the i-th qualified info-list, not the qualified info-list at node i.

LEMMA 1. *Let $\hat{L}_1 \cdots \hat{L}_t$ be qualified info-lists for query p. Let $S_{\hat{L}} = \cup_{i=1}^{t} S_{\hat{L}_i}$, where $S_{\hat{L}_i}$ is the set of instances in $\hat{L}_i$. Then we have: 1) $\forall q \in S_{\hat{L}}, q \prec p$; 2) $\forall q' \in S - S_{\hat{L}}, q' \not\prec p$.*

For every $\hat{L}_i$, let β_i be the β associated with the last instance in $\hat{L}_i$, i.e. β_i is the probability that none of the instances in $\hat{L}_i$ exists. The next lemma (can be easily proved by induction) and theorem show that although we cannot compute $Pr_{sky}(p)$ directly from β_i's, we can compute $Pr^+_{sky}(p)$ to help prune p if this upper bound falls below the threshold θ.

LEMMA 2. *$(1-a_1)\cdots(1-a_t) \geq 1-(a_1+\cdots+a_t)$, where $0 \leq a_i \leq 1, 1 \leq i \leq t$, and $t \geq 1$*

Following the notations in Lemma 1, we have:

THEOREM 1. *Let β_i be the probability associated with the last instance in $\hat{L}_i (1 \leq i \leq t)$ where L_i is a qualified info-list for query p, then $\prod_{i=1}^{t} \beta_i$ is an upper bound of $Pr_{sky}(p)$, i.e.*

$$\prod_{i=1}^{t} \beta_i \geq \prod_{j=1}^{m} \left(1 - \sum_{q \in O_j, q \in S_{\hat{L}}} Pr(q)\right) \geq Pr_{sky}(p) \quad (5)$$

The proof of Theorem 1 is available in Appendix A.1. Theorem 1 shows that $\prod_{i=1}^{t} \beta_i$ is an upper bound of the desired $Pr_{sky}(p)$, which proves that the value returned by $T_g.getUpperBound(p)$ is indeed a $Pr^+_{sky}(p)$. This directly points out a way of pruning the query instance: Given a threshold θ, as soon as we see the current product of β's (which is a $Pr^+_{sky}(p)$) fall below θ, we can stop and declare that p is not in the skyline, since $Pr_{sky}(p) < \theta$ must also hold.

4.2 Obtaining a Tighter Upper Bound

While using the general-PRT alone gives us an upper bound of the skyline probability, a tighter upper bound can be achieved by using both the general-PRT and the colored-PRT.

First, let us review Theorem 1: We have proved that

$$\prod_{i=1}^{t} \beta_i \geq \left(1 - \sum_{q \in O_k, q \prec p} Pr(q)\right) \cdot \prod_{j=1, j \neq k}^{m} \left(1 - \sum_{q \in O_j, q \prec p} Pr(q)\right)$$

By dividing $\left(1 - \sum_{q \in O_k, q \prec p} Pr(q)\right)$ and multiplying $Pr(p)$ at both sides, we have

$$\frac{\prod_{i=1}^{t} \beta_i \cdot Pr(p)}{1 - \sum_{q \in O_k, q \prec p} Pr(q)} \geq Pr(p) \cdot \prod_{j=1, j \neq k}^{m} \left(1 - \sum_{q \in O_j, q \prec p} Pr(q)\right) = Pr_{sky}(p)$$

We further observe that $Pr(p) \leq 1 - \sum_{q \in O_k, q \prec p} Pr(q)$ (remember $p \in O_k$), hence

$$\prod_{i=1}^{t} \beta_i \geq \frac{\prod_{i=1}^{t} \beta_i \cdot Pr(p)}{1 - \sum_{q \in O_k, q \prec p} Pr(q)} \geq Pr_{sky}(p)$$

i.e. $\prod_{i=1}^{t} \beta_i \cdot Pr(p) / \left(1 - \sum_{q \in O_k, q \prec p} Pr(q)\right)$ is a tighter upper bound of $Pr_{sky}(p)$ than $\prod_{i=1}^{t} \beta_i$.

We know $Pr(p)$ and $\prod_{i=1}^{t} \beta_i$ from querying the general-PRT, to obtain this tighter upper bound, the only part we need to know is $\sum_{q \in O_k, q \prec p} Pr(q)$, which is a probability sum that can be obtained by querying the PRT of color k, denoted as T_{c_k}. The algorithm for computing this sum given a query instance p is the same as computing the upper bound with the general-PRT in Figure 5 except this time we carry a sum instead of a product along the search path: Whenever a new probability is read from a qualified info-list (remember that the probability now is σ instead of β, see Section 3.3), we add it to the current sum (initialized to 0). The final sum is then the sum of all σ's we read as we walk along the path. The algorithm to get all qualified info-lists in a colored-PRT given query p is exactly the same as that in the general-PRT described in Section 4.1.

