Probabilistic Skyline Operator over Sliding Windows

Wenjie Zhang [#1], Xuemin Lin [#1], Ying Zhang [#1], Wei Wang [#1], Jeffrey Xu Yu [*2]

[#] *University of New South Wales & NICTA*
[1]{zhangw, lxue, yingz, weiw}@cse.unsw.edu.au

[*] *Chinese University of Hong Kong*
[2]yu@se.cuhk.hk

***Abstract*— Skyline computation has many applications including multi-criteria decision making. In this paper, we study the problem of efficient processing of continuous skyline queries over sliding windows on uncertain data elements regarding given probability thresholds. We first characterize what kind of elements we need to keep in our query computation. Then we show the size of dynamically maintained candidate set and the size of skyline. We develop novel, efficient techniques to process a continuous, probabilistic skyline query. Finally, we extend our techniques to the applications where multiple probability thresholds are given or we want to retrieve "top-k" skyline data objects. Our extensive experiments demonstrate that the proposed techniques are very efficient and handle a high-speed data stream in real time.**

I. Introduction

Uncertain data analysis is an important issue in many emerging important applications, such as sensor networks, trend prediction, moving object management, data cleaning and integration, economic decision making, and market surveillance. In many scenarios of such applications, uncertain data is collected in a streaming fashion. Uncertain streaming data computation has been studied very recently and the existing work mainly focuses on aggregates and top-k queries [8], [14], [28].

Skyline analysis has been shown as a useful tool [3], [7], [21], [24] in multi-criterion decision making. Given a certain data set D, an object $s_1 \in D$ dominates another object $s_2 \in D$ if s_1 is better than s_2 in at least one aspect and not worse than s_2 in all other aspects. The skyline on D comprises of objects in D that are not dominated by any other object from D. Skyline computation against uncertain data has also been studied recently [22]. In this paper, we will investigate the problem of efficient skyline computation over uncertain streaming data where each data element has a probability to occur.

Skyline computation over uncertain streaming data has many applications. For instance, in an on-line shopping system products are evaluated in various aspects such as *price*, *condition* (e.g., brand new, excellent, good, average, etc), and *brand*. In addition, each seller is associated with a "trustability" value which is derived from customers' feedback on the seller's product quality, delivery handling, etc. This "trustability" value can also be regarded as occurrence probability of the product since it represents the probability that the product occurs exactly as described in the advertisement in terms of delivery and quality. A customer may want to select a product, say laptops, according to multi-criteria based ranking, such as low price, good condition, and brand preference. For simplicity we assume the customer prefers ThinkPad T61 only and remove the brand dimension from ranking. Table I lists four qualified results. Both L_1 and L_4 are skyline points, L_1 is better than (dominates) L_2, and L_4 is better than L_3. Nevertheless, L_1 is posted long time ago; L_4 is better than (dominates) L_3 but the trustability of the seller of L_4 is low.

TABLE I
Laptop Advertisements.

Product ID	Time	Price	Condition	Trustability
L_1	107 days ago	\$ 550	excellent	0.80
L_2	5 days ago	\$ 680	excellent	0.90
L_3	2 days ago	\$ 530	good	1.00
L_4	today	\$ 200	good	0.48

In such applications, customers may want to continuously monitor on-line advertisements by selecting the candidates for the best deal - skyline points. Clearly, we need to "discount" the dominating ability from offers with too low trustability. Moreover, too old offers may not be quite relevant. We model such an on-line selection problem as probabilistic skyline against sliding windows by regarding on-line advertisements as a data stream (see Section II for details).

Such a data stream may have a very high speed. Consider the stock market application where clients may want to on-line monitor good deals (transactions) for a particular stock. A deal is recorded by two aspects (price, volume) where price is the average price per share in the deal and volume is the number of shares. In such applications, customers may want to know the top deals so far, as one of many kinds of statistic information, before making trade decisions. A deal a is better than another deal b if a involves a higher volume and is cheaper (per share) than those of b, respectively. Nevertheless, recording errors caused by systems or human beings may make unsuccessful deals be recorded successful, and vise versa; consequently each successful deal recorded has a probability to be true. Therefore, a stream of deals may be treated as a stream of uncertain elements and some clients may only want to know "top" deals (skyline) among the most recent N deals (sliding windows); and we have to take into consideration the uncertainty of each deal. This is another

example of probabilistic skyline against sliding windows.

In this paper we investigate the problem of efficiently processing probabilistic skyline against sliding windows. To the best of our knowledge, there is no similar work existing in the literature in the context of skyline computation over uncertain data steams. In the light of data stream computation, it is highly desirable to develop on-line, efficient, memory based, incremental techniques using small memory. Our contribution may be summarized as follows.

- We characterize the minimum information needed in continuously computing probabilistic skyline against a sliding window.
- We show that the volume of such minimum information is expected to be bounded by logarithmic size in a lower dimensional space regarding a given window size.
- We develop novel, incremental techniques to continuously compute probabilistic skyline over sliding windows.
- We extend our techniques to support multiple pre-given probability thresholds, as well as "top-k" probabilistic skyline.

Besides theoretical guarantee, our extensive experiments demonstrate that the new techniques can support on-line computation against very rapid data streams.

The rest of the paper is organized as follows. In Section II, we formally define the problem of sliding-window skyline computation on uncertain data streams and present background information. Section III and Section IV present our theoretic foundation and techniques for processing probability threshold based sliding window queries. Results of comprehensive performance studies are discussed in Section V. Section VI extends our techniques to top-k skyline, time-based sliding windows, and a data object with multiple instances. Section VII summaries related work and Section VIII concludes the paper.

II. Background

We use DS to represent a sequence (stream) of data elements in a d-dimensional numeric space such that each element a has a probability $P(a)$ $(0 < P(a) \leq 1)$ to occur where $a.i$ (for $1 \leq i \leq d$) denotes the i-th dimension value. For two elements u and v, u *dominates* v, denoted by $u \prec v$, if $u.i \leq v.i$ for every $1 \leq i \leq d$, and there exists a dimension j with $u.j < v.j$. Given a set of elements, the *skyline* consists of all points which are not dominated by any other element.

A. Problem Definition

Given a sequence DS of uncertain data elements, a *possible world* W is a subsequence of DS. The probability of W to appear is $P(W) = \Pi_{a \in W} P(a) \times \Pi_{a \notin W}(1 - P(a))$. Let Ω be the set of all possible worlds, then $\sum_{W \in \Omega} P(W) = 1$.

We use $SKY(W)$ to denote the set of elements in W that form the skyline of W. The probability that an element a appears in the skylines of the possible worlds is $P_{sky}(a) = \sum_{a \in SKY(W), W \in \Omega} P(W)$. $P_{sky}(a)$ is called the *skyline probability* of a. The equation (1) below can be immediately verified.

$$P_{sky}(a) = \Pi_{a' \in DS, a' \prec a}(1 - P(a')) \quad (1)$$

In many applications, a data stream DS is *append-only* [15], [20], [25]; that is, there is no deletion of data element involved. In this paper, we study the skyline computation problem restricted to the append-only data stream model. In a data stream, elements are positioned according to their relative arrival ordering and labelled by integers. Note that the position/label $\kappa(a)$ means that the element a arrives $\kappa(a)$th in the data stream.

Problem Statement. In this paper, we study the problem of efficiently retrieving skyline elements from the most recent N elements, seen so far, with the skyline probabilities not smaller than a given threshold q $(0 < q \leq 1)$; that is, q-skyline. Specifically, we will investigate the problem of efficiently processing such a *continuous* query, as well as *ad-hoc* queries with a probability threshold $q' \geq q$.

B. Preliminaries

Various Dominating Probabilities. Let DS_N denote the most recent N elements. For each element $a \in DS_N$, we use $P_{new}(a)$ to denote the probability that none of the new arrival elements dominates a; that is,

$$P_{new}(a) = \Pi_{a' \in DS_N, a' \prec a, \kappa(a') > \kappa(a)}(1 - P(a')) \quad (2)$$

Note that $\kappa(a') > \kappa(a)$ means that a' arrives after a. We use $P_{old}(a)$ to denote the probability that none of the early arrival elements dominates a; that is,

$$P_{old}(a) = \Pi_{a' \in DS_N, a' \prec a, \kappa(a') < \kappa(a)}(1 - P(a')) \quad (3)$$

The following equation (4) can be immediately verified.

$$P_{sky}(a) = P(a) \times P_{old}(a) \times P_{new}(a). \quad (4)$$

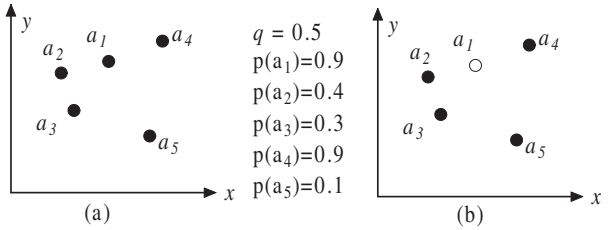


Fig. 1. A Sequence of Data Elements

Example 1: Regarding the example in Figure 1(a) where the occurrence probability of each element is as depicted, assume that $N = 5$, and elements arrive according the element subindex order; that is, a_1 arrives first, a_2 arrives second, ..., and a_5 arrives last. $P_{new}(a_4) = 1 - P(a_5) = 0.9$ and $P_{old}(a_4) = (1 - P(a_2))(1 - P(a_3))(1 - P(a_1)) = 0.042$, and $P_{sky}(a_4) = P(a_4)P_{new}(a_4)P_{old}(a_4) = 0.034$. □

Dominance Relationships. Our techniques will be based on R-trees. Below we define various relationships between each pair of entries E' and E. We use $E.min$ to denote the lower-left corner of the minimum bounding box (MBB) of the elements contained by E, and $E.max$ to denote the upper-right corner of MBB of the elements contained by E. Note

that when E degenerates to a single element a, $E.min = E.max = a$.

