Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Probabilistic skyline operator over sliding windows

Wenjie Zhang [a,*], Xuemin Lin [a], Ying Zhang [a], Wei Wang [a], Gaoping Zhu [a], Jeffrey Xu Yu [b]

[a] University of New South Wales, Australia
[b] Chinese University of Hong Kong, Hong Kong

ARTICLE INFO

Keywords:
Uncertain
Stream
Skyline

ABSTRACT

Skyline computation has many applications including multi-criteria decision making. In this paper, we study the problem of efficiently computing the skyline over sliding windows on uncertain data elements against probability thresholds. Firstly, we characterize the properties of elements to be kept in our computation. Then, we show the size of dynamically maintained candidate set and the size of skyline. Novel, efficient techniques are developed to process continuous probabilistic skyline queries over sliding windows. Finally, we extend our techniques to cover the applications where multiple probability thresholds are given, "top-k" skyline data objects are retrieved, or elements have individual life-spans. Our extensive experiments demonstrate that the proposed techniques are very efficient and can handle a high-speed data stream in real time.

1. Introduction

Uncertain data analysis is a key in many important applications, such as sensor networks, trend prediction, moving object management, data cleaning and integration, economic decision making, and market surveillance. In such applications, uncertain data is often collected in a streaming fashion. Uncertain streaming data computation has drawn considerable attention from the database research community recently (e.g. [9,15,30]).

Skyline analysis is shown as a very useful tool [4,8,23,26] in multi-criterion decision making. Given a certain dataset 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 according to the preferences specified by users. The skyline of 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 [2,21,24]. 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.

In many online monitoring problems, the appearance of a data element is often uncertain. Below are two examples. In an online shopping system, products are evaluated in various aspects such as *price*, *condition* (e.g., brand new, excellent, good, average, etc.), and *brand*. A customer may want to select a product, say laptops, based on the multiple criteria (preferences) such as low price, good condition, and good brand. It is well known [4] that the skyline provides a candidate set of best deals. In the application, each seller is also associated with a "trustability" value which is derived from customers' feedback on the seller's product quality, delivery handling, etc.; the trustability value may be regarded as the "appearance" 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. For simplicity, we assume that a customer only prefers ThinkPad T61; thus we remove the brand dimension from ranking. Table 1 lists four qualified results. Both L_1 and L_4 are skyline points regarding (price, condition), L_1 is better than (dominates) L_2, and L_4 is better

* Corresponding author. Tel.: +61 293854025.
E-mail addresses: zhangw@cse.unsw.edu.au (W. Zhang), lxue@cse.unsw.edu.au (X. Lin), yingz@cse.unsw.edu.au (Y. Zhang), weiw@cse.unsw.edu.au (W. Wang), gzhu@cse.unsw.edu.au (G. Zhu), yu@se.cuhk.hk (J. Xu Yu).

0306-4379/$ - see front matter © 2012 Published by Elsevier Ltd.
http://dx.doi.org/10.1016/j.is.2012.03.002

Table 1
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

than L_3. Nevertheless, L_1 is posted long time ago, and the trustability of L_4 is quite low. In such applications, customers may want to continuously monitor online 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 could model such an online selection problem as probabilistic skyline against sliding windows by treating online advertisements as an uncertain data stream (see Section 2 for details) such that each data element (advertisement) has an occurrence probability.

An uncertain data stream may arrive at a very high speed. Consider stock market applications where clients may be eager to buy a particular stock and want to online monitor good offers (for sale) from other clients for this particular stock. An offer is recorded by two aspects (price, volume) where price is the price per share in the offer and volume is the number of shares offered for sale. In such applications, customers may want to know the top offers so far, as one of many kinds of statistic information, before making trade decisions. An offer a is better than another deal b if a involves a higher volume and is cheaper (per share) than those of b, respectively. Nevertheless, an offer from a client may be withdrawn from time to time; thus, it also has a probability to exist (i.e. has an occurrence probability). Consequently, a stream of sale offers may be treated as a stream of uncertain elements such that each element has a probability to occur. Clearly, some clients may only want to know "top" offers (skyline) among the most recent N offers (sliding windows) or the offers made in the most recent T period; this, together with the consideration of the uncertainty of each deal gives another example of probabilistic skyline against sliding windows.

While the two examples above demonstrate the usefulness of online monitoring skyline over uncertain data, online monitoring skyline over uncertain streaming data regarding sliding windows has many other applications. In this paper we investigate the problem of efficiently monitoring 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 online, 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 poly-logarithmic sizes 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, "top-k" probabilistic skyline data elements, and data elements with individual life-spans.

Our extensive experiments demonstrate that the developed techniques can support online computation against very rapid data streams.

The rest of the paper is organized as follows. In Section 2, we formally define the problem of sliding-window skyline computation on uncertain data streams and present background information. Sections 3 and 4 present the framework, fundamental theories, and techniques for processing probability threshold based sliding window queries. Section 5 extends our techniques to multi-thresholds, top-k skyline, and time-based sliding windows where each data element has a life-span. Results of comprehensive performance studies are discussed in Section 6. Section 7 summarizes related work and Section 8 concludes the paper.

2. 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 $P(a)$ $(0<P(a)\le 1)$ to occur where $a.i$ (for $1\le i\le d$) denotes the i-th dimension value. Without loss of generality, we assume that on each dimension, users prefer small values. For two elements u and v, u *dominates* v, denoted by $u \prec v$, if $u.i \le v.i$ for $1\le i\le d$, 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.

2.1. 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 $P_{sky}(a)=\sum_{W\in\Omega, a\in SKY(W)}P(W)$. $P_{sky}(a)$ is called the *skyline probability* of a. Eq. (1) below can be immediately verified.

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

Problem statement. In many applications, a data stream DS is *append-only* [16,20,27]; 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. Specifically, 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 $(0<q\le 1)$; that is, q-skyline. We focus on developing techniques to efficiently process such a *continuous* query. We will also study some variants of the problem. Note that although the data streams studied in the paper are append-only, we need to

deal with the deletion of data elements due to the elements expiration from a sliding window.

2.2. Preliminaries

In a DS, elements are positioned according to their relative arrival ordering and labeled by integers. Note that the position/label $\kappa(a)$ means that the element a arrives $\kappa(a)$th in the data stream.

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 elements arriving later than a dominates a; that is,

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

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 elements arriving earlier than a dominates a; that is,

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

Note that $P_{new}(a) = 1$ if $\nexists a' \in DS_N$ such that $a' \prec a$ and $\kappa(a') > \kappa(a)$, and $P_{new}(a) = 0$ if $\exists a' \in DS_N$ such that $a' \prec a$, $\kappa(a') > \kappa(a)$, and $P(a') = 1$. Similarly, $P_{old}(a) = 1$ if $\nexists a' \in DS_N$ such that $a' \prec a$ and $\kappa(a') < \kappa(a)$, and $P_{old}(a) = 0$ if $\exists a' \in DS_N$ such that $a' \prec a$, $\kappa(a') < \kappa(a)$, and $P(a') = 1$. The following Eq. (4) can be immediately derived from (1).

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

Example 1. Regarding the example in Fig. 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 $P_{sky}(a_4) = P(a_4) \times P_{new}(a_4) \times P_{old}(a_4) = 0.034$. □

Dominance relationships. Our techniques are developed against an in-memory spatial index. In this paper, we adopt in-memory R-trees though our techniques may be immediately applied to other spatial index including the grid index structure [22]. 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 dominate E'; this is denoted by $E \prec_{partial} E'$. Otherwise, E does *not dominate* E', denoted by $E \prec_{not} E'$.

Regarding the example in Fig. 1(c), immediately $E \prec E_3$, $E \prec_{partial} E_1$, $E \prec_{partial} E_2$. Note that E_1 does not dominate E but $E_2 \prec_{partial} E$. Clearly, some elements in E_1 might 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. *If $E \prec_{not} E'$, then none of the elements in E' can be dominated by any element in E. If $E'' \prec E'''$, then every element in E'' dominates any element in E'''.*

Table 2 below summarizes the mathematic notations used throughout the paper.

3. Framework

A critical requirement in data stream computation is to have small memory space and fast computation. Consequently, given a probability threshold q and a sliding window with length N, it would be ideal if the continuous computation can be conducted against the probabilistic q-skyline $SKY_{N,q}$ of DS_N by excluding all the other elements.

Table 2
Summary of math notations.

Notation	Definition
DS	Stream of uncertain data elements
DS_N	Sliding window of the most recent N elements
$DS_{T,t}$	Sliding window of the most recent time period T
a	Data element
$a_{new}(a_{old})$	Newly (oldest) arrived element in DS_N
$P(a)$	Occurrence probability of a
$P_{new}(a)(P_{old}(a))$	Probability that none of the elements in DS_N arriving later (earlier) than a dominates a
$P_{sky}(a)$	Skyline probability of a in DS_N
q	Given probability threshold q
$SKY_{N,q}$	q-Skyline of DS_N
$SKY_{T,t,q}$	q-Skyline of $DS_{T,t}$
$S_{N,q}$	Set of elements a in DS_N with $P_{new}(a) \geq q$
R	Aggregate R-tree
E	Entry in R
$E_{min}(E_{max})$	Lower-left (upper-right) corner of MBB of E
$P_{noc}(E)$	Multiplication of non-occurrence probabilities of the elements in E
$\kappa(a)$	Position or time-stamp of a
$l(a)$	Life-span of an element a

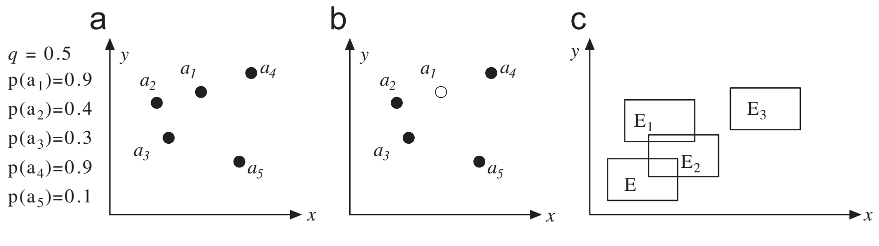

Fig. 1. Examples.