The corollary below for querying colored-PRTs can be derived immediately from Lemma 1:

COROLLARY 1. *The set of instances of all qualified info-lists by querying T_{c_k} is the set of all instances of color k in S that dominate p.*

Input: query instance p, the general-PRT T_g
and m colored-PRTs $T_{c_1}, \cdots, T_{c_m}$
Output: a tighter upper bound for $Pr_{sky}(p)$
1. obtain $Pr^{+}_{sky}(p)$ (*oldBound*) by querying T_g //Figure 5
2. **if** $p \in O_k$ **then** //Section 4.2
3. obtain the probability sum (*sum*) by querying T_{c_k}
4. **end if**
5. *newBound* = *oldBound* * $Pr(p)$ / (1 - *sum*)
6. return *newBound*

Figure 6: Compute a tighter upper bound

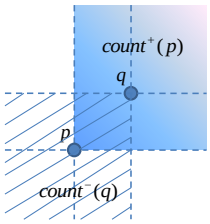


Figure 7: Massive filtering with p and q

Therefore, the probability sum returned by querying T_{c_k} is indeed $\sum_{q \in O_k, q \prec p} Pr(q)$, i.e., the sum of probabilities of all instances that dominate p and belong to O_k at the same time. The algorithm to compute the tighter upper bound is summarized in Figure 6.

4.3 Obtaining a Lower Bound

We start with $d = 2$. For every instance (x, y), we define $s_{iL}(x)$ (resp., $s_{iB}(y)$) to be the sum of the probabilities of instances of color i that are to the left of x (resp., below y). It is straightforward to preprocess the n instances so that a query that asks for $s_{iL}(x)$ or $s_{iB}(y)$ can be processed in $O(\log n_i)$ time: Simply x-sort (resp., y-sort) the instances of color i and store in that sorted list the prefix sums of the probabilities: For each instance p in the list, the prefix sum of p is the sum of probabilities of all instances in the list up till p. Then we process a $s_{iL}(x)$ (resp., $s_{iB}(y)$) query by locating x (resp., y) in that list and reading the relevant prefix sum. Doing such preprocessing for all m colors takes $O(\sum_{i=1}^{m} n_i \log n_i) = O(n \log n)$, where n_i is the number of instances of object O_i. We assume this has been done.

The following lower bound (whose proof we omit) $Pr^{-}_{sky}(p)$ holds for any instance $p = (p.D_1, p.D_2)$ and $p \in O_k$.

$$Pr(p) \cdot \prod_{i=1, i \neq k}^{m} (1 - \min\{s_{iL}(p.D_1), s_{iB}(p.D_2)\})$$

The above lower bound for all n instances can be computed in time $O(m^2 + n \log m)$ (due to the space limit, we omit the details here). While this is good if m is much smaller than n (i.e., if each object has many instances), it is not satisfactory if m is close to n. In such a case we can compute the n lower bounds given below in total time $O(n \log n)$ (they are thus easier to compute, but also less sharp than the above lower bound).

$$Pr(p) \cdot \max\left\{ \prod_{i=1, i \neq k}^{m} (1 - s_{iL}(p.D_1)), \prod_{i=1, i \neq k}^{m} (1 - s_{iB}(p.D_2)) \right\}$$

The above lower bounds can be easily extended to $d > 2$ by computing the sums of probabilities for each dimension, as we did for the first and second dimension in case $d = 2$.

5. AN ELABORATE FILTERING SCHEME

Recall that the preliminary filtering tries to filter out instances by bounding their respective skyline probabilities. The improved filtering scheme of the present section adds inter-instance comparisons to achieve wholesale filtering (positive or negative), i.e., it considers the impact of one instance's elimination or survival on other instances related to it by the dominance relationship. Therefore, the order in which instances are processed (individually, by bounding skyline probabilities as in the preliminary scheme) becomes crucial.

5.1 Filtering Rationale

Before presenting our elaborate filtering scheme, we first define a ratio called the "*key ratio*" for an instance p:

Definition 4. For any instance $p \in O_k$, p's *key ratio* r is:

$$r_p = \frac{Pr(p)}{1 - \sum_{p' \in O_k, p' \prec p} Pr(p')} \quad (6)$$

If $r_p \geq \frac{1}{2}$, we call p a *"target instance"*.

r_p can be easily computed in $O(\log n)$ by querying T_{c_k} to get the probability sum $\sum_{p' \in O_k, p' \prec p} Pr(p')$.

The following theorem states the conditions for negative filtering (see Appendix A.2 for detailed proof):

THEOREM 2. *Let instance $p \in O_k$, instance $q \in O_l$. If $Pr_{sky}(p) < \theta$ and $p \prec q$, then:*
1) $k \neq l$: If p is a target instance, then $Pr_{sky}(q) < \theta$.
2) $k = l$: If p is a target instance or if $Pr(p) \geq Pr(q)$, then $Pr_{sky}(q) < \theta$.

We call instances satisfying the above conditions "killers" – the elimination of themselves causes the massive extinction of others from the skyline result set. In contrast, the corollary below states the conditions for instances to be "saviors" – the survival of themselves causes the survival of others in the final skyline result. The proof of this corollary depends on the proof of $Pr_{sky}(p) \geq Pr_{sky}(q)$, which is exactly the same as the proof in Theorem 2 given in Appendix A.2.