An entry E *fully dominates* another entry E', denoted by $E \prec E'$, if $E.max \prec E'.min$ or $E.max = E'.min$ with the property that there is no element in E allocated at $E.max$ or there is no element in E' allocated at $E'.min$. E *partially dominates* E' if $E.min \prec E'.max$ but E does not fully dominates E'; this is denoted by $E \prec_{partial} E'$. Otherwise, E does *not dominate* E', denoted by $E \prec_{not} E'$.

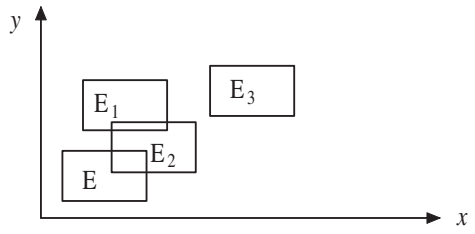


Fig. 2. Dominance relationships.

As depicted in Figure 2, E fully dominates E_3, and partially dominates E_1 and E_2. Note that E_1 does not dominate E but $E_2 \prec_{partial} E$. Clearly, some elements in E_1 may be dominated by elements in E but elements in E cannot be dominated by any elements in E_1. This can be formally stated below which can be verified immediately according to the definitions.

Theorem 1: Suppose that $E \prec_{partial} E'$. Then some elements in E' might be dominated by elements in E. However, if $E' \prec_{not} E''$. Then elements in E'' cannot be dominated by any element in E'.

III. Framework

Given a probability threshold q and a sliding window with length N, below in Algorithm 1 is the framework where a_{old} is the oldest element in current window DS_N and inserting (a_{new}) incrementally computes q-skyline.

Algorithm 1: Continuous Probabilistic Skyline Computation over a Sliding Window

```
while a new element a_new arrives do
    if κ(a_new) ≤ N then Inserting (a_new);
    else Expiring (a_old); Inserting (a_new);
end while
```

Let $S_{N,q}$ denote the set of elements from DS_N with their P_{new} values not smaller than q; that is,

$$S_{N,q} = \{a | a \in DS_N \& P_{new}(a) \geq q\} \quad (5)$$

A critical requirement in data stream computation is to have small memory space and fast computation. In our algorithms, instead of conducting the computation against a whole sliding window (N elements), we do the computation restricted to $S_{N,q}$ which will be shown logarithmic in size regarding N on average. Next, we first show the correctness of restricting the computation to $S_{N,q}$.

A. Using $S_{N,q}$ Only

In this subsection, we will show the following two things: 1) $S_{N,q}$ contains all skyline points with $P_{sky} \geq q$; and 2) computing P_{sky} and P_{new} against $S_{N,q}$ will not lead to *false positive* nor *false negative* to continuously identify $S_{N,q}$ and $SKY_{N,q}$ where $SKY_{N,q}$ is the solution set; that is, for each element a in $SKY_{N,q}$, $P_{sky}(a) \geq q$.

No Missing Elements. The following Lemma is immediate based on (4).

Lemma 1: Each q-skyline point a (i.e., $P_{sky}(a) \geq q$) must be in $S_{N,q}$.

No False Hits to Determine $S_{N,q}$. Suppose that $P_{new}|_{S_{N,q}}(a)$, $P_{old}|_{S_{N,q}}(a)$ and $P_{sky}|_{S_{N,q}}(a)$ denote $P_{new}(a)$, $P_{old}(a)$ and $P_{sky}(a)$ restricted to $S_{N,q}$, respectively.

Example 2: Regarding the example in Figure 1, suppose that elements a_1, a_2, a_3, a_4, and a_5 arrive at time 1, 2, 3, 4, and 5, respectively, and $N = 5$, q = 0.5. We have that $S_{N,q} = \{a_2, a_3, a_4, a_5\}$ since values of P_{new} for a_2, a_3, and a_5 are the same 1, while $P_{new}(a_4) = 0.9$ as shown in Example 1. It can be immediately verified that their P_{new} values restricted to $S_{N,q}$ remain unchanged. Example 1 also shows that $P_{old}(a_4) = 0.042$, while $P_{old}(a_4)|_{S_{N,q}} = 0.6 \times 0.7 = 0.42$ since a_1 is not contained in $S_{N,q}$. □

Next, we show that for each element a in $S_{N,q}$, calculating $P_{new}(a)$ against $S_{N,q}$ is the same as calculating against the whole window DS_N.

Theorem 2: For each element $a \in S_{N,q}$, $P_{new}|_{S_{N,q}}(a) = P_{new}(a)$.

Theorem 2 immediately follows from the following Lemma.

Lemma 2: For each element $a \in S_{N,q}$, if there is an element $a' \in DS_N$ such that $a' \prec a$ and a' is newer than a, then $a' \in S_{N,q}$.

Proof: Since $a' \prec a$ and a' is newer than a, each element that is newer than a' and dominates a' must dominate a. Consequently, $P_{new}(a) \leq P_{new}(a')$. As $P_{new}(a) \geq q$, $P_{new}(a') \geq q$. Thus, the theorem holds. ■

Note that P_{old} values against $S_{N,q}$ are imprecise; nevertheless, below we will show that these will not affect a correct determination of $SKY_{N,q}$.

No False Negative to Determine $SKY_{N,q}$. We show that there is no $a \in SKY_{N,q}$ such that $P_{sky}|_{S_{N,q}}(a) < q$.

Theorem 3: For each element $a \in S_{N,q}$, if $P_{old}(a) \times P_{new}(a) \geq q$ then $P_{old}|_{S_{N,q}}(a) = P_{old}(a)$.

Theorem 3 immediately follows the following lemma - Lemma 3

Lemma 3: For an element a' such that $a' \prec a$, a' arrives earlier than a, and $P_{old}(a) \times P_{new}(a) \geq q$, then $a' \in S_{N,q}$.

Proof: Since $a' \prec a$, any element dominating a' must dominate a. Consequently, $P_{new}(a') \geq P_{new}(a) \times P_{old}(a) \geq q$. Thus, $a' \in S_{N,q}$. ■

Note that $P_{sky}(a) = P(a)P_{old}(a)P_{new}(a)$ where $P(a) \leq 1$. This, together with Lemma 1, Theorems 2 and 3, immediately implies the following corollary.

Corollary 1: For each element $a \in S_{N,q}$, if $P_{sky} \geq q$ then $P_{sky}(a) = P_{sky}|_{S_{N,q}}(a)$.

Corollary 1 immediately implies there is no false negative; that is, there is no $a \in SKY_{N,q}$ such that $P_{sky}|_{S_{N,q}}(a) < q$.

No False Positive to Determine $SKY_{N,q}$. We show that there is no $a \in S_{N,q}$ such that $P_{sky}|_{S_{N,q}}(a) \geq q$ and $P_{sky}(a) < q$.

Theorem 4: For each element $a \in S_{N,q}$, if $P_{old}(a) \times P_{new}(a) < q$, then $P_{old}|_{S_{N,q}}(a) \times P_{new}|_{S_{N,q}}(a) < q$.

Proof: If every element dominating a is in $S_{N,q}$ then $P_{old}(a)|_{S_{N,q}} \times P_{new}(a)|_{S_{N,q}} = P_{old}(a) \times P_{new}(a) < q$. The theorem holds.

Suppose that at least one element that dominates a is not in $S_{N,q}$. From Lemma 2, all such elements must be older than a. Let $Dom(a)$ denote the set of elements that dominate a and are not in $S_{N,q}$. Suppose that a' is the youngest element in $Dom(a)$. It is clear that all elements, which arrive after a' and dominate a', must be contained by $S_{N,q}$ since they dominate a and younger than a'.

Note that $P_{new}(a') < q$. Consequently, $q > P_{new}(a') \geq P_{old}|_{S_{N,q}}(a) \times P_{new}|_{S_{N,q}}(a)$. ■

Note that $P_{sky}(a) = P(a)P_{old}(a)P_{new}(a)$ and $P(a) \leq 1$. These, together with Theorems 2, 3, and 4, immediately imply the following corollary.