However, this is impossible. For instance, regarding the example in Fig. 1(a), assume that $N=5$ and $q=0.5$. It is immediate that $P_{sky}(a_4)=0.0342<q$; that is, $a_4 \notin SKY_{N,q}$ regarding the first 5 elements. If we do not keep a_4 and none of the newly coming elements dominates a_4, then we miss a_4 as q-skyline element after a_1 to a_3 expire from the sliding window.

In our algorithm, we continuously maintain a subset $S_{N,q}$ of DS_N with the following property.

1. $SKY_{N,q} \subseteq S_{N,q}$.
2. For each element $a \in DS_N$, the problem of determining if $a \in S_{N,q}$ against DS_N is equivalent to the problem of determining if $a \in S_{N,q}$ against $S_{N,q}$ only.
3. For each element $a \in S_{N,q}$, if the skyline probability of a restricted to $S_{N,q}$ is not smaller than the given threshold q, then the skyline probability of a regarding the sliding window must not be smaller than q.

The first property ensures that the q-skyline of DS_N will be covered by $S_{N,q}$, while the second property ensures that we only need to focus on $S_{N,q}$ when continuously deciding $S_{N,q}$. Note that the skyline probability of an element a restricted to $S_{N,q}$ is not smaller than the skyline probability of a regarding the sliding window since $S_{N,q} \subseteq DS_N$; subsequently there is no false negative to determine the skyline element against $S_{N,q}$. The third property ensures that there is no false positive to determine the q-skyline element against $S_{N,q}$.

In our algorithm, we propose to maintain the following subset $S_{N,q}$ of DS_N such that each element in $S_{N,q}$ has $P_{new} \geq q$; that is,

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

The framework of continuously computing the probabilistic q-skyline $SKY_{N,q}$ of DS_N is outlined below in Algorithm 1 where a_{old} and a_{new} are the oldest element and newly arrived element, respectively, in the current window DS_N. Inserting (a_{new},$S_{N,q}$,$SKY_{N,q}$) and Expiring (a_{old},$S_{N,q}$,$SKY_{N,q}$) incrementally computes $SKY_{N,q}$. The details of Inserting (a_{new},$S_{N,q}$,$SKY_{N,q}$) and Expiring (a_{old},$S_{N,q}$, $SKY_{N,q}$) will be introduced in Section 4.

Algorithm 1. Continuously computing probabilistic skyline.

```
1   while a new element $a_{new}$ arrives do
2       if $\kappa(a_{new}) \leq N$ then Inserting ($a_{new}$,$S_{N,q}$,$SKY_{N,q}$);
3       else Expiring ($a_{old}$,$S_{N,q}$,$SKY_{N,q}$); Inserting ($a_{new}$,$S_{N,q}$,$SKY_{N,q}$);
4   end while
```

Next, we first show the correctness of conducting the skyline computation on $S_{N,q}$ instead of DS_N. Then, we show that $S_{N,q}$ is the minimum information we should continuously maintain to ensure that the three properties above hold, and the expected size of $S_{N,q}$ is poly-logarithmic.

3.1. Correctness of using $S_{N,q}$ only

Given an element a, 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. Since $S_{N,q} \subseteq DS_N$, $P_{new}|_{S_{N,q}}(a) \geq P_{new}(a)$, $P_{old}|_{S_{N,q}}(a) \geq P_{old}(a)$, and $P_{sky}|_{S_{N,q}}(a) \geq P_{sky}(a)$. To continuously conduct the skyline computation on $S_{N,q}$ instead of DS_N, we calculate $P_{new}|_{S_{N,q}}(a)$, $P_{old}|_{S_{N,q}}(a)$ and $P_{sky}|_{S_{N,q}}(a)$ instead of $P_{new}(a)$, $P_{old}(a)$ and $P_{sky}(a)$; below we show that this will not give wrong results. Specifically, we show the following.

Property 1 (*No missing skyline points*). *The following Lemma is immediate based on* (4) *and* (5).

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

Property 2 (*No false hits to determine $S_{N,q}$*). *We show that incrementally maintaining $S_{N,q}$ based on itself is equivalent to determining $S_{N,q}$ based onDS_N. Lemma 2 is the key.*

Lemma 2. *For two elements a and a' in DS_N, if a' is newer than a, $P_{new}(a) \geq q$, and $a' \prec a$, then $P_{new}(a') \geq q$.*

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 lemma holds. □

Lemma 2 immediately implies that for each element a in $S_{N,q}$ (i.e. $P_{new}(a) \geq q$), calculating $P_{new}(a)$ against the elements in $S_{N,q}$ is the same as calculating $P_{new}(a)$ against the whole window DS_N; this is formally stated by Theorem 2.

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

Example 2. Regarding the example in Fig. 1 (a), 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 the $P_{new}|_{S_{N,q}}$ values of a_2, a_3, a_4, and a_5 are also 1, 1, 0.9, and 1, respectively. □

Lemma 2 and Theorem 2 imply that incrementally maintaining $S_{N,q}$ based on the elements in $S_{N,q}$ is equivalent to maintaining $S_{N,q}$ based on DS_N.

Property 3 (*No false hits to determine $SKY_{N,q}$*). *Continuing Example2, we can verify that $P_{old}|_{S_{N,q}}(a_4) = 0.6 \times 0.7 = 0.42$ since a_1 is not contained in $S_{N,q}$, while Example1 also shows that $P_{old}(a_4)=0.042$. This shows that for an element $a \in S_{N,q}$, $P_{old}|_{S_{N,q}}(a)$ does not always equal $P_{old}(a)$. Since $P_{sky}|_{S_{N,q}}(a) \geq P_{sky}(a)$, we only need to show that for each element $a \in S_{N,q}$, if $P_{sky}|_{S_{N,q}}(a) \geq q$ then $P_{sky}(a) \geq q$; this is formally stated in Theorem3.*

Theorem 3. *For each $a \in S_{N,q}$, if $P_{sky}|_{S_{N,q}}(a) \geq q$ then $P_{sky}(a) \geq q$.*

To show Theorem 3, we only need to show that if $P_{sky}(a) < q$ then $P_{sky}|_{S_{N,q}}(a) < q$. Note that $P_{sky}(a) = P(a) \times P_{new}(a) \times P_{old}(a)$ and $P_{sky}|_{S_{N,q}}(a) = P(a) \times P_{new}|_{S_{N,q}}(a) \times P_{old}|_{S_{N,q}}(a)$. Theorems 4 and 5 below are the key.

Theorem 4. *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 4 immediately follows from Lemma 3 below.

Lemma 3. *For any 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, $P_{new}(a') \geq P_{new}(a) \times P_{old}(a)$. Since $P_{new}(a) \times P_{old}(a) \geq q$, $P_{new}(a') \geq q$. Thus, $a' \in S_{N,q}$. □

Theorems 2 and 4 immediately imply the following corollary.

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

Now we show Theorem 5 below.

Theorem 5. *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 all elements dominating a are in $S_{N,q}$ then $P_{old}|_{S_{N,q}}(a) \times P_{new}|_{S_{N,q}}(a) = P_{old}(a) \times P_{new}(a) < q$. 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 are younger than a'.

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

Corollary 1 and Theorem 5 immediately imply the following corollary.

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

Corollary 2 immediately implies Theorem 3.

Summary. Based on the above discussions, in our techniques we only need to incrementally maintain $S_{N,q}$, calculate all probabilities against $S_{N,q}$, and select elements a with $P_{sky}|_{S_{N,q}}(a) \geq q$.

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

Minimality. The following theorem shows that removing an element from $S_{N,q}$ will make the Property 2 not hold with the arrival of a new element.

Theorem 6. *Suppose that an element $a \in S_{N,q}$ is removed. If a is not the oldest element in the sliding window (i.e., a must expire once a new element arrives), then the Property 2 may not hold for a new element b regarding any subset of the current $DS_N - \{a\}$; that is, false positive may happen if a is removed.*

Proof. Suppose that before a new element arrives, $p = P_{new}(a)$; note that $p \geq q$. We assume that the new arrived element b has the occurrence probability q/p. It is straightforward to get such an element b that b is dominated by a but is not dominated by any data elements that do not dominate a. Let c be the latest element c arriving earlier than a and dominating a. Assume that new elements keep arriving till c expires, and all those new elements arriving after b do not dominate b. It can be immediately verified that the skyline probability of b restricted to DS_N by excluding a and expiring c is q; consequently, the skyline probability of b restricted to any subset of DS_N is not smaller than q. Nevertheless, once c expires $P_{sky}(b) = q \times (1 - p(a)) < q$. Thus, the Property 2 does not hold for any subset of the current $DS_N - \{a\}$. □

Note that in principle, $S_{N,q}$ might include the oldest data element that is not a q-skyline point. Such an oldest data element could be removed since it will expire once the next element arrives. Thus, $S_{N,q}$ may be one element more than the minimal in the worst case. In our algorithm, we do not remove such an element from $S_{N,q}$ for the following reasons: (1) $S_{N,q}$ may be only one element away from being minimal, (2) such an element may already have been excluded from $S_{N,q}$, (3) in the time-stamp based sliding window model there is no such data element which can be specified due to the continuity of time.

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 when the value distribution of each data element on any dimension, including arriving order, is the same and independent.

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.

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

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 $P(\neg W) \triangleq \prod_{j \in W}(1 - P_j)$ 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 7 immediately follows from (6).