COROLLARY 2. *Let instance $p \in O_k$, instance $q \in O_l$. If $Pr_{sky}(q) \geq \theta$ and $p \prec q$, then:*
1) $k \neq l$: If p is a target instance, then $Pr_{sky}(p) \geq \theta$.
2) $k = l$: If p is a target instance or if $Pr(p) \geq Pr(q)$, then $Pr_{sky}(p) \geq \theta$.

5.2 Schedule Instances

The theorem and corollary in the previous section together point out a way of filtering instances massively based on a single instance's skyline probability. As we have mentioned earlier, the order in which instances are processed is crucial. The goal of our refined filtering scheme is to maximize both negative filtering ("*killing*") and positive filtering ("*saving*") as we process the candidates list so that the number of the PRT queries (either for bounding or computing the exact

skyline probability) is minimized. We propose the following heuristic for scheduling instances to achieve this goal:

Using the standard dominance counting techniques [30], we preprocess all n instances in $O(n \log n)$ time to compute two quantities $count^+(p)$ and $count^-(p)$ for every instance p, where $count^+(p)$ is the number of instances dominated by p and $count^-(p)$ is the number of instances that dominate p. We first sort the instances according to $count^+$ in the descending order. The list then becomes our initial candidate list for computing the skyline results.

Figure 7 shows two instances p and q whose skyline probabilities have been bounded. If p is not a skyline result and is also a target instance, then p can kill all instances in the shaded region at the top-right corner (the number of such instances is $count^+(p)$). Hence q will be killed. On the other hand, if q is a skyline result and p is a target instance, then q can save all target instances in the shaded region at the bottom-left corner (the number of such instances is less than or equal to $count^-(q)$. Hence p will be saved.

5.3 Algorithm

The algorithm for the elaborate filtering consists of two parts: first the negative filtering, then the positive filtering. After scheduling all n instances to form the initial candidate list, we process each instance p in the candidate list in order by upper bounding $Pr_{sky}(p)$ (using the techniques in the preliminary filtering scheme). Then we do the negative filtering as shown in Figure 8. In line 6, we obtain the set of instances that are dominated by p by querying a mirror of our general-PRT (i.e. instead of returning instances that dominate p, it returns instances that are dominated by p). The order that we process instances guarantees that the current instance, if turned out to be a killer, can kill the largest number of instances (because its $count^+$ is the biggest among the unprocessed candidates).

After the candidate list has been exhausted, i.e. all killings have been done, we sort the remaining instances in the list by their $count^-$ in the descending order. We then process each instance q in this new candidate list in order by computing $Pr^-_{sky}(q)$ and compare it with θ to see whether q survives as a skyline result. If it survives, we move it from the candidate list to the skyline result S_{sky}. The rest of the algorithm is similar to the one in Figure 8.

Notice that we do negative filtering (killing) first, followed by positive filtering (saving). This is due to the asymmetry of killing and saving: A killer p kills all instances dominated by p, whereas a savior q only saves a portion of all instances that dominate q — only the target instances among them can be saved. Hence killing filters more than saving. It should come before saving to minimize the number of instances that need to be processed or further evaluated.

6. COMPUTING SKYLINE PROBABILITIES

If an instance cannot be pruned by the preliminary or the elaborate filtering, we need to further evaluate it to decide whether the instance is really a skyline result by computing the exact skyline probabilities. This can be done by querying either the general-PRT or the colored-PRTs.

6.1 Using General-PRT

From qualified info-lists ($\hat{L}$'s) for query p (see Section 4.1), we can get all instances in $\hat{L}$'s. By Theorem 1, these instances are all instances that dominate the query instance

Input: data set S, threshold θ
Output: the candidate list $Cand$ after filtering
1. create the initial $Cand$ from S //Section 5.2
2. **for each** instance p in $Cand$ **do**
3. compute $Pr^+_{sky}(p)$ // Figure 5 and 6
4. **if** $Pr^+_{sky}(p) < \theta$ **then**
5. remove p from $Cand$
6. get the set of instances dominated by p
7. **for each** instance q in the set **do**
8. **if** p is a target instance **then**
9. remove q from $Cand$
10. **else**
11. **if** p, q are of the same color and $Pr(p) \geq Pr(q)$ **do**
12. remove q from $Cand$
13. **end if**
14. **end if**
15. **end for each**
16. **end if**
17. **end for each**
18. **return** $Cand$

Figure 8: Algorithm for negative filtering in the elaborate filtering scheme

p in S. Therefore, we can go through all such instances to compute the exact $Pr_{sky}(p)$ according to Equation 1 and add p to the skyline result if $Pr_{sky}(p) \geq \theta$. The time complexity for computing $Pr_{sky}(p)$ is $O((\log n)^{d-1} + t)$, where t is the number of all instances in $\hat{L}$'s.