Corollary 2: For each element $a \in S_{N,q}$, if $P_{sky}|_{S_{N,q}}(a) < q$, then $P_{sky}(a) < q$.

Therefore, in our techniques we only need to maintain $S_{N,q}$, calculate all probabilities against $S_{N,q}$, and select elements a with $P_{sky}|_{S_{N,q}}(a) \geq q$. For notation simplification, in the remaining of the paper, $P_{sky}|_{S_{N,q}}$, $P_{old}|_{S_{N,q}}$, and $P_{new}|_{S_{N,q}}$ are **abbreviated to** P_{sky}, P_{old}, P_{new}, respectively if there is no ambiguity.

B. Estimating sizes of $S_{N,q}$ and $SKY_{N,q}$

Minimality. It can be immediately verified that in order to avoid getting a wrong solution, $S_{N,q}$ is the **minimum** information to be maintained.

Theorem 5: Each element a in the current $S_{N,q}$ with $P(a) \times P_{new}(a) < q$ will never become a q-skyline point; however, there is a data stream such that removing a away will lead to false positive. Moreover, an $a \in S_{N,q}$ with $P(a) \times P_{new}(a) \geq q$ and $P_{sky} < q$ may become a skyline point if old elements dominating e expire and newly arriving elements do not dominate e.

Theorem 5 is quite intuitive and we omit the proof due to space limits. Below we give an example.

Example 3: Regarding the example in Figure 1 (a), assume that $N = 4$. Considering the first window, there are 4 elements a_1, a_2, a_3, and a_4. $S_{N,q} = \{a_2, a_3, a_4\}$ since $P_{new}(a_1) = 0.6 \times 0.7 < 0.5$, while P_{new} values for a_2, a_3, a_4 are all 1. Note that $P_{sky}|_{S_{N,q}}(a_4) = 0.378$; consequently, a_4 is not a q-skyline point based on the current window.

Regarding the second window when a_1 expires and a_5 arrives. $S_{N,q} = \{a_2, a_3, a_4, a_5\}$ where $P_{new}(a_4) = 0.9$. Other P_{new} values are 1, $P_{sky}(a_3) = P(a_3) = 0.3 < 0.5$, and $P_{sky}(a_4) = 0.34 < 0.5$. If we do not record a_3 and a_4 in $S_{N,q}$, then $P_{sky}(a_4)$ will be calculated as $(1-P(a_5))P(a_4) > 0.5$ leading to the false result, because $P_{sky}(a_4)$ should be $(1 - P(a_2))(1 - P(a_3))(1 - P(a_5))P(a_4) < 0.5$.

Assume that a_1 and a_2 expire, a_5 is as illustrated, and a_6 does not dominate a_4. Regarding the window containing a_3, a_4, a_5, and a_6, $P_{sky}(a_4) = 0.9 \times (1 - 0.3) \times (1 - 0.1) > 0.5$; thus, a_4 is a skyline point. □

Estimating Sizes. Next we show that the expected sizes of $S_{N,q}$ and $SKY_{N,q}$ are bounded by a logarithmic number regarding N.

Suppose that $\chi_{q,i}$ is a random variable such that it takes value 1 if the ith arrival element is a q-skyline point; and $\chi_{q,i}$ takes 0 otherwise. Clearly, the expected size $E(SKY_{N,q})$ of $SKY_{N,q}$ is as follows.

$$E(SKY_{N,q}) = E(\sum_{i=1}^{N} \chi_{q,i}) = \sum_{i=1}^{N} P(\chi_{q,i} = 1) \quad (6)$$

Let $I_N = \{j | 1 \leq j \leq N\}$. Given a set of N probability values $\{P_j | 1 \leq j \leq N \ \& \ 0 < P_j \leq 1\}$, let $P(\neg w) \stackrel{\triangle}{=} \prod_{j \in W}(1 - P_j)$ where W is a subset of I_N. Let $P(W \prec i)$ denote the probability that the ith element is dominated and only dominated by the elements in $\{a_j | j \in W\}$.

Theorem 6: Let DS_N be a sequence of N data elements with probabilities P_1, P_2, ... , P_N. Then,

$$E(SKY_{N,q}) = \sum_{\forall W, i \notin W, P_i \times P(\neg W) \geq q} P(W \prec i) \times P_i \times P(\neg W) \quad (7)$$

Below we show that (7) is bounded by a logarithmic size. Given a P_i, let $q_{k,i} \stackrel{\triangle}{=} \max\{P_i \times P(\neg W) | \ |W| = k\}$. Removing the probability value from each data element in DS_N to make DS_N be a sequence DS_N^c of N certain data elements. Let $P(DOM_i^k)$ denote the probability that there are exactly k elements in DS_N^c dominating an element i. The following lemma immediately follows from (6). Clearly, $q_{k,i}$ is monotonically decreasing regarding k; that is, $q_{k',i} >= q_{k,i}$ if $k' < k$. Let k_i denote the largest integer such that $q_{k,i} \geq q$ for a given q.

Lemma 4: $E(SKY_{N,q}) \leq \sum_{i=1}^{N} \sum_{j=0}^{k_i} P(DOM_i^j) \times q_{j,i}$.

Let $P(DOMT_i^k)$ denote the probability that there are at most k elements dominating the element i. Clearly, $P(DOM_i^k) = P(DOMT_i^k) - P(DOMT_i^{k-1})$.

Corollary 3:

$$\begin{aligned} E(SKY_{N,q}) \quad &\leq \quad \sum_{i=1}^{N}(\sum_{j=0}^{k_i - 1} P(DOMT_i^j) \times (q_{j,i} - q_{(j+1),i}) \quad (8) \\ &+ \quad P(DOMT_i^{k_i}) q_{k_i,i}). \end{aligned}$$

Let $H_{1,l} = \sum_{i=1}^{l} \frac{1}{i}$. The d-th order harmonic mean (for integers $d \geq 1$ and $l \geq 1$) is $H_{d,l} = \sum_{i=1}^{l} \frac{H_{d-1,i}}{i}$. The theorem below presents the value of $P(DOMT_i^k)$.

Theorem 7: For a sequence DS_N^c of N certain data points in a d-dimensional space, suppose that the value distribution of each element on any dimension is the same and independent. Moreover, we assume the values of the data elements in each dimension are distinct. Then, $P(DOMT_i^k) \leq \frac{k+1}{N} \times (1 + H_{d-1,N} - H_{d-1,k+1})$ when $d \geq 2$ and $P(DOMT_i^k) = (k + 1)/N$ when $d = 1$.

Proof: Without lose of generality, we assume that the data elements in DS_N^c are sorted on the first dimension. Since

the value distribution of each element on any dimension is the same and independent, an element has the equal probability to take jth position on the first dimension among total N positions; that is $\frac{1}{N}$ probability to take jth position ($1 \leq j \leq N$) on the first dimension. *Note that when a_i takes jth position. any element takes j'th position cannot dominate a_i if $j' > j$.*

When $d = 1$, element a_i must take the first $(k+1)$ positions to ensure there are at most k other elements dominating a_i. Consequently, $P(DOMT_i^k) = (k+1)/N$.

We use mathematic induction to prove the theorem for $d \geq 2$. For $d = 2$, clearly when a_i takes the first $(k+1)$ positions, there are at most $(k+1)$ other elements dominating a_i. When a_i takes a jth position for $j > k+1$, the conditional probability that there must be at most k elements dominating a_i is $\frac{k+1}{j}$ since for each permutation with a_i at jth position on the first dimension, the the value of a_i on the second dimension must take one of the $(k+1)$ smallest value among the j elements with the j smallest values on the first dimension. Thus, we have:

$$\begin{aligned} P(DOMT_i^k) &= \frac{(k+1)}{N} + \frac{1}{N}\left(\sum_{j=k+2}^{N} \frac{k+1}{j}\right) \\ &= \frac{k+1}{N} \times (1 + H_{1,N} - H_{1,k+1}) \end{aligned} \tag{9}$$

Assume that the theorem holds for $d = l$. For $d = l+1$, it still holds that when a_i's value on the first dimension is allocated at the first $(k+1)$ positions, then there must be at most k other elements dominating a_i. When a_i takes a jth position for $j > k+1$, the conditional probability that there are at most k elements dominating a_i is $P(DOM_i^k)_{j,l}$ regarding a l-dimensional space and j elements for each permutation with a_i at jth position on the first dimension. Based on our assumption, $P(DOM_i^k)|_{j,l} \leq \frac{k+1}{j} \times (1 + H_{l-1,j} - H_{l-1,k+1})$; consequently, the $P(DOM_i^k)$ regarding the $(l+1)$-dimensional space and N data elements is:

$$P(DOMT_i^k) \leq \frac{k+1}{N} + \frac{1}{N}\sum_{j=k+2}^{N} \frac{k+1}{j} \times (1 + H_{l-1,j} - H_{l-1,k+1})$$

Since $1 \leq H_{l-1,k+1}$, we have:

$$\begin{aligned} P(DOMT_i^k) &\leq \frac{k+1}{N} + \frac{1}{N}\sum_{j=k+2}^{N} \frac{k+1}{j} \times (H_{l-1,j}) \\ &= \frac{k+1}{N}(1 + H_{l,N} - H_{l,k+1}) \end{aligned}$$

It can be immediately verified that $H_{d,N} = O(\ln^d N)$; consequently $P(DOMT_i^k) = O(k \ln^{d-1} N)$. This together with Theorem 7 and Corollary 3 immediately implies that the expected size of $SKY_{N,q}$ in a d-dimensional space is poly-logarithmic regarding N with order $(d-1)$. ∎

Size of $S_{N,q}$. Elements in the candidate set can be regarded as skyline points in a $(d+1)$-space by including the time as an additional dimension since P_{new} can be regarded as the non-dominance probability in such a $(d+1)$-space. We have the following theorem.