Theorem 7. *Let DS_N be a sequence of N data elements with probabilities $P_1, P_2, \ldots, P_N$. Then,*

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

Below we show that (7) is bounded by a logarithmic size. Given a P_i, let $q_{k,i} \triangleq \max\{P_i \times P(\neg W) | |W| = k\}$. 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 (7). Clearly, $q_{k,i}$ is monotonically decreasing regarding k; that is, $q_{k',i} \geq q_{k,i}$ 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.

$$E(SKY_{N,q}) \leq \sum_{i=1}^{N} \left(\sum_{j=0}^{k_i - 1} P(DOMT_i^j) \times (q_{j,i} - q_{(j+1),i}) + P(DOMT_i^{k_i}) q_{k_i,i} \right). \tag{8}$$

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

Theorem 8. *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 on each dimension are distinct. Then, $P(DOMT_i^k) \le (k+1)/N \times (1+H_{d-1,N}-H_{d-1,k+1})$ when $d \ge 2$ and $P(DOMT_i^k) = (k+1)/N$ when $d=1$.*

Proof. Without loss 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 $1/N$ probability to take jth position $(1 \le j \le N)$ 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 $d \ge 2$. 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 $(k+1)/j$ since for each permutation with a_i at jth position on the first dimension, the value of a_i on the second dimension must take one of the $(k+1)$ smallest values among the j elements with the j smallest values on the first dimension. Thus, we have

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

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, $P(DOM_i^k)|_{j,l} \le (k+1)/j \times \ (1+H_{l-1,j}-H_{l-1,k+1})$; consequently, the $P(DOM_i^k)$ regarding the $(l+1)$-dimensional space and N data elements is

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

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

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

It can be immediately verified that $H_{d,N} = O(\ln^d N)$; consequently $P(DOMT_i^k) = O(k/N \ln^{d-1} N)$. This together with Theorem 8 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 $(d-1)$ since each $0 \le q_{j,i} \le 1$.

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. The following theorem is immediate. Let $p_{k,i} \triangleq \max\{P(\neg W) \mid |W| = k\}$ and $P(skyt_i^j)$ denote the probability that there are at most j elements in DS_N^c (remove element probabilities from DS_N) that dominate the ith element and arrive after i.

Theorem 9. *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.*

$$E(S_{N,q}) \le \sum_{i=1}^{N}\left(\sum_{j=0}^{k_i-1} P(skyt_i^j) \times (p_{j,i}-p_{(j+1),i}) + P(skyt_i^{k_i})p_{k_i,i}\right) \tag{10}$$

Note that $P(skyt_i^{k_i})$ can be estimated in the same way as that in Theorem 8 by replacing d by $d+1$. Therefore, the expected size of $S_{N,q}$ is poly-logarithmic regarding N with the order of d. While the upper-bounds of the expected sizes of $S_{N,q}$ and $SKY_{N,q}$ are shown in poly-logarithmic regarding N, our experiments also demonstrate that the sizes of $S_{N,q}$ and $SKY_{N,q}$ are significantly smaller than that of DS_N in a low dimensional space even when distributions of elements are strongly correlated.

Remark: Note that in our incremental techniques, we will use $1/(1-P(a))$. Nevertheless, a data element a could have occurrence probability 1. This will make $(1-P(a)) = 0$. To resolve this, we will use two probability values, 1 and $(1-q')$, to represent the probability value 1, where q' is any number smaller than q. Then, we use 1 to represent $P(a)$ and $(1-q')$ to represent $(1-P(a))$. Let DS_N be a given data stream and DS'_N is the data stream after we modify the probability value 1 as above. It can be immediately verified that the $S_{N,q}$ for both DS_N and DS'_N is the same regarding a given q, as well as the q-skyline; this is because $(1-P(a)) = 0$ is equivalent to $(1-P(a)) < q$ against a given q. Thus, we can use this modification of probability 1 in our algorithm.

4. 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 $P_{sky}|_{S_{N,q}}(a) \ge q$. Note a new data element may cause several elements to be deleted from $S_{N,q}$, nevertheless, the time complexity is $O(|S_{N,q}|)$ per element which is expected to be poly-logarithmic regarding N with the order of d (Section 3.2). 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 $S_{N,q}$.

Our algorithms calculate $P_{sky}|_{S_{N,q}}$, $P_{old}|_{S_{N,q}}$, and $P_{new}|_{S_{N,q}}$. For notation simplification, in the remaining of this section, $P_{sky}|_{S_{N,q}}$, $P_{old}|_{S_{N,q}}$, and $P_{new}|_{S_{N,q}}$ are abbreviated to P_{sky}, P_{old}, P_{new}, respectively.

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, the correctness and complexity of our techniques are shown.

4.1. 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 storing a data element twice.

In-memory *R*-trees R_1 and R_2 on the points of $SKY_{N,q}$ and the points of $(S_{N,q}-SKY_{N,q})$, respectively, will be used and continuously maintained; see Fig. 2 for an example.

To avoid drilling down the *R*-trees every time, at each entry *E* of an *R*-tree the following aggregate information will be stored to facilitate deferring the update propagation of aggregate information to leaves, and removing (or inserting) entire entries.

- $P^{global}_{new}(E)$ stores the captured multiplication of non-occurrence probabilities of the elements which dominate all elements in *E* and are newer than all elements in *E*. Note that $P^{global}_{new}(E)$ needs to be integrated later.
- We use $P^{global}_{old}(E)$ to store the multiplication of non-occurrence probabilities of the captured elements which dominate any element in *E* and are pruned from $S_{N,q}$ (or expire). Note that we use P^{global}_{old} since those captured removed elements must arrive earlier than any element in the remaining $S_{N,q}$ according to Lemma 2, and the expired element certainly arrives earlier than any element in *E*. We need to update P_{old} and P_{sky} by $P^{global}_{old}(E)$ since *E* is no longer in the current $S_{N,q}$ and we continuously monitor $S_{N,q}$.
- We use $P_{noc}(E)$ to store $\prod_{e \in E}(1-P(e))$. This will be used to facilitate the computation of P^{global}_{old} when *E* is pruned from $S_{N,q}$.
- $P_{sky,min}(E)$ and $P_{sky,max}(E)$ store the minimum skyline probability and maximum skyline probability of the elements rooted at *E* (excluding the P^{global}_{old} and P^{global}_{new} recently captured at some entries).
- $P_{new,min}(E)$ and $P_{new,max}(E)$ store the minimum and maximum P_{new} values of the elements rooted at *E* (excluding the recently captured P^{global}_{new} at some entries).

Below, we use one example to show the effectiveness of using aggregate information in continuously computing $SKY_{N,q}$ from $S_{N,q}$.

Illustrative example. Regarding the example in Fig. 2, assume that $N=13$, $q=0.2$, the occurrence probabilities are as depicted, and $DS_N = \{a_i | 1 \le i \le 13\}$. 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 $2 \le i \le 13$, 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 $S_{N,q}$ and $SKY_{N,q}$. In fact, a_{14} dominates entries E_4, E_2, and $R_2.root$ (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 value of P_{new} ($P_{new,max}$) multiplied by $(1-P(a_{14}))$ is 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} (i.e., $P_{new,min}$) 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} (i.e., $P_{sky,min}$ and $P_{sky,max}$) 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 are in $S_{N,q}$, we can update their P_{new} values globally by keeping a global value $P^{global}_{new} = P^{global}_{new} \times (1-P(a_{14}))$ at E_2 to avoid propagating the updates of all elements contained by E_2.

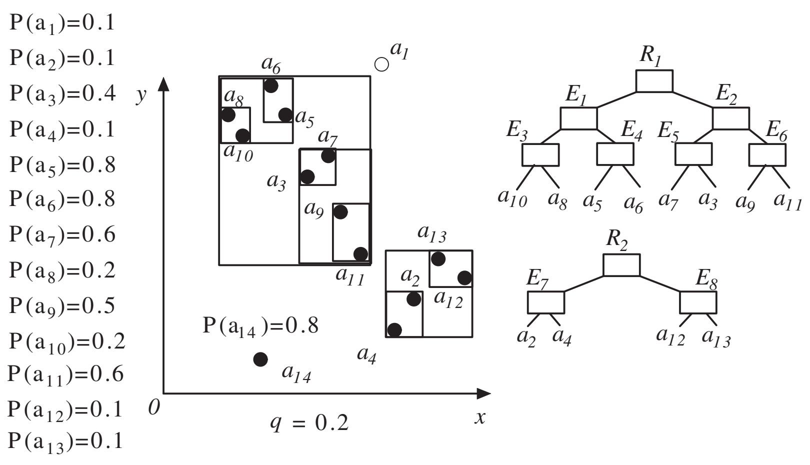

Fig. 2. Aggregate *R*-trees.

Furthermore, in this example some elements may be pruned from $S_{N,q}$ once a_{14} arrives. For example, when a_{14} arrives, $P_{new}(a_2) = 0.18 < 0.2$; thus a_2 needs to be pruned. Note that since $P_{new}(a_{12}) = P_{new}(a_{13}) = 0.2$ after a_{14} arrives, we still need to keep a_{12} and a_{13} in $S_{N,q}$; that is, we still need to keep E_8. To avoid propagating the update of each element (i.e. a_{12} and a_{13}) contained by E_8 individually due to the removal of a_2, we can keep a global value $P^{global}_{old} = P^{global}_{old} \times (1-P(a_2))$ at E_8 so that we know that the P_{old} values for elements in E_8 will be updated by multiplying $1/P^{global}_{old}$ as we focus on $S_{N,q}$. 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 non-occurrence probability of the elements in E $-P_{noc}(E) = \Pi_{a \in E}(1-P(a))$, then we can update P^{global}_{old} at E' by multiplying $P_{noc}(E)$.

Continue the example in Fig. 2 against the first 13 elements. P^{global}_{old} and P^{global}_{new} at each internal entry are initialized to 1. When a_{10} arrives, we update $P^{global}_{new}(E_4)$ from 1 to $(1-P(a_{10})) = 0.8$ since a_{10} dominates the MBB of E_4, while other P^{global}_{new} 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 the leaf-level and is only captured at the entry E_4; consequently the captured skyline probabilities for a_6 and a_5 are still $P(a_6) \times (1-P(a_8))$ (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^{global}_{new} give the current 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 current values.