6.2 Using Colored-PRTs

For the instance p that belongs to object O_k, we can compute $Pr_{sky}(p)$ by querying all colored-PRTs except the one with color k. For each colored-PRT T_{c_i} $(1 \leq i \leq m, i \neq k)$, we obtain the sum of probabilities of all instances of color i that dominate p (see Section 4.2). We denote this sum as s_i. Then $Pr_{sky}(p) = Pr(p) \prod_{i=1, i \neq k}^{m} (1 - s_i)$. The time complexity for computing $Pr_{sky}(p)$ is $O(m(\log n)^{d-1})$.

7. PROBABILISTIC SKYLINE ALGORITHM

Now that we have presented our preliminary and the more elaborate filtering schemes and our algorithm for computing the exact skyline probabilities, we can propose our final algorithm for computing the instance-level probabilistic skylines given a threshold θ.

7.1 Two-Stage Algorithm

We propose a two-stage scheme for our instance-level probabilistic skyline algorithm, given the threshold θ:

1. **Filtering stage**:
   1) Initialize the skyline result S_{sky} to an empty set
   2) Initialize the candidate list to be all n instances in S
   3) Use the elaborate filtering scheme to reorder the candidate list, eliminate instances with skyline probabilities below θ, and move those with skyline probabilities at least θ to S_{sky}
2. **Refining stage**:
   1) For each remaining instance p in the candidate list, compute the exact $Pr_{sky}(p)$ by querying the PRTs
   2) Add p to S_{sky} if $Pr_{sky}(p) \geq \theta$
   3) Return the final S_{sky} as the set of skyline results to our

probabilistic skyline problem

The filtering stage uses the elaborate filtering scheme we proposed in Section 5, which includes the usage of the preliminary filtering scheme in Section 4. We can also use the preliminary filtering scheme alone in the above algorithm, by changing Step 3 of the filtering stage. The remaining instances in the candidate list after the filtering stage are instances that can neither be eliminated nor guaranteed to belong to S_{sky}. We then query either the general-PRT or the colored-PRTs to compute the exact skyline probabilities of the instances and add those with skyline probabilities at least θ to S_{sky}.

7.2 Probabilistic Skylines at Object Level

While [29] computes all uncertain objects whose skyline probabilities meet a given probability threshold, our probabilistic skyline algorithms return all instances in the data set S whose skyline probabilities meet the threshold. The granularity of our skyline results is at the instance level, which is finer compared with the object level in [29]. Moreover, our instance-level algorithms can be adapted for obtaining the skyline results at the object level as follows:

For each object O_i, we compute the lower and the upper bounds of all its instances by using the preliminary filtering scheme. The sum of the lower bounds (resp. upper bounds) of O_i's instances becomes $Pr^-_{sky}(O_i)$ (resp. $Pr^+_{sky}(O_i)$). Let the threshold for the object-level probabilistic skylines be θ_o. We then check whether $Pr^-_{sky}(O_i) \geq \theta_o$ (i.e. O_i is a skyline result) and whether $Pr^+_{sky}(O_i) < \theta_o$ (i.e. O_i is not a skyline result). If O_i can neither be put to the skyline result set nor be discarded, we further compute the exact skyline probabilities of its instances, sum them up to obtain $Pr_{sky}(O_i)$ and compare it with θ_o.

8. EXPERIMENTAL RESULTS

To evaluate the effectiveness of our filtering schemes and the scalability of our algorithms, we used two data sets, one real data set and one synthetic data set. The experiments were performed on a PC with Intel Core 2 Duo T9600 2.8GHz CPU and 6GB main memory running Ubuntu Linux operating system. All algorithms were implemented in C++. Currently our probabilistic range trees are stored in memory. Our future work will be to store the data structures on the disk to support efficient query processing at a larger scale.

8.1 Data Sets

In our experiments, we used the real data set: the NBA data set as in [29], kindly provided to us by the authors of [29]. The NBA data set contains 339,721 records about 1,313 players. Like in [29], we treat each player as an uncertain object and the records of the player as the instances of the object. Each record has three attributes: number of points, number of assists, and number of rebounds (large values are favored over small ones), i.e., the dimension $d = 3$. We assign random probabilities to instances of the same object such that the probabilities sum up to 1 (later for the synthetic data set, we allow missing probabilities of objects). This is different from [29], which assigns equal probabilities to instances of an object. Allowing different records of a player have different probabilities captures the fact that the physical condition of a player usually changes from game to game (e.g., the player could be in great physical condition in some games, and have suffered from small injuries prior to other games).

Besides the NBA data set, we also used a synthetic data set generated similarly to [29, 6, 3] as follows: We first randomly generated the centers c of each uncertain object. The value at each dimension of an instance has a domain [1, 1000] and was randomly generated in the hyper-rectangular region centered at c with the edge size uniformly distributed in the range [1, 200]. The default number of uncertain objects m is 20,000. The number of instances for an object is uniformly distributed in the range [1, 30] by default. Therefore, if $m = 20,000$ the expected total number of instances n is around 300,000. The default threshold θ is 0.01 for the instance-level skyline probabilities. Although the absolute value of the threshold seems small, this threshold is already very selective among skyline probabilities of all instances, as we can see later in the experiments below. This is mainly due to the fact that an uncertain object may have many instances, resulting in small occurrence probabilities for these instances to begin with before bounding/computing their skyline probabilities.