Theorem 8: In a d-dimensional space, suppose that the distribution on each dimension, including arriving order are independent. On each dimension, the values of the data items are distinct. Let $P(skyt_i^j)$ denote the probability that there are at most j elements in DS_N^c (remove element probabilities from DS_N) dominating the ith element. Let $p_{k,i} \triangleq \max\{P(\neg W)|\ |W| = k\}$

$$\begin{aligned} E(SKY_{N,q}) &\leq \sum_{i=1}^{N}\sum_{j=0}^{k_i-1} P(skyt_i^j) \times (p_{j,i} - p_{(j+1),i}) \\ &+ P(skyt_i^{k_i}) p_{k_i,i}. \end{aligned} \tag{10}$$

Note that $P(skyt_i^{k_i})$ can be estimated in the same way as that in Theorem 7 by replacing d by $d+1$. Therefore, the expected size of $S_{N,q}$ is poly-logarithmic regarding N with the order of d.

IV. Algorithms

A trivial execution of Algorithm 1 is to visit each element in $S_{N,q}$ to update skyline probability when an element inserts or deletes; then choose elements a from $S_{N,q}$ with $P_{sky}(a) \geq q$. Note a new data element may cause several elements to be deleted from $S_{N,q}$, nevertheless, the amortized time complexity is $O(|S_{N,q}|)$ per element which is poly-logarithmic regarding N with the order of d (Section III-B).

In this section, we present novel techniques to efficiently execute Algorithm 1 based on aggregate-R trees with the aim to visit as few elements as possible. We continuously, incrementally maintain $SKY_{N,q}$ and $S_{N,q}$.

The rest of the section is organized as follows. We first present data structures to be used. Then we present our efficient techniques to deal with the arrival of a new element for a given probability threshold. This is followed by our techniques to deal with the expiration of an old element for a given probability threshold. Then, we extend our techniques to deal with applications where multiple probability thresholds are given. Finally, correctness and complexity of our techniques are shown.

A. Aggregate R-trees

Since $SKY_{N,q} \subseteq S_{N,q}$, we continuously maintain $SKY_{N,q}$ and $(S_{N,q} - SKY_{N,q})$ to avoid store a data element twice.

In-memory R-trees R_1 and R_2 on $SKY_{N,q}$ and $(S_{N,q} - SKY_{N,q})$, respectively will be used and continuously maintained. We aim to conduct an efficient computation. Thus, we develop in-memory aggregate R-trees based on the following observation.

Observation. Regarding the example in Figure 3, assume that $N = 13$, $q = 0.2$, the occurrence probabilities are as depicted, and $DS_N = \{a_i | 1 \leq i \leq 13\}$. Suppose that elements arrive according to the increasing order of elements sub-indexes. It can be immediately verified that $P_{new}(a_1) < 0.2$, $S_{N,q}$ contains a_i for $2 \leq i \leq 13$, and $SKY_{N,q}$ contains only the elements in R_1. Two R-trees are built: 1) R_1 is built against the elements in $SKY_{N,q}$; and 2) R_2 is built against the elements in $(S_{N,q} - SKY_{N,q})$.

When a new element a_{14} arrives and a_1 expires. We need to find out the elements which are dominated by a_{14} and then to determine the elements which need to be removed from

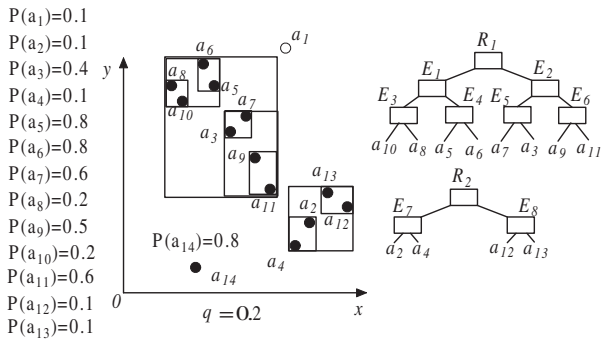


Fig. 3. Aggregate R-trees

$S_{N,q}$ and $SKY_{N,q}$. In fact a_{14} dominates entries E_4, E_2, and $R2.root$ (root entry of R_2). If we keep the maximum and minimum values of P_{new} for the elements contained by those entries, respectively, we have a chance not to visit the elements of those entries. Specifically, at an entry if the maximum values of P_{new} multiplied by $(1-P(a_{14}))$ smaller than q, the entry (i.e. all elements contained) will be removed from $S_{N,q}$. On the other hand if the minimum value of P_{new} multiplied by $(1-P(a_{14}))$ is not smaller than q, then the entry (i.e. all elements contained) remains in $S_{N,q}$. Similarly, at each entry we keep the minimum and maximum values of P_{sky} for the elements contained to possibly terminate the determination of whether elements contained are in $SKY_{N,q}$.

Moreover, in this example elements contained by E_2 is in $S_{N,q}$, we can update their P_{new} values globally by keeping a global value $P_{new}^{global} = P_{new}^{global} \times (1-P(a_{14}))$ at E_2 to avoid individually update all elements contained in E_2.

Furthermore, in this example a_2 will be removed from $S_{N,q}$ once a_{14} arrives. To avoid update each element contained by E_8 individually due to the removal of a_2, we can keep a global value $P_{old}^{global} = P_{old}^{global} \times (1-P(a_2))$ at E_2 so that we know that the P_{old} values for elements in E_2 will be updated by multiplying $\frac{1}{P_{old}^{global}}$. From time to time, we may remove an entry E from $S_{N,q}$ and E fully dominates another entry E' which stays in $S_{N,q}$. If we keep the no-occurrence probability of the elements in E - $P_{noc} = \Pi_{a \in E}(1-P(a))$, then we can update P_{old}^{global} at E' by multiplying P_{noc}.

Aggregate Information. Motivated by the observation above, we maintain R_1 and R_2 as aggregate R-trees to keep the above information at each entry. We summarize it below.

- At each entry E, the following information will be stored. $P_{new}^{global}(E)$ stores the captured multiplication of non-occurrence probabilities of the elements which dominate all elements rooted at E. $P_{old}^{global}(E)$ stores the multiplication of non-occurrence probability of the elements that expired and dominate the elements rooted at E.
- At each entry E, we use $P_{noc}(E)$ to store $\prod_{e \in E}(1-P(e))$.
- At each entry E, $P_{sky,min}(E)$ and $P_{sky,max}(E)$ store the minimum skyline probability and maximum skyline probability of the elements rooted at E without including P_{old}^{global} and P_{new}^{global} at E. $P_{new,min}(E)$ and $P_{new,max}(E)$ store the minimum and maximum P_{new} values of the elements rooted at E without including P_{new}^{global} at E.

Example 4: Continue the example in Figure 3 against the first 13 elements.

P_{old}^{global} and P_{new}^{global} at each internal entry are initialized to 1. When a_{10} arrives, we update $P_{new}^{global}(E_4)$ from 1 to $(1-P(a_{10})) = 0.8$ since a_{10} dominates the MBB of E_4, while other P_{new}^{global} values remain 1.

Here, $P_{noc}(E_3) = (1-P(a_{10}))(1-P(a_8)) = 0.64$. Similarly, we can calculate values of P_{noc} at entries E_4, E_5, and E_6. Then, $P_{noc}(E_1) = P_{noc}(E_3) \times P_{noc}(E_4)$ and $P_{noc}(E_2) = P_{noc}(E_5) \times P_{noc}(E_6)$. The multiplication of $P_{noc}(E_1)$ and $P_{noc}(E_2)$ gives P_{noc} at the root. Similarly, P_{noc} values at each internal entry in R_2 can be calculated.