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

Invariants. As mentioned earlier, the basic idea of our algorithm is to defer the propagation of updates to the leaves (data elements) and to move entries at levels as high as possible. To ensure the correctness, our algorithm maintains the four invariants in the incremental computation. Intuitively, the $P_{sky,min}$ and $P_{sky,max}$ at E multiplied by the P^{global}_{new} and $1/P^{global}_{old}$ at entries from the root to E (inclusive) give the correct values of $P_{sky,min}$ and $P_{sky,max}$ at E regarding $S_{N,q}$. Similar invariants hold for $P_{new,min}$ and $P_{new,max}$ (but excluding P^{global}_{old} values). Let L_E denote the set of entries on the path from the root of the R_i to E (inclusive) where R_i is R_1 or R_2; we describe the four invariants precisely below. For each entry E,

Invariant 1 : $P_{sky,min}(E) \times \Pi_{E' \in L_E} P^{global}_{new}(E')/P^{global}_{old}(E')$ gives the correct value of $P_{sky,min}$ value at E.

Invariant 2 : $P_{sky,max}(E) \times \Pi_{E' \in L_E} P^{global}_{new}(E')/P^{global}_{old}(E')$ gives the correct value of $P_{sky,max}$ value at E.

Invariant 3 : $P_{new,min}(E) \times \Pi_{E' \in L_E} P^{global}_{new}(E')$ gives the correct value of $P_{new,min}$ value at E.

Invariant 4 : $P_{new,max}(E) \times \Pi_{E' \in L_E} P^{global}_{new}(E')$ gives the correct value of $P_{new,max}$ value at E.

Top-down and bottom-up: recalculate the aggregates. To retain the four invariants and save the computation costs, in our algorithm we push down (top-down) P^{global}_{new} and P^{global}_{old} iteratively to the children entries of an encountered entry, when we search down along the R-tree, while we also iteratively use the bottom-up paradigm to update the values $P_{new,min}$, $P_{new,max}$, $P_{sky,min}$, and $P_{sky,new}$ at an encountered entry by its children entries. Below we detail the top-down and bottom-up methods at an encountered entry E.

Top-down (E). Here, $E \subseteq R_i$ and R_i is R_1 or R_2. We push $P^{global}_{old}(E)$ and $P^{global}_{new}(E)$ down to the children entries of E. That is, for each child E_1 of an entry E, we update $P^{global}_{old}(E_1)$ to $P^{global}_{old}(E_1) \times P^{global}_{old}(E)$ and then reset $P^{global}_{old}(E)$ to 1 if $P^{global}_{old}(E) < 1$; similarly, $P^{global}_{new}(E_1)$ is updated to $P^{global}_{new}(E_1) \times P^{global}_{new}(E)$ and $P^{global}_{new}(E)$ is reset to 1 if $P^{global}_{new}(E) < 1$.

Bottom-up (E). Here, $E \subseteq R_i$ and R_i is R_1 or R_2. We recalculate $P_{sky,min}(E)$, $P_{sky,max}(E)$, $P_{new,min}(E)$, and $P_{new,max}(E)$ from its children entries as follows.

Let $\mathcal{E}$ denote the set of all children of E. $P_{sky,min}(E) = \min_{E' \in \mathcal{E}}\{P_{sky,min}(E') \times 1/P^{global}_{old}(E') \times P^{global}_{new}(E')\}$, and $P_{sky,max}(E) = \max_{E' \in \mathcal{E}}\{P_{sky,max}(E') \times 1/P^{global}_{old}(E') \times P^{global}_{new}(E')\}$. Similarly, $P_{new,min}(E) = \min_{E' \in \mathcal{E}}\{P_{new,min}(E') \times P^{global}_{new}(E')\}$, and $P_{new,max}(E) = \max_{E' \in \mathcal{E}}\{P_{new,max}(E') \times P^{global}_{new}(E')\}$

Bottom-up to update P_{noc}. Once an entry E in R_1 or R_2 is removed, inserted, or modified, we also update P_{noc} in a bottom-up fashion to reflect the change. For example, if an entry E is deleted, then for each entry E' on the path from its parent entry to the root, $P_{noc}(E') := P_{noc}(E')/P_{noc}(E)$. Other cases will be dealt similarly.

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

We detail Inserting (a_{new},$S_{N,q}$,$SKY_{N,q}$) and Expiring (a_{old},$S_{N,q}$,$SKY_{N,q}$) in Algorithm 1 in the next two subsections.

4.2. *Inserting (a_{new},$S_{N,q}$,$SKY_{N,q}$)*

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 (prune) the elements a with updated $P_{new}(a) < q$ from R_1 and R_2 and update P_{old} values of the remaining elements in $S_{N,q}$ dominated by some of the pruned elements, (3) update P_{sky} (via P_{old} and P_{new}) values for the elements dominated by some of those pruned elements or a_{new}, (4) move elements a in R_1 with $P_{sky}(a) < q$ to R_2, and (5) calculate $P_{sky}(a_{new})$ and insert a_{new} to R_1 or R_2 accordingly since $P_{new}(a_{new}) = 1$.

The remaining elements in $S_{N,q}$ may be dominated by some pruned elements. Nevertheless, Lemma 2 implies that in the task (2) above, we only need to update P_{old} values for those remaining elements since the remaining elements cannot be dominated by pruned elements which arrive later. Moreover, by dominance transitivity 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}.

The central idea to deal with a newly arrived element a_{new} is to iteratively traverse R_1 and R_2 to identify the entries which are either fully dominated by a_{new} or fully dominate a_{new}, where $C_{a_{new}<}$ stores the entries fully dominated by a_{new}, and $C_{<a_{new}}$ stores the entries fully dominate a_{new}. Then, entries in $C_{a_{new}<}$ are used for tasks (1)–(4), and the entries in $C_{<a_{new}}$ are used for task (5). Algorithm 2 below is an outline of our techniques.

Algorithm 2. Inserting (a_{new},$S_{N,q}$,$SKY_{N,q}$).

```
Input: N: window size; q: skyline probability threshold; $a_{new}$: newly arrived data element.
$R_1$ and $R_2$: two aggregate R-trees on $SKY_{N,q}$ and ($S_{N,q}-SKY_{N,q}$), respectively.
Output: Updated $R_1$ and $R_2$
  Step 1: Iteratively, traverse from $R_1.root$ and $R_2.root$ to identify the entries to be loaded into $C_{a_{new}<}$ or $C_{<a_{new}}$.
  Step 2: Use $C_{a_{new}<}$ to conduct the tasks (1)–(4) above, and use $C_{<a_{new}}$ to conduct the task (5) above.
```

Next we detail Step 1 and Step 2.

Step 1. Recall the definitions of full dominance, partial dominance, and non-dominance relationships between two entries in Section 2.2. We use C_1 to keep the entries in R_1 and R_2 which is partially dominated by a_{new} but do not dominate a_{new}, C_2 to keep the entries which partially dominate a_{new} but are not dominated by a_{new}, and $C_{1,2}$ to keep the entries which partially dominate a_{new} and are partially dominated by a_{new}. Note that the descendent entries of C_1 may be fully dominated by a_{new} but cannot fully dominate a_{new} according to Theorem 1. Similarly, the descendent entries of C_2 may fully dominate a_{new} but cannot be fully dominated by a_{new}. That is, the descendent entries of the entries in C_1 could only be loaded into $C_{a_{new}<}$, and the descendent entries of the entries in C_2 could only be loaded into $C_{<a_{new}}$. It is immediate that the descendent entries of the entries in $C_{1,2}$ could be loaded into either $C_{a_{new}<}$ or $C_{<a_{new}}$.

We initially feed $C_{1,2}$ the root entries of R_1 and R_2. Then, we iteratively dequeue an entry E from $C_{1,2}$ to do the following (till $C_{1,2}=\emptyset$). For each child entry E_1 of E, E_1 is allocated to $C_{a_{new}<}$, or $C_{<a_{new}}$, or C_1, or C_2, or $C_{1,2}$; otherwise E_1 is just discarded. This can be immediately conducted based on their definitions.

We also iteratively dequeue an entry E from C_1. For each child entry E_1 of E, E_1 is allocated to $C_{a_{new}<}$, or C_1; otherwise E_1 is just discarded. Similarly, we iteratively identify the descendent entries of the entries in C_2 to be loaded to $C_{<a_{new}}$.

Note that while iteratively conducting the above search, we push down P_{old}^{global} and P_{new}^{global} at the entries E encountered by top-down (E) in Section 4.1.

Step 2. Note that in $C_{a_{new}<} \cup C_{<a_{new}}$, there do not exist two entries such that one is decedent of the other. For each entry E_1 in C_1, we conduct the following.

Step 2.1 : Update $P_{new}^{global}(E_1)$ to $P_{new}^{global}(E_1) \times (1-P(a_{new}))$ and push down $P_{new}^{global}(E_1)$ to the children entries of E_1 by top-down (E_1) in Section 4.1.

Step 2.2 : Recalculate $P_{new,min}(E_1)$, $P_{new,max}(E_1)$, $P_{sky,min}(E_1)$, and $P_{sky,max}(E_1)$ by bottom-up (E_1) in Section 4.1.

Step 2.3 : Iteratively search down the sub-R tree rooted at E_1 to identify the decedent entries of E_1 which should be removed from $S_{N,q}$—store them in S_r, or should be moved from R_1 to R_2—store them in S_1^{switch}, or should stay in the original R-trees—store them in S_1^{stay} and S_2^{stay} accordingly.

Note that when we iteratively search down level-by-level from E_1, we also push down P_{old}^{global} and P_{new}^{global} at each entry E encountered by top-down (E) in Section 4.1, and then recalculate $P_{new,min}(E)$, $P_{new,max}(E)$, $P_{sky,min}(E)$, and $P_{sky,max}(E)$ by bottom-up (E).

In Step 2.3, we put an entry E into: (1) S_r if the updated $P_{new,max}(E)<q$; (2) S_1^{switch} if E was in R_1, the updated $P_{sky,max}(E)<q$, and $P_{new,min}(E)\geq q$; (3) S_1^{stay} if E was in R_1 and $P_{sky,min}(E)\geq q$, and (4) S_2^{stay} if E was in R_2, $P_{new,min}(E)\geq q$, and $P_{sky,max}(E)<q$. If none of the above four cases, we continue to drill down E.