8.2 Effectiveness of Filtering

We evaluated the effectiveness of our two filtering schemes: we computed the percentage of instances filtered by the upper bounds and the lower bounds in the preliminary scheme, as well as the percentage of instances filtered by massive killing and saving in the elaborate scheme. During the evaluation, we also varied several parameters of the data set to test the scalability of our algorithms as well as to see how the parameters affect the filtering gain. Such parameters include: the data set size (number of objects/instances), the threshold, the average number of instances per object, the number of dimensions.

8.2.1 *Effectiveness of the Preliminary Scheme*

We evaluated the percentage of instances filtered by the upper bounds and the lower bounds in our preliminary scheme on the synthetic data set with $m = 2000$. We also evaluated the respective filtering capabilities of the upper bounds (Section 5) and the corresponding tighter upper bounds (Section 6). The result is shown in the first chart of Figure 9. We always use the "normal" upper bounds (as opposed to "tighter" upper bounds) first to filter instances, as they are easier to compute than the tighter ones. If the normal upper bound is above the threshold (i.e., cannot filter), then we further compute the tighter upper bound to see if the tighter one will help us filter the instance. Notice that if the upper bound is below the threshold, then the tighter upper bound must also fall below the threshold, indicating that both bounds can filter the instance.

Similarly, we show the effect of the lower bound on filtering in the second chart of Figure 9. While the filtering percentage of both upper bounds increases as the threshold increases, the trend is reversed for the lower bound. This is because with higher thresholds, it is easier for an upper bound to fall below the threshold but harder for a lower bound to exceed it. Furthermore, we can see that the two upper bounds filter much more than the lower bound (over 97% of instances are filtered by the upper bounds), although these two kinds of bounds are computed independently.

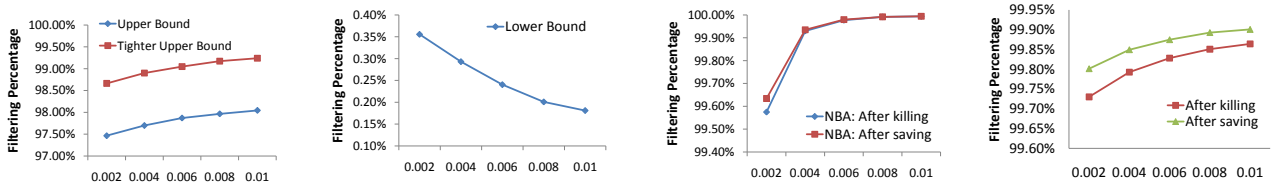


Figure 9: Effectiveness of preliminary filtering and elaborate filtering with respect to threshold

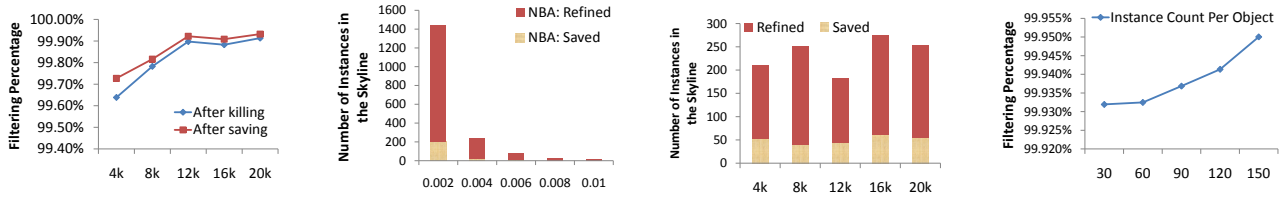


Figure 10: Effect of data set size (m), threshold and instance count per object

8.2.2 *Effectiveness of the Elaborate Scheme*

The elaborate scheme uses the upper and lower bounds from the preliminary scheme, and further exploits the dominance relationship between instances for massive filtering (negative or positive). We evaluated the effectiveness of our elaborate filtering on both the real NBA data set and the synthetic data sets. The percentage of instances filtered after "killing" (i.e., negative filtering) and that after "saving" (i.e., positive filtering) are shown in the third chart of Figure 9 for the NBA data with a varying threshold. The same plot is drawn for the synthetic data in the last chart of Figure 9. For both data sets, killing filters instances massively while saving contributes an additional 0.1% or less to the final filtering percentage, as most of the instances (above 99.5% for a threshold over 0.002) have already been identified as uninteresting. This demonstrates our earlier statement that negative filtering filters much more than positive filtering in the elaborate filtering scheme. We also plotted the filtering percentage against the data set size (i.e., different m's) on the synthetic data in the first chart of Figure 10. As the number of objects increases, the filtering percentage increases in general (because more instances are present to compete with each other and more instances are likely to be dominated by others) with an exception from $m = 12k$ to $m = 16k$, which might due to their particular distributions of instances in the data space.