The information that a_{10} dominates both a_5 and a_6 has not been pushed down to leaf-level and is only captured at the entry E_4; consequently the captured skyline probabilities for a_6 and a_5 are $P(a_6) \times (1-P(a_8))$ (0.64) and $P(a_5)$ (0.8). Therefore, at E_4, $P_{sky,max} = 0.8$ and $P_{sky,min} = 0.64$; $P_{new,max} = 1$ and $P_{new,min} = (1-P(a_8))$ (0.8). These multiplied by P_{new}^{global} give the exact values of $P_{sky,max}$, $P_{sky,min}$, $P_{new,max}$, and $P_{new,min}$ at E_4, respectively. At other entries, $P_{sky,max}$, $P_{sky,min}$, $P_{new,max}$ and $P_{new,min}$ take exact values.

Once a_2 removes, at E_8, P_{old}^{global} is updated from 1 to $(1-P(a_2)) = 0.9$. □

Removing an Entry. When an entry E removes from R_1 or R_2, we first push down the aggregate information along the path from the root to E and update the siblings' aggregate information for each entry on the path. For example, when remove E_3, we first recalculate the max and min probabilities at the root by CalProb ($R.root$), Algorithm 2. Then we push-down P_{new} and P_{old} to E_1 and E_2, respectively by UpdateOldNew ($R_1.root$, E_1) and UpdateOldNew ($R_1.root$, E_2) (Algorithm 3). Then we reset P_{old}^{global} and P_{new}^{old} at $R.root$ by 1. We perform the same operations from E_1 to E_3 and E_4.

Algorithm 2: CalProb (E)

```
if P_old^global(E) < 1 then
    update P_sky,min(E), P_sky,max(E) by multiplying 1/P_old^global ;
end if
if P_new^global(E) < 1 then
    update P_sky,min(E), P_sky,max(E), P_new,min(E), P_new,max(E) by multiplying P_new^global ;
end if
```

Algorithm 3: UpdateOldNew (E, E')

```
if P_old^global(E) < 1 then
    P_old^global(E') := P_old^global(E') × P_old^global(E) ;
end if
if P_new^global(E) < 1 then
    P_new^global(E') := P_new^global(E') × P_new^global(E) ;
end if
```

After E removes from R, we recalculate min and max probabilities, as well as P_{noc} along the path in a bottom-up fashion from E.

Inserting an Entry. In our algorithm, we may need to remove an entry from R_1 and insert it to R_2, and vice versa. When an entry E inserts into R_1 (or R_2), we find an appropriate level to insert E; that is, the level with the length to the leaf to be the same as the depth of E. We also first push down the aggregate information, in the same way as a deletion, to the level. After inserting E, we also recalculate the same aggregate information in the same way as that in a deletion.

Re-balancing. When a re-balancing of R_1 or R_2 as an R-tree is called, we treat it as a deletion followed by an insertion.

B. Inserting a New Element

As depicted in the last subsection, once a new element a_{new} arrives, we need to conduct the following tasks: 1) update P_{new} values of the elements dominated by a_{new} by multiplying $(1 - P(a_{new}))$, 2) remove the elements a with updated $P_{new}(a) < q$ from R_1 and R_2, 3) update P_{sky} (via P_{old} and P_{new}) values for the elements dominated by some of those removed elements, 4) move elements a in R_1 with $P_{sky}(a) < q$ to R_2, and 5) calculate $P_{sky}(a_{new})$ and insert it to R_1 or R_2 accordingly since $P_{new}(a_{new}) = 1$.

According to Lemma 2, if a remaining element a in $S_{N,q}$ is dominated by a removed element a', then a' must be older than a; consequently in the task 3) above, we only need to update P_{old} values. Moreover, by dominance transitivity all the tasks 1) - 4) only need to be conducted against the elements dominated by a_{new}. Clearly, the task 5) is conducted against entries/elements which dominate a_{new}. Therefore, it is critical to identify entries/elements in R_1 and R_2 which are fully dominated by a_{new}, as well as the entries/elements which dominate a_{new}. Algorithm 4 is an outline of our techniques.

Algorithm 4: Inserting (a_{new})

Input : N: window size; q: skyline probability threshold. a_{new}: data element. R_1 and R_2: two aggregate trees on $SKY_{N,q}$ and $(S_{N,q} - SKY_{N,q})$ respectively.
Output : Updated R_1 and R_2

1 $P_{sky}(a_{new}) := P(a_{new})$; $P_{old}(a_{new}) := 1$; $P_{new}(a_{new}) := 1$;
2 **for each** $E \in \{R_1.root, R_2.root\}$ **do**
3 **if** $E \prec a_{new}$ **then**
4 $P_{sky}(a_{new}) := P_{sky}(a_{new}) \times P_{noc}(E)$;
5 $P_{old}(a_{new}) := P_{old}(a_{new}) \times P_{noc}(E)$;
6 **else**
7 **if** $a_{new} \prec E$ **then** add E to R;
8 **if** $E \prec_{partial} a_{new}$ & $a_{new} \prec_{not} E$ **then** add E to $C1$;
9 **if** $E \prec_{partial} a_{new}$ & $a_{new} \prec_{partial} E$ **then**
10 add E to $C12$;
11 **if** $a_{new} \prec_{partial} E$ & $E \prec_{not} a_{new}$ **then** add E to $C2$;
12 **if** $C1 \neq \emptyset$ **then** Probe ($C1$, $P_{sky}(a_{new})$);
13 **if** $C2 \neq \emptyset$ **then** Probe ($C2$, R);
14 **if** $C12 \neq \emptyset$ **then** Probe ($C12$, R, $P_{sky}(a_{new})$);
15 **if**$R \neq \emptyset$ **then** UpdateProb (R);
16 **if** $P_{sky}(a_{new}) \geq q$ **then** Add a_{new} to R_1 **else** add a_{new} to R_2

In Algorithm 4, we use $C1$ to store the entries partially dominate a_{new}, $C2$ to store the entries partially dominated by a_{new}, and $C12$ to store the entries which are partially dominated by a_{new} and partially dominate a_{new}. Then, we use Probe ($C1$, $P_{sky}(a_{new})$) and Probe ($C12$, R, $P_{sky}(a_{new})$) to traverse the two aggregate R-trees to get all entries/elements dominating a_{new}. We also use Probe ($C2$, R) and Probe ($C12$, R, $P_{sky}(a_{new})$) to traverse the two aggregate R-trees to get all entries/elements fully dominated by a_{new} and put in R. Finally, UpdateProb (R) conducts tasks 2)-4) and the task 5) is conducted in line 16 by the inserting operation to an aggregate R-tree (R_1 or R_2) as described in Section IV-A. Next, we provide details for the procedures Prob () and UpdateProb().

Probe ($C1$, $P_{sky}(a_{new})$) (Algorithm 5). According to Theorem 1, entries in C_1 cannot contain any element which is dominated by a_{new}. Probe ($C1$, $P_{sky}(a_{new})$) is to iteratively traverse the aggregate R-trees to get entries which dominate a_{new} and then update P_{sky} and P_{old} of a_{new}. In Algorithm 5, we use Dequeue () combining with UpdateOldNew () (Algorithm 3) to push down the aggregate information. Algorithm 6 gives details of Dequeue ().

Algorithm 5: Probe ($C1$, P_{sky})

1 **while** $C1 \neq \emptyset$ **do**
2 E := Dequeue ($C1$);
3 **for each** Children E' of E **do**
4 UpdateOldNew (E, E');
5 **if** $E' \prec a_{new}$ **then**
6 $P_{sky}(a_{new}) := P_{sky}(a_{new}) \times P_{noc}(E')$;
7 $P_{old}(a_{new}) := P_{old}(a_{new}) \times P_{noc}(E')$;
8 **else**
9 **if** $E' \prec_{partial} a_{new}$ **then** add E' to $C1$;
10 **if** E' is the last child of E **then**
11 reset $P_{new}^{global}(E)$ and $P_{old}^{global}(E)$ to 1;
12 return $P_{sky}(a_{new})$;

Algorithm 6: Dequeue ($C1$)

1 **if** $C1 \neq \emptyset$ **then**
2 get an E in $C1$; /* remove E from $C1$ */;
3 CalProb (E); /* Algorithm 2 */;
4 return E;

Probe ($C2$, R). Note that entries in $C2$ do not contain any elements that dominate a_{new} according to Theorem 1. Similarly, Probe ($C2$, R) is to iteratively traverse to get all entries/elements which are dominated by a_{new} and then place them in R. As a by-product, we push down the aggregate information and update P_{new}^{global} values of those entries/elements in R. The details are presented in Algorithm 7.

Probe ($C12$, R, $P_{sky}(a_{new})$). Entries in $C12$ partially dominate a_{new} and are also partially dominated by a_{new}. Consequently, elements contained by entries in $C12$ might dominate a_{new} or are dominated by a_{new}. Probe ($C12$, R, $P_{sky}(a_{new})$), combing with Algorithms 5 and 7, is to iteratively traverse the aggregate R-trees to possibly further update $P_{sky}(a_{new})$ and add more to R. We present the details below in Algorithm 8.