Update P_{old}^{global}. Once S_r, S_1^{switch}, S_1^{stay}, and S_2^{stay} are determined, we need to update P_{old}^{globdl} for the (descendent) entries in $S^{mix}=S_1^{switch}\cup S_1^{stay}\cup S_2^{stay}$ dominated by elements in S_r. We use the *synchronous traversal* paradigm [12] to traverse S_r and S^{mix} level-by-level by following the R-tree structures of the entries in S_r and S^{mix}. Here, we create a dummy root entry for S_r with all entries in S_r to be children of the root; similar treatments are done for S^{mix}. We put the root entry of S_r to S_1 and the root entry of S^{mix} to S_2. Iteratively, for each pair $E1 \in S_1$ and $E2 \in S_2$,

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 S_1 and the children of $E2$ to S_2 for the next iteration.

Note that when we iteratively traverse down S^{mix}, we also push down P_{old}^{global} and P_{new}^{global} at each encountered entry in S^{mix}.

Update entries and probability mass. For each entry E in S_1^{switch} (i.e., in R_1), we find an appropriate level in R_2 to insert E; we propose to use the level with the length to the leaf to be the same as that of E in R_1. Note that we also iteratively push down P_{old}^{global} and P_{new}^{global} to the children of each encountered entry while searching down in R_2 for the insertion.

To ensure the four invariants, we need to update the values of $P_{sky,min}$, $P_{sky,max}$, $P_{new,min}$, and $P_{new,max}$ in the bottom-up fashion. Details are given below.

For each entry E in S_r, we iteratively recalculate those values from the parent E' of E along the path between E' and the root of the corresponding R-tree (i.e., R_1 or R_2) in a bottom-up fashion. That is, we first recalculate those values in E' by bottom-up (E'), then the parent of E', so on and so fourth. Here, E will be excluded when recalculating E'. Each entry E in S_1^{stay} or S_2^{stay} will be processed in exactly the same way except that we should include E to recalculate E'. Each entry E in S_1^{switch} inserted into R_2 will be treated as a deletion from R_1 and an insertion to R_2; consequently, do updates from E to the root of R_1 in exactly the same as those in S_r and do updates to R_2 in exactly the same way as those in S_2^{stay}.

In the above bottom-up updates regarding an $E \in S_r$, we also update P_{noc} at each entry from E' (the parent of E) to the root (inclusive) by multiplying $1/P_{noc}$. Regarding an entry $E \in S_1^{switch}$, in the bottom-up updates in R_1, P_{noc} in R_1 will be updated in the same way as those regarding S_r, while P_{noc} in R_2 will also be updated in the same way except multiplying by $P_{noc}(E)$ instead of $1/P_{noc}(E)$.

Remark: To avoid processing an entry multiple times, in our implementation we integrate together the bottom-up updates for entries in S_r, S_1^{switch}, S_1^{stay}, and S_2^{stay} from the lowest level.

Inserting a_{new}. It involves the following two steps: (1) assign $P_{old}(a_{new})$ and $P_{sky}(a_{new})$ by $\Pi_{E \in C_{<a_{new}}} P_{noc}(E)$ and $P(a_{new}) \times \Pi_{E \in C_{<a_{new}}} P_{noc}(E)$, respectively, and assign $P_{new}(a_{new})$ by (1) and (2) insert a_{new} into R_1 or R_2 according to the value of $P_{sky}(a_{new})$.

All updates in Step (2) will be processed in the same way as an entry in S_1^{switch} to be inserted in R_2.

4.3. *Expiring (a_{old},$S_{N,q}$,$SKY_{N,q}$)*

Once an element a_{old} expires, we first check if it is in $S_{N,q}$. If a_{old} is already pruned from $S_{N,q}$, then we do nothing.

Otherwise, if it is in $S_{N,q}$ then we need to do the following: (1) increase the P_{old} values for elements dominated by a_{old} by multiplying $1/(1-P(a_{old}))$; and (2) we need to determine the elements that need to be moved from R_2 to R_1. These will be dealt as follows.

Particularly, we adopt the techniques in the part of "Update P_{old}^{global}" in last subsection to update P_{old}^{global} values in R_1 and R_2; that is, treat $S_r = \{a_{old}\}$ and treat $R_1 \cup R_2$ as S^{mix}.

Then we identify the entries in R_2 which will be moved to R_1 after updating P_{old}^{global}. Therefore, we still have three categories, (1) the set S_1 of entries in R_1 with the P_{old}^{global} values changed (note that entries in R_1 are impossible to be moved to R_2 since the skyline probability will only be increased), (2) the set S_2 of entries in R_2 with the P_{old}^{global} values changed but still staying in R_2, and (3) the set S_3 of entries in R_2 which will be moved to R_1 due to the change of P_{old}^{global}. We iteratively drill down each entry in R_2 with the updated P_{old}^{global} to identify S_2 and S_3. Then we update R_1 by S_1 in the same way as updating R_1 by S_1^{stay}, and update R_2 by S_2 in the same way as updating R_2 by S_2^{stay} in the last subsection. We move S_3 from R_2 to R_1 and do the updates similarly to that we move S_1^{switch} from R_2 to R_1 and do the corresponding updates in the last subsection.

Finally, we need to remove a_{old} from R_1 or R_2. This will be treated similarly to that we remove S_r from R_1 in the last subsection.

4.4. *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 $S_{N,q}$. Since the four invariants are retained, Theorems, Lemmas and Corollaries in Section 3.1 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 are much faster than the trivial algorithm against $S_{N,q}$ as what are discussed in the beginning of this section.

5. Variants

The techniques developed in the paper can be immediately extended to cover the following variants.

5.1. *Multiple confidences*

Continuous queries. Different users may specify different confidences; that is, different thresholds q. A naive way to process this is to run k queries, with k different thresholds, separately. This not only involves the continuous computation k times but also needs to store k different $S_{N,q}$ and many elements may be stored k times. In the applications where all the queries are centrally handled by one computer, sharing computation and minimizing the memory usage is critical. Below, we extend our techniques for a single given confidence to achieve this goal.

Suppose that users specify k confidences $q_1, q_2, \ldots, q_k$ where $q_i < q_{i+1}$. Instead of maintaining a single solution set R_1, we maintain k solution sets $R^1, R^2, \ldots, R^k$ such that elements in R^i (for $1 \le i \le k-1$) have the skyline probabilities in $[q_i, q_{i+1})$ and the elements in R^k have the skyline probabilities in $[q_k, 1]$, while R^0 keeps the elements in $(S_{N,q_1} - \bigcup_{i=1}^{k} R^i)$. Those R^i for $i=0$ to k are also maintained as aggregate R-trees with the same aggregate information as that in Section 4.1.

All the techniques from Algorithms in Section 4 are immediately applicable except that now, we need to detect the switch from R^j to R^i for any $j > i$ when inserting a new element, and for any $i > j$ when expiring an element.

Note that once a new query q' issues, if $q' > q_1$, we can immediately process it by splitting a R^j. If $q' < q_1$, then we need to process q' from the scratch.

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 $q' \ge q_1$. Then, we first find an R^i such that $q_i \le q' < q_{i+1}$. Clearly, the elements $\{R^j$: $j \ge i+1\}$ are all contained in the solution. We can apply the iteratively drill down search paradigm in Section 4 to get all elements in R^i with skyline probabilities $\ge q$ but without updating aggregate probabilities information.

As a by-product to continuously computing skyline, we can answer any ad hoc query in the way above. Nevertheless, when $q' < q_1$, the current maintained R^j for $j \le (k+1)$ will miss the solution; therefore, we need to retrieve the skyline against the whole sliding windows by the non-index based existing techniques.

5.2. *Probabilistic top-k skyline elements*

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

The algorithms in Section 4 can be immediately extended to continuously computing the top-k skyline points as follows. $S_{N,q}$ is maintained by removing points with $P_{new} < q$ and is maintained by R_1 and R_2 where R_1 is the solution set. The aggregate information at each entry is also retained, including the probabilities P_{sky}, P_{old}, P_{new}, etc., as described in Section 4.

Note that when a new element a_{new} arrives, we need to determine S_r, S_1^{switch}, S_1^{stay}, and S_2^{stay}. While determining S_r is exactly the same as that in Section 4.2, we treat R_1 and R_2 as two "heap trees" to efficiently determine S_1^{switch}, S_1^{stay}, and S_2^{stay}. Here, we treat R_1 as a min-heap against P_{sky}, and R_2 as a max-heap against P_{sky}. Since $P_{sky,max}(E)$, $P_{sky,min}(E)$, $P_{old}^{global}(E)$, and $P_{new}^{global}(E)$ are maintained at each entry E, $P_{sky,min}$ may be used as the min-heap search key together with P_{new}^{global} and P_{old}^{global} at each entry and $P_{sky,max}$ may be used as the max-heap search key at each entry together with P_{new}^{global} and P_{old}^{global}. Subsequently, 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, which are not in top-k, to R_2.

It will be ideal if we could remove the constraint of "but not smaller than q" and keep candidates of the top-k points with the maximal skyline probabilities. Clearly, the smallest skyline probability in the top-k points may change from time to time. This makes our incremental techniques for maintaining a candidate set not applicable since our techniques are applicable only when a fixed q is given or q increases.

5.3. *Time stamp based sliding windows*

In many applications, a sliding window may be specified against the time. In addition, each data element also has an expiration time.

Problem definition. In a data stream DS on a d-dimensional numeric space, each data element has (1) a time-stamp $\kappa(a)$—the issuing time of a; and (2) a life-span $l(a)$—a is unavailable after $l(a)+\kappa(a)$.

In this subsection, we study the problem of efficiently computing the skyline elements among the *available* data elements issued in the most recent time period T, with the skyline probabilities not smaller than a given threshold q $(0 < q \le 1)$. Suppose that the current system time is t; we calculate the q-skyline of the data elements in $DS_{T,t}$ where

$$DS_{T,t} = \{a \mid t - \kappa(a) \le \min\{T, l(a)\}\} \tag{11}$$

In such a sliding model, we expire an element if it is not within a pre-given most recent time period T or its life-span is over. Note that the larger the vale of $\kappa(a)$, the younger the element a.