As we know from Section 7, the final skyline result set consists of two parts: the instances that are saved during the elaborate filtering, and the instances whose exact skyline probabilities are verified to be above the threshold during the refining stage. We call the former "saved" ones; and the latter "refined" ones. The second and the third charts in Figure 10 both compare "saved" and "refined", and display them in stacked columns, since the sum of the two is the actual skyline result size. The former is plotted from the real NBA data set while the latter is drawn from the synthetic data set. The two charts also differ in the x-axes: the former plots the skyline set against the threshold while the latter plots it against the data size m with a fixed threshold at 0.01. We observe that the threshold seems to have a much more significant effect than m on the size of the skyline results – the higher the threshold, the smaller the set. The effect

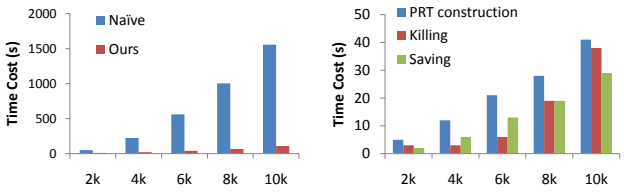


Figure 11: Comparison between our algorithm and the naïve algorithm

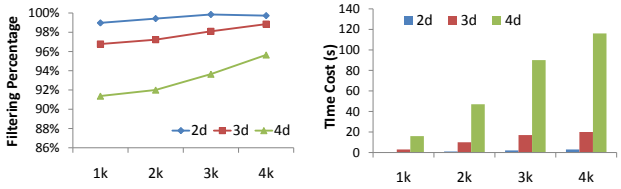


Figure 12: Effect of dimensionality

of m on the skyline size is not as obvious, though we can still presume that the bigger the data set, the bigger the final skyline set. It may depend heavily on the threshold in use: a smaller threshold may yield a clearer trend of this, since there are more instances that are likely to be above the threshold with a larger data set.

8.2.3 *Comparison with the Naive Approach*

We implemented the naïve approach to the instance-level probabilistic skylines for benchmarking, which uses a nested loop ($O(n^2)$) to compute the exact skyline probability of an instance by looking at all other instances. The time cost of our algorithm using the elaborate filtering scheme and that of the naïve algorithm are shown in the first chart of Figure 11. The dimensionality is 3 and the threshold is 0.01. Our algorithm performs significantly better than the naïve one, and the advantage of our algorithm becomes even bigger as the data set size grows. The second chart of Figure 11 provides a detailed view on how the time cost of our algorithm

breaks down to three parts: the time cost for constructing PRT's (the general-PRT and the colored-PRT's), the time cost for negative filtering (killing) and that for positive filtering (saving). We can see that constructing the indices is actually the most expensive of the three: This is due to the fact that when we construct the trees, we also need to compute the probability information stored with each node for later use. Time cost for killing and saving also increases as m grows. More optimizations can be done for saving to reduce the time cost, which involves designing strategies to efficiently compute the lower bound. Due to the space limit, we will not discuss the details here.

Since our algorithms are designed specifically for the instance-level filtering with a more general uncertain model, while [29] focuses on the object-level filtering for probabilistic skylines, we do not think a comparison of the two will yield convincing results when either their algorithms or our algorithms have to be specifically modified and optimized in order to suit the other's case and become comparable.

8.2.4 *Effect of Other Parameters*

We have mentioned the effect of the threshold and m on filtering in the previous sections. Now we will discuss the effect of the dimensions as well as the effect of the number of instances per object.

The two charts in Figure 12 show how filtering percentage and time cost change with different dimensions and different data set size. Here the filtering percentage is computed as the total number of instances killed and saved divided by the total number of instances in the data set. We observe that given a data set, the filtering percentage decreases as the dimension increases. This is because an instance p is less likely to be dominated by another instance q that has values better than or equal to p's own values in every dimension. In addition, for all dimensions, the filtering percentage increases as m increases. For the time cost, increasing dimensions bring increasing overhead in constructing and querying PRT's, as seen in the second chart of Figure 12: the time cost of the 4d data is over ten times more than that of the 2d data for the same number of objects.

Finally, the last chart of Figure 10 shows how filtering changes with respect to the number of instances per object. The x-axis represents the maximum number of instances an uncertain object can have. For example, 90 means the instance count per object is generated uniformly between 1 and 90. We fixed the total number of instances to around 300,000 while changing the range of the instance count per object from [1, 30] to [1, 150]. Intuitively, more instances per object implies fewer objects given a fixed total number of all instances. It also suggests that each instance now has smaller probabilities to occur in the first place (the sum of the probabilities over all instances of an uncertain object cannot exceed 1), which means that the skyline probabilities of these instances may also be smaller because they cannot exceed their own occurrence probabilities. Therefore, the filtering percentage increases, as the number of interesting instances has decreased.

9. RELATED WORK

Many studies have been conducted to design efficient skyline algorithms for large data sets [6, 28, 21, 22, 16, 23]. Various kinds of skyline analysis have been proposed for different settings. For example, skylines for distributed data sets [2, 38], for dynamic data sets [36], for incomplete data [20], for time series [18], etc. In addition, many interesting variations of skyline computation have been explored recently, including representative skylines [27, 34], reverse skylines [14, 25], and privacy-preserving skylines [8].