UpdateProb (R). R contains all entries/elements which are fully dominated by a_{new} and obtained by Probe ($C12$, R, $P_{sky}(a_{new})$) and Probe ($C2$, R). Note that in our implemen-

Algorithm 7: Probe ($C2$, R)

```
while C2 ≠ ∅ do
    E := Dequeue (C2);
    for each Children E' of E do
        UpdateOldNew (E');
        if a_new ≺ E' then
            P_new^global(E') := (1 − P(a_new)) × P_new^global(E');
            add E' to R;
        else
            if a_new ≺_partial E' then add E' to C2;
        if E' is the last child of E then
            reset P_new^global(E) and P_old^global(E) to 1;
return R;
```

Algorithm 8: Probe ($C12$, R, $P_{sky}(a_{new})$)

```
while C12 ≠ ∅ do
    E := Dequeue (C12);
    for each Children E' of E do
        UpdateOldNew (E');
        if a_new ≺ E' then
            P_new^global(E') := (1 − P(a_new)) × P_new^global(E');
            add E' to R;
        else
            if a_new ≺_partial E' & E' ≺_not a_new then
                add E' to C2;
            if a_new ≺_not E' & E' ≺_partial a_new then
                add E' to C1;
            if a_new ≺_partial E' & E' ≺_partial a_new then
                add E' to C12;
            if E' ≺ a_new then
                P_sky(a_new) := P_sky(a_new) × P_noc(E') ;
                P_old(a_new) := P_old(a_new) × P_noc(E') ;
        if E' is the last child of E then
            reset P_new^global(E) and P_old^global(E) to 1;
if C1 ≠ ∅ then Probe (C1, P_sky) (Algorithm 5);
else return P_sky(a_new);
if C2 ≠ ∅ then Probe (C2, R) (Algorithm 7);
else return R;
```

tation, we use a link list to point to all these entries/elements in R. UpdateProb (R) is to traverse those entries in R, along the aggregate R-trees to which they belong, to detect and remove entries/elements with the updated P_{new} values smaller than q. Moreover, it also updates the P_{old} values of remaining elements in R which are dominated by some removed elements, as well as detects the remaining elements in R with $P_{sky} < q$. Algorithm 9 provides details.

Lines 1-11: Iteratively detect the elements/entries to be removed (i.e. with $P_{new} < \theta$) and put them to R_3.

Lines 12: UpdateOld (R_3, R_4) is to update the values of P_{old}^{global} of elements/entries in R_4 dominated by some in R_3 as follows. For each pair $E1 \in R_3$ and $E2 \in R_4$,

> if $E1$ fully dominates $E2$, then update $P_{old}^{global}(E2)$ by multiplying $P_{noc}(E1)$; otherwise, if $E1$ partially dominates $E2$ then put the children of $E1$ to R_3 and the children of $E2$ to R_4 for the next iteration.

In our implementation, we mark entries from R (i.e., R_3 and R_4) within R_1 and R_2. Then, we use the *synchronous traversal* paradigm [11] to traverse R_3 and R_4 by following the R-tree structures of the entries in R_3 and R_4. Here, we create a dummy root for R_3 with all entries in R_3 to be children of the root; similar treatments are done for R_4.

Algorithm 9: UpdateProb (R)

```
1  while R ≠ ∅ do
2      E := Dequeue (R);
3      if P_new,min(E) < q ≤ P_new,max(E) then
4          for each Children E' of E do
5              UpdateOldNew (E', E);
6              add E' to R;
7              if E' is the last child of E then
8                  reset P_new^global(E) and P_old^global(E) to 1;
9      else
10         if P_new,min(E) ≥ q then add E to R_4;
11         else add E to R_3;            /* P_new,max(E) < q */;
12 if R_3 ≠ ∅ and R_4 ≠ ∅ then UpdateOld (R_3, R_4);
13 if R_3 ≠ ∅ then Remove (R_3);
14 if R_4 ≠ ∅ then Place (R_4);
```

lines 13: We remove entries/elements in R_3 from R_1 and R_2 as what discussed in Section IV-A.

lines 14: Place (R_4) is to determine elements/entries in R_4 to be in R_1 or R_2. In fact, we only need to check $R_4 \cap R_1$ according to Corollaries 1 and 2; it is conducted as follows. For each entry $E \in R_4 \cap R_1$, we use depth-first search to find out all its highest level decedent entries with $P_{sky,min}$ greater than q - Algorithm 10. In lines 10-11 of Algorithm 10, we first remove E from R_1 in the way as described in Section IV-A. Then, we insert E into R_2 in the way as described in Section IV-A.

Algorithm 10: Place (R_4)

```
1  while R_1 ∩ R_4 ≠ ∅ do
2      E := Dequeue (R_1 ∩ R_4);
3      if P_sky,min(E) < q ≤ P_sky,max then
4          for each Children E' of E do
5              UpdateOldNew (E');
6              add E' to R_1 ∩ R_4;
7              if E' is the last child of E then
8                  reset P_new^global(E) and P_old^global(E) to 1;
9      else
10         if P_sky,max(E) < q then
11             Move E from R_1 to R_2;
```

C. Expiration

Once an element a_{old} expires, we first check if it is in $S_{N,q}$. If it is in $S_{N,q}$ then we need to increase the P_{old} values for elements dominated by a_{old}. After that, we need to determine the elements that need to be moved from R_2 to R_1. Algorithm 11 below presents details.

In Algorithm 11, Move ($R \cap R_2$) is to move the elements in $R \cap R_2$ with updated skyline probability not smaller than q to R_1. It is executed in the same way as Place (R_4) but replace $R_1 \cap R_4$ by $R \cap R_2$ and move from R_2 to R_1 instead of R_1 to R_2.

Algorithm 11: Expiring (a_{old})

```
1  if a_old ∈ S_{N,q} then
2      Remove (a_old);
3      for E ∈ {R_1.root, R_2.root} do
4          if a_old ≺ E then
5              P_old(E) = P_old(E)/(1 − P(a_old));
6              add E to R;
7          else
8              if a_old ≺_partial E then add E to C;
9      while C ≠ ∅ do
10         E := Dequeue (C);
11         for each Children E' of E do
12             UpdateOldNew (E');
13             if a_old ≺ E' then
14                 P_old(E') := P_old(E')/(1 − P(a_old));
15                 add E' to R;
16             else
17                 if a_old ≺_partial E then add E' to C;
18             if E' is the last child of E then
19                 reset P_new^global(E) and P_old^global(E) to 1;
20     if R ≠ ∅ then Move (R ∩ R_2);
```

D. Multiple Confidences

Continuous queries Different users may specify different confidences. Suppose that users specify k confidences q_1, q_2, ..., q_k where $q_i < q_{i-1}$. Our techniques for a single given confidence can be immediately extended to cover multiple confidences as follows.

Instead of maintaining a single solution set R_1 in Algorithm 11, we maintain k solution sets R_1, R_2, ..., R_k such that elements in R_i (for $2 \leq i \leq k$) have the skyline probabilities in $[q_i, q_{i-1})$ where $q_0 = 1$ and R_{k+1} keeps the elements in $(S_{N,q_k} - \cup_{i=1}^{k} R_i)$. Those R_i for $i = 1$ to $(k+1)$ are also maintained as aggregate R-trees with the same aggregate information.

All the techniques from Algorithm 11 are immediately applicable except that now in Algorithm 9, we need to detect where to place some elements in $R \cap R_i$ for $i \leq k$; that is, we need to consider all R_j for $i < j < k+1$. In Algorithm 11, now we need to detect where to move some elements in R_{k+1}; that is, we need to consider R_j (for $1 \leq j \leq k$) instead of just R_1 in the case of single confidence.

Ad-hoc Queries. Users may also issue an ad-hoc query, "find the skyline with skyline probability at least q'". Assume that currently we maintain k skylines as discussed above and $q' \geq q_k$. Then, we first find an R_i such that $q_i \leq q' < q_{i-1}$; clearly elements $\{R_j\colon j < i-1\}$ } are contained in the solution. We can apply the search paradigm in Place (R_4) (Algorithm 10) to get all elements in R_i with skyline probabilities $\geq q$ but without updating aggregate probabilities information.

E. Algorithm Analysis

Correctness. Our sliding window techniques maintain aggregate information against $S_{N,q}$ and then get skyline according to the skyline probabilities restricted to $S_{N,q}$, Theorems, Lemmas and Corollaries in Section III-A ensure that our algorithms are correct.

Space Complexity. Clearly, in our algorithm we use aggregate-R trees to keep each element in $S_{N,q}$ and each element is kept only once. Thus, the space complexity is $O(|S_{N,q}|)$.

Time Complexity. It seems hard to provide a sensible time complexity analysis; nevertheless, our experiment demonstrates the algorithms in this section is much faster than the trivial algorithm against $S_{N,q}$ as what discussed in the beginning of this section.

V. Performance Evaluation

In this section, we only evaluate our techniques since this is the first paper studying the problem of probabilistic skyline computation over sliding windows. Specifically, we implement and evaluate the following techniques.