Example 3. As depicted in Fig. 3, the window size T is 7 min and $q = 0.5$. Suppose that a_1 arrives at 9:00 am, a_2 arrives at 9:01 am, a_3 arrives at 9:02 am, a_4 arrives at 9:03 am, and a_5 arrives at 9:04 am, where the life-span $l(a_1) = 8$ min, $l(a_2) = 5$ min, $l(a_3) = 6$ min, $l(a_4) = 3$ min, and $l(a_5) = 5$ min. Moreover, $p(a_1) = 0.9$, $p(a_2) = 0.4$, $p(a_3) = 0.3$, $p(a_4) = 0.9$, and $p(a_5) = 0.1$. The threshold is $q = 0.5$.

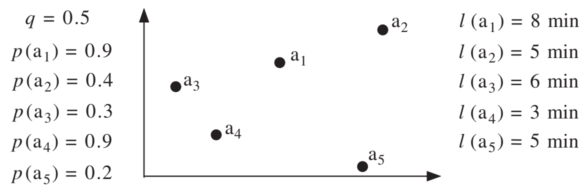

Fig. 3. An example.

Let $t_1 = 9:065$ am and $t_2 = 9:08$ am. It can be immediately verified that $DS_{T,t_1} = \{a_1, a_3, a_5\}$ since the life-span of a_2 and a_4 are exhausted, and $DS_{T,t_2} = \{a_3, a_5\}$ since a_1 expires from the sliding window, and the life-spans of a_2 and a_4 are exhausted.

$S_{N,q}$ *insufficient*. Recall that in the sliding window defined against the most recent N data elements, $S_{N,q} = \{a \mid P_{new}(a) \ge q \& a \in DS_N\}$. We could immediately extend the definition of $S_{N,q}$ to define $S_{T,t,q}$ against a sliding window regarding the most recent time period T where $S_{T,t,q} = \{a \mid P_{new}(a) \ge q \& a \in DS_{T,t}\}$.

It can be immediately verified that the techniques developed in Sections 3 and 4 can be applied to a sliding window T regarding $S_{T,t,q}$ if all elements in the sliding window follow FIFO—"First In, First Out".

Nevertheless, with an arbitrary life-span value of each element a, FIFO does not always hold; consequently, continuously maintaining $S_{T,t,q}$ based on $S_{T,t,q}$ is not sufficient. Below is an example.

Example 4. Continue Example 3. Suppose that t=9:05 am then $S_{T,t,q} = \{a_3, a_4, a_5\}$ since $P_{new}(a_1) < q$ and $P_{new}(a_2) < q$ where q=0.5. Continuously maintaining $S_{T,t,q}$ based on $S_{T,t,q}$ cannot bring a_1 back since a_1 is removed. Nevertheless, when $t_1 =$ 9:065 am, $a_1 \in DS_{T,t_1}$ and $P_{sky}(a_1) = 0.63 > q$; thus, a_1 should be included as a skyline point. This means the skyline point a_1 is lost if we continuously maintain $S_{T,t,q}$ based on $S_{T,t,q}$ only.

To resolve this, we need to replace P_{new} and P_{old} by P_{late} and P_{early}. For each element a, $P_{late}(a)$ is the probability that the elements, expiring not earlier than a in the sliding window T and dominating a, do not occur, while $P_{early}(a)$ is the probability that the elements, expiring earlier than a in the sliding window T and dominating a, do not occur. Given a sliding window defined over the time period T, for each element a its *expiration time* $e(a)$ regarding T is defined as

$$e(a) = \kappa(a) + \min\{T, l(a)\} \tag{12}$$

Precisely, P_{late} and P_{early} are defined as follows at time t.

$$P_{late}(a) = \Pi_{a' \in DS_{T,t}, a' \prec a, e(a') \ge e(a)}(1 - P(a')) \tag{13}$$

$$P_{early}(a) = \Pi_{a' \in DS_{T,t}, a' \prec a, e(a') < e(a)}(1 - P(a')) \tag{14}$$

Immediately, the skyline probability of each element a in $DS_{T,t}$ is $P_{sky}^T(a) = P(a) \times P_{early}(a) \times P_{late}(a)$. Let $C_{T,t,q} = \{a \mid P_{late}(a) \ge q \& a \in DS_{T,t}\}$.

Example 5. Continue Example 3. $e(a_1) = 9:07$ am, $e(a_2) = 9:06$ am, $e(a_3) = 9:08$ am, $e(a_4) = 9:06$ am, $e(a_5) = 9:09$ am.

At $t = 9:05$ am, $DS_{T,t} = \{a_1, a_2, a_3, a_4, a_5\}$ where $P_{late}(a_1) = 0.7$, $P_{early}(a_1) = 0.1$, $P_{late}(a_2) = 0.0056$, $P_{early}(a_2) = 1$, $P_{late}(a_3) = 1$, $P_{early}(a_3) = 1$, $P_{late}(a_4) = 1$, $P_{early}(a_4) = 1$, $P_{late}(a_5) = 1$, $P_{early}(a_5) = 1$. Thus, $C_{T,t,q} = \{a_1, a_3, a_4, a_5\}$.

Framework. Given a threshold q and a sliding window T, the problem of continuously computing the probabilistic q-skyline $SKY_{T,t,q}$ can be conducted by continuously maintaining $C_{T,t,q}$ and retrieving the points continuously from $C_{T,t,q}$ with $P^T_{sky}(a)|_{C_{T,t,q}} \geq q$ where $SKY_{T,t,q} = \{a | a \in DS_{T,t} \& P^T_{sky}(a) \geq q\}$. The framework is similar to Algorithm 1 and is outlined below in Algorithm 3.

Algorithm 3. Continuously computing $SKY_{T,t,q}$.

```
1   while a new element $a_{new}$ arrives do
2       |Inserting ($a_{new}$,$C_{T,t,q}$,$SKY_{T,t,q}$);
3   end while
4   while an old element $a_{old}$ expires do
5       |Expiring ($a_{old}$,$C_{T,t,q}$,$SKY_{T,t,q}$);
6   end while
```

As with the techniques in Sections 3 and 4, we continuously insert new elements into $C_{T,t,q}$ and remove the elements a from $C_{T,t,q}$ which expire or are discovered with $P^T_{late}(a)|_{C_{T,t,q}} < q$.

Using $C_{T,t,q}$ only. We can show that we only need to continuously maintain $C_{T,t,q}$ instead of $DS_{T,t}$. Particularly, we can show that the lemmas and theorems in Section 3 hold if we replace P_{new} by P_{late}, P_{old} by P_{early}, P_{sky} by P^T_{sky}, and $S_{N,q}$ by $C_{T,t,q}$. Below is a summary.

- *No missing skyline points*. Lemma 1 holds if we replace P_{sky} by P^T_{sky} and $S_{N,q}$ by $C_{T,t,q}$.
- *No false hits to determine $S_{T,t,q}$*. Lemma 2 holds if we replace P_{new} by P_{late} and replace "a' is newer than a" by "a' expires no later than a". Theorem 2 holds if P_{new} is replaced by P_{late} and $S_{N,q}$ by $C_{T,t,q}$.
- *No false hits to determine $SKY_{T,t,q}$*. Theorems 3 and 4, Lemma 3, Corollaries 1 and 2 hold if we replace P_{new} by P_{late}, P_{old} by P_{early}, P_{sky} by P^T_{sky}, and $S_{N,q}$ by $C_{T,t,q}$, as well as we replace "a' is newer than a" by "a' expires no later than a" in Lemma 3.

We can also show that Theorem 6 also holds if we replace $S_{N,q}$ by $C_{T,t,q}$, P_{new} by P_{late}, P_{old} by P_{early}, and P_{sky} by P^T_{sky}. Moreover, we can remove the constraint "If a is not the oldest element in the sliding window (i.e., a must expire once a new element issues)" from Theorem 6. Therefore, we can conclude that $C_{T,t,q}$ is the minimum set of data elements to be retained to ensure the above three properties.

Size estimation. Theorem 8 can be immediately applied to estimate $SKY_{T,t,q}$; consequently, the expected size of $SKY_{T,t,q}$ in a d-dimensional space is poly-logarithmic regarding N_T with order $(d-1)$ if the distributions' assumption in Theorem 8 is adapted where N_T is the maximum number of elements issued with a sliding window T. Moreover, using the same arguments in the proof of Theorems 8 and 9, it can be immediately shown that expected size of $C_{T,t,q}$ is poly-logarithmic regarding N_T with order d if the distributions' assumptions in Theorem 8 are used and $\kappa(a)+l(a)$ follows a uniform distribution.

Aggregate R-trees. Similar to Section 4.1, two aggregate R-trees R_1 and R_2 will be maintained where R_1 is built on the elements in $SKY_{T,t,q}$ and R_2 is built on the elements in $(C_{T,t,q}-SKY_{T,t,q})$. In the aggregate R-trees R_1 and R_2, we replace P^{global}_{new}, P^{global}_{old}, $P_{new,min}$, $P_{new,max}$, $P_{sky,min}$, and $P_{sky,max}$ against $S_{N,q}$ by P^{global}_{late}, P^{global}_{early}, $P_{late,min}$, $P_{late,max}$, $P^T_{sky,min}$, and $P^T_{sky,max}$ against $C_{T,t,q}$, respectively.

In addition, we also maintain P^{global}_{RE} at each entry. P^{global}_{RE} records the multiplication of non-occurrence probabilities of the captured elements which dominate all elements in E, expire earlier than all elements in E, and are still kept in R_1 and R_2. This is because a new arriving element may expire earlier than some existing elements and may expire later than the other elements. To continuously maintain P^{global}_{RE}, P^{global}_{early}, and P^{global}_{RE}, at each entry E we maintain $t_{min}(E)$ and $t_{max}(E)$ where t_{min} records the earliest expiration time of the elements in E and t_{max} records the latest expiration time of the elements in E. Below are the details.