All the above skyline research focuses on certain data. However, uncertainty in data is inherent in many applications due to the incompleteness of data, delayed updates, measurement inaccuracy, etc. As research on uncertain data draws increasing attention to the database research community, much work has been done to design databases to manage uncertainty [4, 7, 32]. Such uncertainty in data also poses many problems for querying the data, such as the nearest-neighbor problem [10, 12, 5, 9], indexing [31, 11, 19, 1], ranking and top-k queries [33, 26, 17, 13, 15, 24], etc. Advanced data analysis with uncertain data has also been studied recently. For example, several papers have studied skyline analysis for uncertain data [3, 25, 37] following the pioneering work of [29]. Such analysis is useful in applications where uncertainty is inherent. While some applications model uncertainty as continuous with probability density functions (pdf's), others model it as discrete with probability distributions over a set of possible values [32]. This latter case is useful when the nature of the uncertainty is discrete (e.g. integrating data from different sources), or when the underlying pdf is unknown and a set of samples have to be drawn to approximate the real pdf [29]. While [25] focuses on reverse skyline queries for uncertain data, [3, 37, 29] all studied the *probabilistic skyline analysis* proposed in [29]. The goal in [29] was to find all uncertain objects with skyline probabilities greater than or equal to a given threshold. Two algorithms (top-down and bottom-up) were proposed to efficiently compute the skyline results by leveraging upper and lower bounds of the objects' skyline probabilities to avoid the expensive computations of the exact skyline probabilities. In contrast, [3] took a different approach by motivating the problem of computing skyline probabilities for all instances and proposed a sub-quadratic algorithm for doing so. It also removed the two assumptions made in [29] that the instances of the same object has equal probabilities and the probabilities of the instances sum up to 1. [29, 3] are closest to our work. The difference between our paper and [3] is obvious: We aim at minimizing the number of skyline probability computations by introducing a threshold and designing algorithms to leverage the threshold for filtering, whereas [3] designed an algorithm with a worse-case sub-quadratic time complexity to compute all skyline probabilities and hence no threshold is needed. The difference between our paper and [29] is two-fold: i) we filter at the instance level instead of the object level; ii) like [3], we remove the two assumptions of [29] made on the uncertain data model. [37], on the other hand, extends [29] by continuously retrieving skyline results over sliding windows.

10. CONCLUSIONS

In this paper, we study the problem of computing the probabilistic skylines at the instance level for uncertain objects with multiple instances. We propose two filtering schemes for avoiding the expensive skyline probability computations. In our preliminary filtering scheme, we design two indexing structures based on the range search tree to facilitate bounding of a query instance's skyline probability for either negative or positive filtering. Such range trees are aug-

mented with probabilistic information and are hence called probabilistic range trees. Our more refined filtering scheme uses the preliminary filtering scheme, and further explores the dominance relationship between instances to filter out other instances given one instance whose skyline probability has been shown to either meet the threshold or fall below the threshold. We propose our instance-level probabilistic skyline algorithm based on our filtering schemes, which can also be easily adapted to the computation of the object-level probabilistic skylines. Our experiments show that our algorithm can filter massively to avoid computing the exact skyline probabilities.

11. ACKNOWLEDGMENTS

We are grateful to the authors of [29] for sharing the input NBA data set with us. Portions of this work were supported by Grants CNS-0627488, CNS-0915436, and CNS-0913875 from the National Science Foundation, by Grant FA9550-09-1-0223 from the Air Force Office of Scientific Research, and by sponsors of the Center for Education and Research in Information Assurance and Security.

12. REFERENCES

[1] P. K. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi). Indexing uncertain data. In *SIGMOD*, 2009.

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

[3] M. Atallah and Y. Qi. Computing all skyline probabilities for uncertain data. In *PODS*, 2009.

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

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

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

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

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

[9] R. Cheng, L. Chen, J. Chen, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *EDBT*, 2009.

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

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

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

[13] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, 2009.

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

[15] T. Ge, S. Zdonik, and S. Madden. Top-k queries on uncertain data: On score distribution and typical answers. In *SIGMOD*, 2009.

[16] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, 2005.

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

[18] B. Jiang and J. Pei. Online interval skyline queries on time series. In *ICDE*, 2009.

[19] B. Kanagal and A. Deshpande. Indexing correlated probabilistic databases. In *SIGMOD*, 2009.

[20] M. Khalefa, M. F. Mokbel, and J. Levandoski. Skyline query processing for incomplete data. In *ICDE*, 2007.

[21] Kian-Lee, T. P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.

[22] D. Kossmann, F. Ramsak, and S. Rost. An online algorithm for skyline queries. In *VLDB*, 2002.

[23] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in z order. In *VLDB*, 2007.

[24] F. Li, K. Yi, and J. Jestes. Ranking queries on distributed probabilistic data. In *SIGMOD*, 2009.

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

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

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

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

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

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

[31] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *ICDE*, 2007.

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

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

[34] Y. Tao, D. Ling, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.