- SSKY Techniques presented in Section IV to continuously compute q-skyline (i.e., skyline with the probability not less than a given q) against a sliding window.
- MSKY Techniques in Section IV-D to continuously computing multiple q-skylines currently regarding multiple given probability thresholds.
- QSKY Techniques in Section IV-D to processing an ad-hoc skyline query with a probability threshold.

All algorithms are implemented in C++ and compiled by GNU GCC. Experiments are conducted on PCs with Intel Xeon 2.4GHz dual CPU and 4G memory under Debian Linux. Our experiments are conducted on both real and synthetic datasets.

Real dataset is extracted from the stock statistics from NYSE (New York Stock Exchange). We choose 2 million stock transaction records of Dell Inc. from *Dec 1st 2000* to *May 22nd 2001*. For each transaction, the average price per volume and total volume are recorded. This 2-dimensional dataset is referred to as *stock* in the following. We randomly assign a probability value to each transaction; that is, probability values follows *uniform* distribution. Elements' arrival order is based on their transaction time.

Synthetic datasets are generated as follows. We first use the methodologies in [3] to generate 2 million data elements with the dimensionality from 2 to 5 and the spatial location of data elements follow two kinds of distributions, *independent* and *anti-correlated*. Then, we use two models *uniform* or *normal* distributions to randomly assign occurrence probability of each element to make them be uncertain. In *uniform* distribution, the occurrences probability of each element takes a random value between 0 and 1, while in the *normal* distribution, the mean value P_μ varies from 0.1 to 0.9 and standard deviation S_d is set 0.3. We assign a random order for elements' arrival in a data stream.

Choosing q. q is the probability threshold in evaluating efficiency of query processing. To evaluate SSKY, we use 0.3 as a default value of q, while to evaluate MSKY with k given probability thresholds q_1, ..., q_k, we let these k values evenly spread $[0.3, 1]$. To evaluate QSKY, we issue 1000 queries across $[q, 1]$ where q is the minimum probability threshold when multiple thresholds are pre-given for multiple

continuous skylines. We record average time to process these 1000 queries.

Table II summarizes parameters and corresponding default values. **In our experiments, all parameters take default values unless otherwise specified.**

TABLE II
SYSTEM PARAMETERS

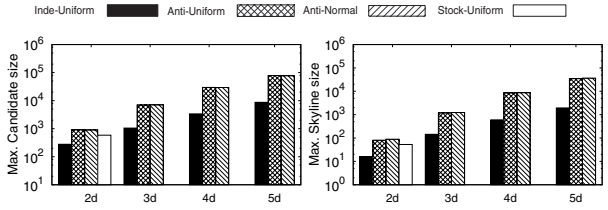
Notation	Definition (Default Values)
n	Number of points in the dataset (2M)
N	Sliding Window size (1M)
d	Dimensionality of the of the dataset (3)
D	Dataset (Anti)
D_P	Probabilistic distribution of appearance (*uniform*)
P_μ	expected appearance probability (0.5)
q	probabilistic threshold (0.3)
q'	probabilistic threshold q' ($q \leq q' \leq 1$)

In our experiments, we evaluate the efficiency of our algorithm as well as space usage against dimensionality, size of sliding window, probabilistic threshold, distribution of objects' spatial location and appearance probability distribution.

A. Evaluate Space Efficiency

We evaluate the space usage in terms of the number of uncertain elements kept in $S_{N,q}$ against different settings. As this number may change as the window slides, we record the maximal value over the whole stream. Meanwhile, we also keep the maximal number of $SKY_{N,q}$.

The first set of experiments is reported in Figure 4 where 4 datasets are used: Inde-Uniform (Independent distribution for spatial locations and Uniform distribution for occurrence probability values), Anti-Uniform, Anti-Normal, and Stock-Uniform. We record the maximum sizes of $S_{N,q}$ and $SKY_{N,q}$. It is shown that very small portion of the 2-dimensional dataset needs to be kept. Although this proportion increases with the dimensionality rapidly, our algorithm can still achieve a 89% space saving even in the worst case, 5 dimensional *anti-correlated* data. Size of $SKY_{N,q}$ is much smaller than that of candidates. Since the *anti-correlated* dataset is the most challenging, it will be employed as the default dataset in the following.

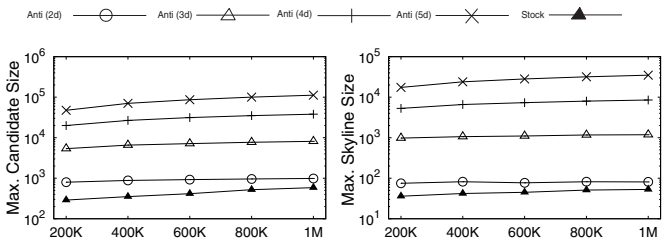


(a) Max. Candidate Size (b) Max. Skyline Size

Fig. 4. Space Usage vs Diff. Data set

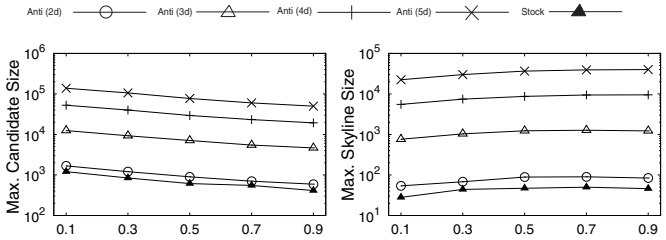
The second set of experiment evaluates the impact of sliding window size N on the space efficiency. As depicted in Figure 5, the space usage is sensitive towards the increment of window size.

Figure 6 reports the impact of occurrence probability distribution against the space usage and number of skyline points on different datasets. The occurrence probability follows *normal* distribution and the mean of the appearance probability P_μ



(a) Max. Candidate Size(*uniform*)) (b) Max. Skyline Size (*uniform*)

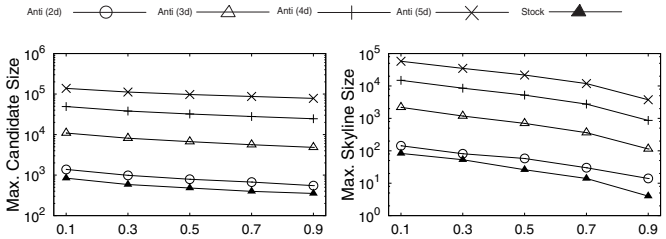
Fig. 5. Space Usage vs Window Size



(a) Max. Candidate Size (b) Max. Skyline Size

Fig. 6. Space Usage vs Appearance Probability

increases from 0.1 to 0.9. It demonstrates that the smaller the average appearance probability of the points, the more points will be kept in $S_{N,q}$. As shown in Figure 6(a), the size of the candidate decreases with the increase of average appearance probability. Interestingly, although the candidate size is large with smaller average occurrence probability, the number of probabilistic skyline is small, as illustrated in Figure 6(b). This is because the small occurrence probability prevents the uncertain objects from becoming probabilistic skyline.



(a) Max. Candidate Size(*uniform*) (b) Max. Skyline Size (*uniform*)

Fig. 7. Space Usage vs Probability threshold

Figure 7 reports the effect of probabilistic threshold q on space efficiency. As expected, both candidate set size and skyline set size drop as q increases.

B. Evaluation Time Efficiency

We evaluate the time efficiency of our continuous query processing techniques, SSKY and MSKY, as well as ad-hoc query processing technique QSKY. We first compare SSKY with the trivial algorithm against $SKY_{N,q}$ as described in the beginning of Section IV. We find it is about 20 times slower than SSKY against anti (3d). Thus, we exclude the trivial algorithm from further evaluation.

Since the processing time of one element is too short to capture precisely, we record the average time for each batch of 1K elements to estimate the delay per element.

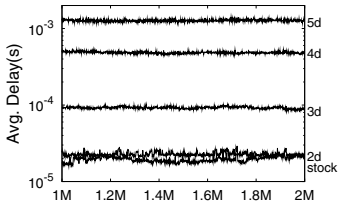


Fig. 8. Time Efficiency vs n

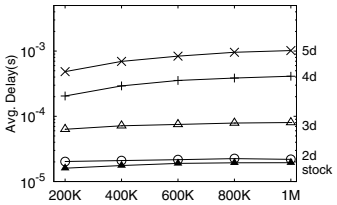


Fig. 9. Avg. Delay vs W

The first set of experiment is depicted in Figure 8. It shows that SSKY is very efficient, especially when the dimensionality is low. For 2 dimensional dataset, SSKY can support a workload where elements arrive at the speed of more than 38K per second even for *stock* and *anti-correlated* dataset. For $5d$ *anti-correlated* data, our algorithm can still support up to 728 elements per second, which is a medium speed for data streams.

Figure 9 evaluates the system scalability towards the size of the sliding window. The performance of SSKY is not sensitive to the size of sliding window. This is because the candidate size increases slowly with N, as reported in Figure 5.