- $P^{global}_{late}(E)$ stores the multiplication of non-occurrence probabilities of the captured elements which dominate all elements in E and expire not earlier than all elements in E.
- $P^{global}_{early}(E)$ stores the multiplication of non-occurrence probabilities of the captured elements which dominate all elements in E and expire earlier than all elements in E, and removed because they either expire or are pruned.
- $P^{global}_{RE}(E)$ records the multiplication of non-occurrence probabilities of the captured elements which dominate all elements in E and expire earlier than all elements in E but are still kept in R_1 and R_2.
- $P^T_{sky,min}(E)$ and $P^T_{sky,max}(E)$ store the minimum skyline probability and maximum skyline probability, respectively, of the elements in E (excluding the recently $P^{global}_{late}(E)$, $P^{global}_{early}(E)$, and P^{global}_{RE} at some entries).
- $P_{late,min}(E)$ and $P_{late,max}(E)$ store the minimum and maximum P_{late} values of the elements in E (excluding the recently captured P^{global}_{late} at some entries).

Moreover, we still maintain P_{noc} at each entry.

Algorithms. In our algorithm, we also continuously maintain the four invariants similarly to those in Section 4.1 except that we replace P^{global}_{new}, P^{global}_{old}, $P_{new,min}$, $P_{new,max}$, $P_{sky,min}$, and $P_{sky,max}$ against $S_{N,q}$ by P^{global}_{late}, P^{global}_{early}, $P_{late,min}$, $P_{late,max}$, $P^T_{sky,min}$, and $P^T_{sky,max}$ against $C_{T,t,q}$, respectively, and we also include P^{global}_{RE} in the numerator multiplying P^{global}_{late}.

All the techniques developed in Section 4 can be immediately applied to continuously maintaining R_1 and R_2, and computing $SKY_{T,t,q}$ if we use P^{global}_{late}, P^{global}_{early}, $P_{late,min}$, $P_{late,max}$, $P^T_{sky,min}$, $P^T_{sky,max}$ and $C_{T,t,q}$ instead of P^{global}_{new}, P^{global}_{old}, $P_{new,min}$, $P_{new,max}$, $P_{sky,min}$, and $P_{sky,max}$, and $S_{N,q}$. There is one additional request that we need to update P^{global}_{late}, P^{global}_{early}, and P^{global}_{RE} using t_{min} and t_{max}. For example, for each entry $E \in C_{a_{new} <}$, we do the following:

- If $a_{new} \prec E$ and $t_{max} < e(a_{new})$, then $P^{global}_{late}(E) := P^{global}_{late}(E) \times (1-P(a_{new})$.
- If $a_{new} \prec E$ and $t_{min} > e(a_{new})$, then $P^{global}_{RE}(E) := P^{global}_{RE}(E) \times (1-P(a_{new})$.
- Otherwise, put the children entries into $C_{a_{new} <}$.

P^{global}_{RE} will be iteratively pushed down in the same way as P^{global}_{new} described in Section 4. We update P^{global}_{late} and P^{global}_{early}

in a similar way by using t_{min} and t_{max} combining with the techniques in Section 4. The bottom-up update of $P^T_{sky,min}$ and $P^T_{sky,max}$ will be conducted in the same way as $P^T_{sky,min}$ and $P^T_{sky,max}$, respectively, except that we need to multiply P^{global}_{RE} in the numerators.

After expiring or pruning elements, we also need to update t_{min} and t_{max}; these will be conducted in a bottom-up fashion along with the updates to $P^T_{sky,min}$ and $P^T_{sky,max}$. Moreover, the expiration of elements regarding the time-frame sliding window model is triggered by the expiration time $e(a)$ of an element. As we maintain t_{min} in R_1 and R_2, respectively, the t_{min} in R_1 (and R_2) can serve as such a trigger; that is, once the time reaches t_{min} we expire elements in R_1 (and R_2) with their expiration time t_{min}.

The correctness immediate follows from the four invariants, and the space complexity can be analyzed in the same way as that in Section 4.4. Also, it is hard to provide any sensible analysis of the time complexity. It is worth to note that similar extension can be conducted to solve the queries in Sections 5.1 and 5.2.

6. Performance evaluation

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

SSKY	Techniques in Section 4 to continuously compute q-skyline (i.e., skyline with the probability no less than a given q) against a sliding window.
MSKY	Techniques in Section 5.1 to continuously compute multiple q-skylines concurrently regarding multiple given probability thresholds.
QSKY	Techniques in Section 5.1 to process a skyline query with an ad hoc probability threshold.
TOPK	Techniques in Section 5.2 to retrieve k elements with the largest skyline probabilities.
TIME STAMP	Techniques in Section 5.3 to continuously compute q-skyline against the TIME STAMP based sliding window model.

All algorithms are implemented in C++ and compiled by GNU GCC. Experiments are conducted on PCs with Intel Xeon 2.4 GHz dual CPU and 4 G 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 experiment. We randomly assign a probability value to each transaction; that is, probability values follow the *uniform* distribution. The arrival order of elements follows their transaction time.

Synthetic datasets are generated as follows. We first use the methodologies in [4] to generate 2 million data elements with the dimensionality from 2 to 5 and the spatial locations 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 occurrence 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 to 0.3. Note that, since the domain of the *normal* distribution is $(-\infty,\infty)$, we truncate the values which are smaller than 0 or larger than 1. We assign a random order for elements' arrival in a data stream.

In the TIME STAMP sliding window model, we use the arriving order as the issuing time of an element; that is, the issuing time of an element a is 1 if it arrives first. The life-span of an element follows *uniform* distribution in $[0, N]$ where N is the size of sliding window, which corresponds to the time period T in Section 5.3.

Choosing q. q is the probability threshold in evaluating the efficiency of the queries. To evaluate SSKY, TOPK, and TIME STAMP, we use 0.3 as a default value of q, while to evaluate MSKY with $|Q|$ given probability thresholds $q_1, \ldots, q_{|Q|}$, we let these $|Q|$ 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 the average time to process these 1000 queries.

Table 3 summarizes parameters and their corresponding default values. *In our experiments, all parameters take default values unless otherwise specified.*

In our experiments, we evaluate the time efficiency of our algorithm as well as the space efficiency (i.e., space usage) against dimensionality, size of sliding window, probabilistic threshold, distribution of objects' spatial location and occurrence probability distribution. The processing time of an element e is the delay regarding the arrival of e, including the CPU costs for the insertion of e and the deletion of expired elements invoked by e. Since the processing time of one element is too short to capture precisely, we record the average time for each batch of 1 K elements to estimate the delay per element. In the paper, each intermediate entry of the R-Tree occupies a page and the page size is 1024 bytes in our implementation.

6.1. Evaluating against baseline algorithms

In order to evaluate the effectiveness of the SSKY algorithm, we implement two baseline algorithms, named

Table 3
System parameters.

Notation	Definition (default values)
n	Number of elements in the dataset (2 M)
N	Sliding window size (1 M)
d	Dimensionality of the of the dataset (3)
D	Dataset (anti)
D_P	Occurrence probability distribution of elements (*uniform*)
P_μ	Expected occurrence probability (0.5)
q	Probabilistic threshold (0.3)
q'	Probabilistic threshold q' with $q \le q' \le 1$
k	Number of elements retrieved in TOPK queries (30)

SSKY-NSC and SSKY-NS respectively. SSKY-NSC is a naive algorithm to continuously compute the probabilistic skyline; it keeps all elements in the sliding window by 2 R-trees, R_1 stores the current skyline, and R_2 keeps the remaining elements. We incrementally maintain R_1 and R_2 by using R-trees to prune irrelevant entries and data elements. SSKY-NS algorithm is a naive version of SSKY - where we use R_1 and R_2 to store $SKY_{N,q}$ and $(S_{N,q}-SKY_{N,q})$, respectively, while no statistic information (e.g., P^{global}_{new}, P^{global}_{old}) will be kept. Consequently, any update will have to be immediately pushed down to the leaf levels.

Fig. 4(a) records the maximum numbers of the MBRs accessed for each update of SSKY, SSKY-NS and SSKY-NSC algorithms where the window size varies from 200 k to 1 M. As expected, SSKY-NSC algorithm accesses much more MBRs and the number grows linearly against the window size because all elements in the sliding window are kept. This implies that only a small portion of the elements are kept in the candidate set in SSKY and SSKY-NS. As shown in Fig. 4(a), SSKY can further reduce the number of the MBRs accessed by taking advantage of the statistic information. Similar trends are observed in Fig. 4(b), where the average numbers of the MBRs accessed are reported.

We compare the time efficiency of SSKY, SSKY-NS, SSKY-NSC in Fig. 5(a) and (b) by varying the probability threshold q and the sliding window size N, respectively. As shown, SSKY-NSC is up to 2 orders of magnitude slower than SSKY since it keeps all elements in the sliding window. As expected, SSKY is more efficient than SSKY-NS with the help of statistic information.

In Fig. 6, we study the performance of the algorithms in the worst case scenario, in which every element e dominates all elements which arrive later than e. Thus, every element e in the sliding window is kept since its P_{new} value equals 1. Note that SSKY-NS and SSKY-NSC have the same performance in the worst case since all elements in the sliding window are kept, and hence we only evaluate SSKY and SSKY-NS in this experiment. As expected, the performances of both algorithms degrade due to the large candidate set size. Nevertheless, SSKY significantly outperforms SSKY-NS because SSKY can take advantage of the

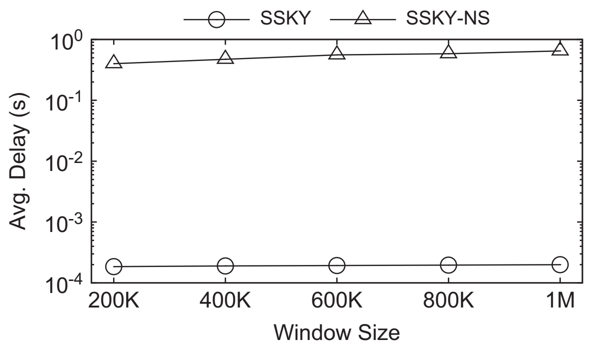

Fig. 6. Worse case study.