[35] D. E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 1985.

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

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

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

APPENDIX

A. DETAILED PROOFS

A.1 Proof of Theorem 1 (Section 4.1)

PROOF. We know from Definition 3 that $\beta_i = \prod_{j=1}^{m}(1 - s_{ij})$ where s_{ij} is the sum of probabilities of instances that belong to O_j and at the same time are instances in $\hat{L}_i$. We expand $\beta_1, \cdots, \beta_t$ as follows:

$$\begin{aligned}
\beta_1 &= (1 - s_{11}) \cdot (1 - s_{12}) \cdots (1 - s_{1m}) \\
\beta_2 &= (1 - s_{21}) \cdot (1 - s_{22}) \cdots (1 - s_{2m}) \\
&\vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \\
\beta_t &= (1 - s_{t1}) \cdot (1 - s_{t2}) \cdots (1 - s_{tm})
\end{aligned}$$

We multiply the above t equations together and obtain:

$$\prod_{i=1}^{t} \beta_i = \prod_{i=1}^{t}(1 - s_{i1}) \prod_{i=1}^{t}(1 - s_{i2}) \cdots \prod_{i=1}^{t}(1 - s_{im})$$

Each product $\prod_{i=1}^{t}(1 - s_{ij}), 1 \leq j \leq m$ on the right hand side (RHS) is for the same uncertain object O_j. Since $0 \leq s_{ij} \leq 1$, apply Lemma 2 m times and we have:

$$\prod_{i=1}^{t} \beta_i \geq \prod_{j=1}^{m} \left(1 - \sum_{i=1}^{t} s_{ij}\right) = \prod_{j=1}^{m} \left(1 - \sum_{q \in O_j, q \in S_{\hat{L}}} Pr(q)\right)$$

From Lemma 1, we further conclude that

$$\prod_{i=1}^{t} \beta_i \geq \prod_{j=1}^{m} \left(1 - \sum_{q \in O_j, q \prec p} Pr(q)\right)$$

Let $p \in O_k$. Since $\sum_{q \in O_k, q \prec p} Pr(q) + Pr(p) \leq 1$, the following holds for the RHS of the above inequality:

$$\begin{aligned}
RHS &= \left(1 - \sum_{q \in O_k, q \prec p} Pr(q)\right) \cdot \prod_{j=1, j \neq k}^{m} \left(1 - \sum_{q \in O_j, q \prec p} Pr(q)\right) \\
&\geq Pr(p) \cdot \prod_{j=1, j \neq k}^{m} \left(1 - \sum_{q \in O_j, q \prec p} Pr(q)\right) = Pr_{sky}(p)
\end{aligned}$$

Therefore, Inequality 5 holds. □

A.2 Proof of Theorem 2 (Section 5.1)

PROOF. 1) Since p is a target instance, $r_p \geq \frac{1}{2}$. We can deduce that

$$Pr(p) \geq 1 - \sum_{p' \prec p, p' \in O_k} Pr(p') - Pr(p) \tag{7}$$

Due to the transitivity of the dominance relationship, any instance that dominates p must dominate q. Hence

$$(7) \geq 1 - \sum_{p' \prec q, p' \in O_k} Pr(p')$$

Using the above inequality and the transitivity of the dominance relationship as well as the fact that $Pr(q) \leq 1 - \sum_{p' \prec q, p' \in O_l} Pr(p')$ (because both p' and q belong to O_l), we have

$$\begin{aligned}
Pr_{sky}(p) &= Pr(p) \cdot \prod_{i=1, i \neq k}^{m} \left(1 - \sum_{p' \prec p, p' \in O_i} Pr(p')\right) \\
&\geq \left(1 - \sum_{p' \prec q, p' \in O_k} Pr(p')\right) \cdot \prod_{i=1, i \neq k}^{m} \left(1 - \sum_{p' \prec q, p' \in O_i} Pr(p')\right) \\
&= \left(1 - \sum_{p' \prec q, p' \in O_l} Pr(p')\right) \cdot \prod_{i=1, i \neq l}^{m} \left(1 - \sum_{p' \prec q, p' \in O_i} Pr(p')\right) \\
&\geq Pr(q) \cdot \prod_{i=1, i \neq l}^{m} \left(1 - \sum_{p' \prec q, p' \in O_i} Pr(p')\right) = Pr_{sky}(q)
\end{aligned}$$

Since $Pr_{sky}(p) < \theta$, $Pr_{sky}(q) < \theta$ also holds.
2) If p is a target instance and $k = l$, the proof in 1) still holds; if $Pr(p) \geq Pr(q)$, since $k = l$, we have:

$$\begin{aligned}
\theta > Pr_{sky}(p) &= Pr(p) \cdot \prod_{i=1, i \neq k}^{m} \left(1 - \sum_{p' \prec p, p' \in O_i} Pr(p')\right) \\
&\geq Pr(q) \cdot \prod_{i=1, i \neq k}^{m} \left(1 - \sum_{p' \prec q, p' \in O_i} Pr(p')\right) = Pr_{sky}(q)
\end{aligned}$$

□