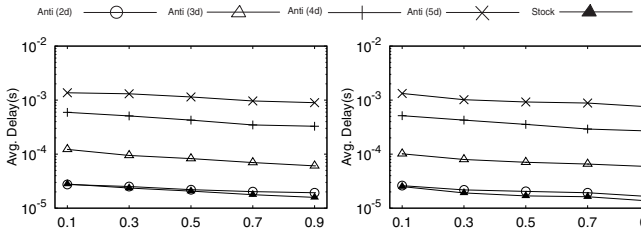


Fig. 10. Avg. Delay vs P_μ

Fig. 11. Avg. Delay vs q

Figure 10 evaluates the impact of occurrence probability distribution on time efficiency of SSKY where normal distribution is used for probability values. As expected, large P_μ leads to better performance since the candidate size is small when P_μ is large.

Figure 11 evaluates the effect of probability threshold q on SSKY. Since both size of candidate set and skyline objects set are small when q is large as depicted in Figure 7, SSKY is more efficient when q increases.

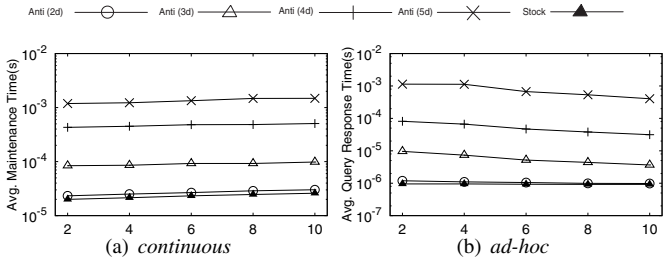


Fig. 12. Query Cost vs $|Q|$

The last experiment evaluates the efficiency of our multi probability thresholds based continuous query processing techniques MSKY and ad-hoc query processing techniques. Results are reported in Figures 12(a) and 12(b), respectively. As expected, Figure 12(a) shows that cost to process each element by MSKY increases when k increases, while Figure 12(b) shows the ad-hoc query processing cost decreases when k increases.

C. Summary

As a short summary, our performance evaluation indicates that we only need to keep a small portion of stream objects in order to compute the probabilistic skyline over sliding windows. Moreover, our continuous query processing algorithms are very efficient and can support data streams with high speed for 2d and 3d datasets. Even for the most challenging data distribution, *anti-correlated*, we can still support the data stream with medium speed of more than 700 elements per second when dimensionality is 5.

VI. Applications

The techniques developed in this paper can be immediately extended to the following applications.

Probabilistic Top-k Skyline Elements. Given an uncertain data stream, a threshold q, and a sliding window size W, find the k skyline points with the highest skyline probabilities (but not smaller than q).

We can apply our algorithms in Section IV to remove points with $P_{new} < q$, update aggregate information at each entry, probabilities (P_{sky}, P_{old}, P_{new}, etc). We do not move any elements in $R_4 \cap R_1$ to R_2. Instead, we treat R_1 and R_2 as two "heap trees". In fact, both R_1 and R_2 maintain two heaps on P_{sky}: 1) min-heap, and 2) max-heap; this is because we keep $P_{sky,min}$ and $P_{sky,max}$ at each entry. We use min-heap on R_1 and max-heap on R_2 to move elements in top-k from R_2 to R_1 and move elements in R_1 but not in top-k to R_2.

Time Stamp based Sliding Windows. In such a model, we expire an old element if it is not within a pre-given most recent time period T. Our techniques can be immediately extended to sliding windows based on the most recent time period T.

Object with Multiple Elements. Suppose that an uncertain stream contains a sequence of objects such that each object consists of a set of instances [22] or PDF. In fact, our skyline probability model is a special case of the model in [22]. In our sliding window model, we assume that each object is *atomic*.[1] Then we want to compute objects with skyline probabilities not smaller than q. It can be immediately verified that all our techniques are immediately applicable to discrete cases except we compute skyline probability in a different way; that is, based on the definition in [22]. For continuous cases, we can use Monte-Carlo sampling method [16] to discrete them.

VII. Related Work

We review related work in two aspects, skylines and uncertain data streams. To the best of our knowledge, this paper is the first one to address the problem of skyline queries on uncertain data streams.

Skylines. Börzsönyi *et al* [3] first study the skyline operator in the context of databases and propose an SQL

[1] When an object arrives, all its instances arrive; when an object expires, all its instances expire.

syntax for the skyline query. They also develop two computation techniques based on *block-nested-loop* and *divide-and-conquer* paradigms, respectively. Another *block-nested-loop* based technique SFS (*sort-filter-skyline*) is proposed by Chomicki *et al* [7], which takes advantage of a pre-sorting step. SFS is then significantly improved by Godfrey *et al* [10]. The *progressive* paradigm that aims to output skyline points without scanning the whole dataset is firstly proposed by Tan *et al* [24]. It is supported by two auxiliary data structures, *bitmap* and *search tree*. Kossmann *et al* [18] present another progressive technique based on the nearest neighbor search technique. Papadias *et al* [21] develop a *branch-and-bound* algorithm (BBS) to progressively output skyline points based on R-trees with the guarantee of minimal I/O cost. Variations of the skyline operator have also been extensively explored, including skylines in a distributed environment [2], [12], skylines for partially-ordered value domains [4], skyline cubes [23], [26], [27], reverse skylines [9], approximate skylines [5], [6], [17], etc.

Skyline queries processing in data streams is investigated by Lin *et al* [20] against various sliding windows. Tao *et al* [25] independently develop efficient techniques to compute sliding window skylines.

The skyline query processing on uncertain data is firstly approached by Pei *et al* [22] where *Bounding-pruning-refining* techniques are developed for efficient computation. Lian *et al* [19] combine reverse skylines [9] with uncertain semantics and model the *probabilistic reverse skyline* query in both monochromatic and bichromatic fashion. Efficient pruning techniques are developed to reduce the search space for query processing.

Uncertain Data Streams. Although numerous research aspects have been addressed on managing certain stream data, works on uncertain data streams have abounded only very recently. Aggregates over uncertain data streams have been studied recently [8], [13], [14]. Problems such as clustering uncertain data stream [1], frequent items retrieval in probabilistic data streams [28], and sliding window top-k queries on uncertain streams [15] are also investigated. Since skyline queries are inherently different from these problems, techniques proposed in none of the above papers can be applied directly to the problems studied in this paper.

VIII. Conclusion

In this paper, we investigate the problem of efficiently computing skyline against sliding windows over an uncertain data stream. We first model the probability threshold based skyline problem. Then, we present a framework which is based on efficiently maintaining a candidate set. We show that such a candidate set is the minimum information we need to keep. Efficient techniques have been presented to process continuous queries. We extend our techniques to concurrently support processing a set of continuous queries with different thresholds, as well as to process an ad-hoc skyline query. Finally, we show that our techniques can also be extended to support probabilistic top-k skyline against sliding windows over an uncertain data streams. Our extensive experiments demonstrate that our techniques can deal with a high-speed data stream in real time.

Acknowledgement. The work of Xuemin Lin and Ying Zhang was partially supported by ARC Grant (DP0881035, DP0666428 and DP0987557) and a Google Research Award. The work of Wei Wang is partially supported by ARC grant (DP0881779). The work of Jeffrey Xu Yu was supported by a grant of RGC, Hong Kong SAR, China (No. 419008)

References

[1] C. C. Aggarwal and P. S. Yu. A framework for clustering uncertain data streams. In *ICDE 2008*.
[2] W.-T. Balke, U. Guntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT 2004*.
[3] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE 2001*.
[4] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with paritally ordered domains. In *SIGMOD 2005*.
[5] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, and A. K. H. Tung. On high dimensional skylines. In *EDBT 2006*.
[6] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD 2006*.
[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE 2003*.
[8] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD 2007*.
[9] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB 2007*.
[10] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB 2005*.
[11] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Spatial joins using r-trees: Breadth-first traversal with global optimizations. In *VLDB 1997*.
[12] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in MANETs. In *ICDE 2006*.
[13] T. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *SODA 2007*.
[14] T. S. Jayram, A. McGregor, S. Muthukrishan, and E. Vee. Estimating statistical aggregrates on probabilistic data streams. In *PODS 2007*.
[15] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. In *VLDB 2008*.
[16] M. H. Kalos and P. A. Whitlock. *Monte Carlo Methods.* Wiley Interscience, 1986.
[17] V. Koltun and C. Papadimitriou. Approximately dominating representatives. In *ICDT 2005*.
[18] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB 2002*.
[19] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD 2008*.
[20] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE 2005*.
[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal progressive algorithm for skyline queries. In *SIGMOD 2003*.
[22] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB 2007*.
[23] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skylin: A semantic approach based on decisive subspaces. In *VLDB 2005*.
[24] K.-L. Tan, P. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB 2001*.
[25] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. In *TKDE 2006*.
[26] T. Xia and D. Zhang. Refreshing the sky: The compressed skycube with efficient support for frequent updates. In *SIGMOD 2006*.
[27] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. Y. Xu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB 2005*.
[28] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *SIGMOD 2008*.