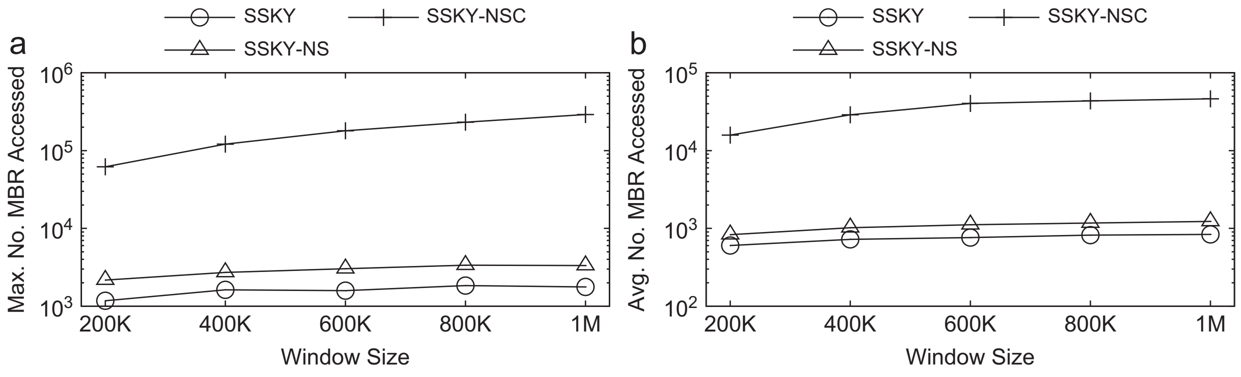

Fig. 4. # MBRs accessed. (a) Max. # MBRs accessed. (b) Avg. # MBRs accessed.

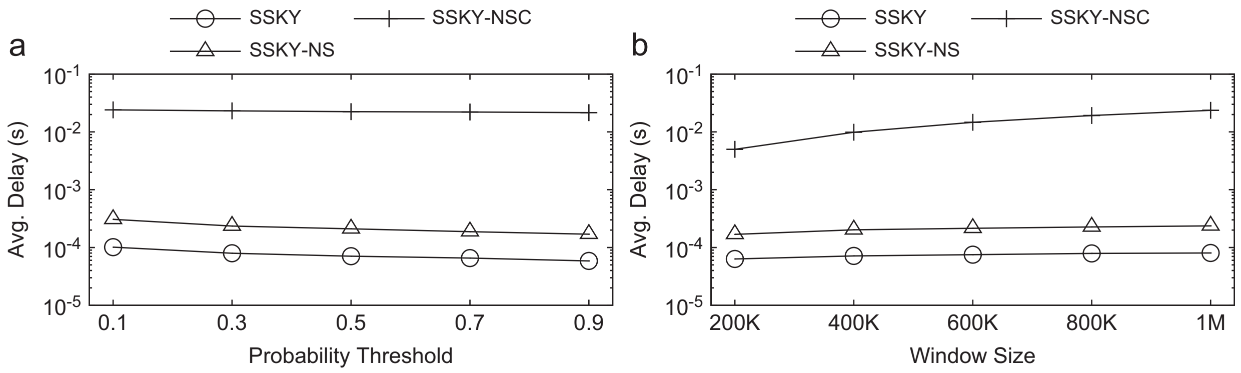

Fig. 5. Comparison with baseline algorithms. (a) Avg. delay vs q. (b) Avg. delay vs N.

statistics information and stop to travel the *aR*-tree on high levels.

We also report the number of pages accessed for each update of SSKY, SSKY-NS and SSKY-NSC algorithms when the window slides in Fig. 7. It shows that the numbers increase in the initialization phase (i.e., the window is not full) and become stable when the window is full. This is because the number of the elements in the window grows with *n* (the number of elements arrived) in the initialization phase and becomes a fixed number (i.e., window size) when the window is full.

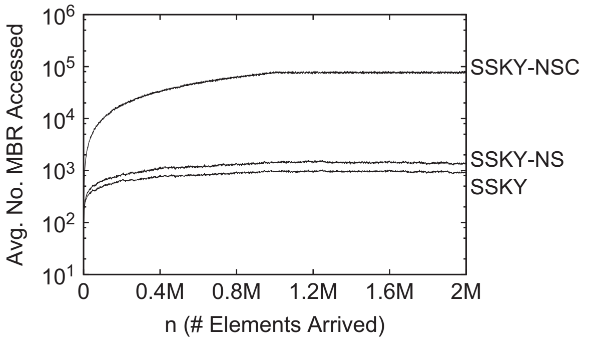

Fig. 7. Time efficiency vs *n*.

6.2. Evaluating SSKY, MSKY and QSKY

In this subsection, we evaluate the performance of SSKY, MSKY and QSKY algorithms in terms of the space usage and the time efficiency.

6.2.1. Space efficiency

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

In Fig. 8, we report the performance of SSKY algorithm against four datasets: Inde-Uniform (Independent distribution for spatial locations of the elements and Uniform distribution for occurrence probability values associated with the elements), Anti-Uniform, Anti-Normal, and Stock-Uniform. We record the maximum sizes of $S_{N,q}$ and $SKY_{N,q}$.

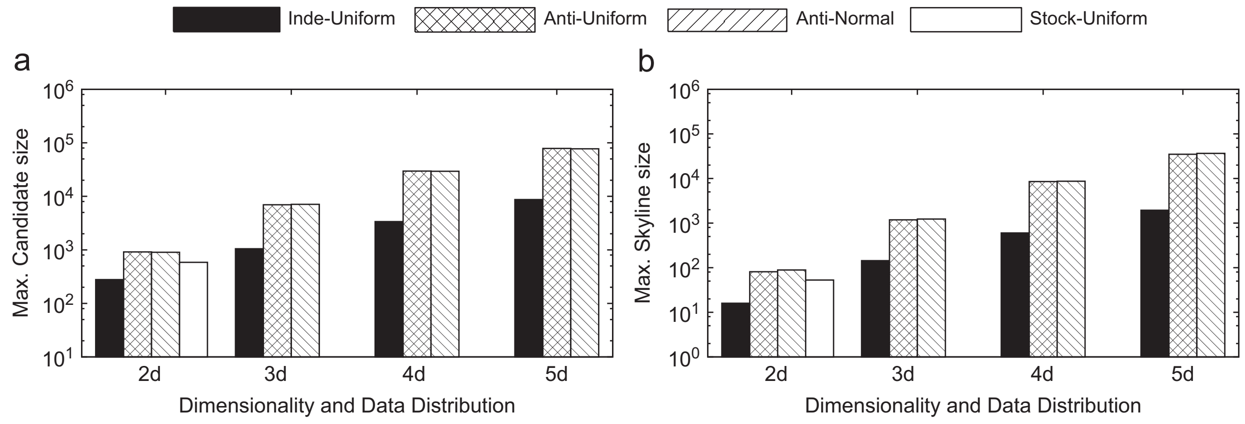

Fig. 8. Space usage vs diff. dataset. (a) Max. candidate size. (b) Max. skyline size.

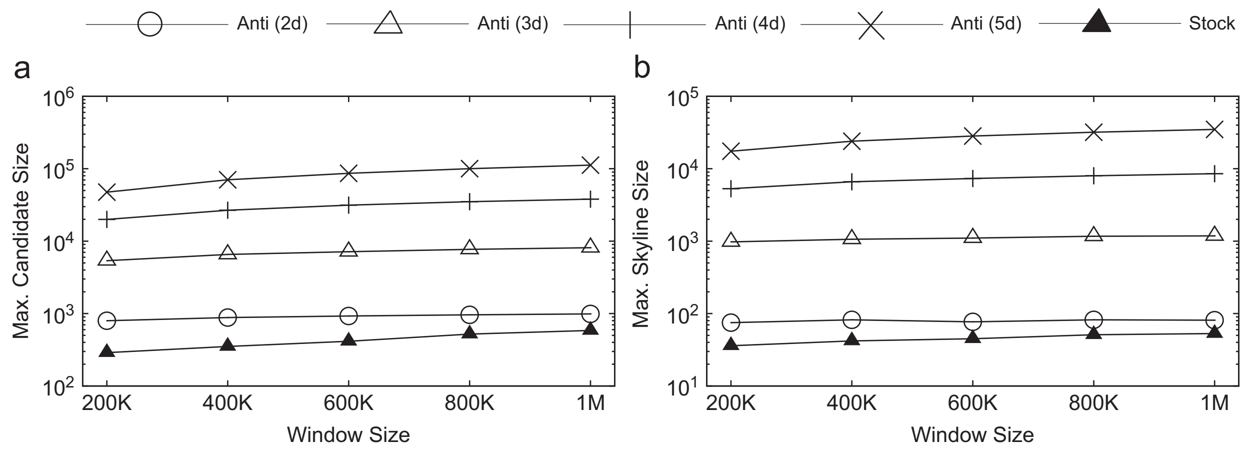

Fig. 9. Space usage vs window size. (a) Max. candidate size (*uniform*). (b) Max. skyline size (*uniform*).

It is shown that a very small portion of the 2-dimensional dataset needs to be kept. Although this proportion increases with the dimensionality, 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 rest of the experiments.

Fig. 9 evaluates the impact of the sliding window size N on the space efficiency. As depicted in Fig. 9, the space usage grows with the window size because more elements are involved when the window size increases.

Fig. 10 reports the impact of the occurrence probability distribution against the space usage on different datasets. The occurrence probability follows the *normal* distribution and the mean of the occurrence probability P_μ increases from 0.1 to 0.9. It demonstrates that the smaller the average occurrence probability of the elements, the more elements will be kept in $S_{N,q}$. As shown in Fig. 10(a), the size of the candidate decreases with the increase of average occurrence probability. Interestingly, although the candidate size is large with smaller average occurrence probability, the number of probabilistic skyline is small, as illustrated in Fig. 10(b). This is because the small occurrence probability prevents the uncertain objects from becoming probabilistic skyline.

Fig. 11 reports the effect of the probabilistic threshold q on the space efficiency. As expected, both candidate set size and skyline size drop as q increases because more objects are pruned when q becomes large.

We measure the memory usage of SSKY algorithm by the number of pages. Note that the memory usage changes when window slides. Thus, we report the maximum and average memory usage in terms of the number of pages in Fig. 12(a) and (b), respectively, regarding different sizes of sliding windows. As shown, both the average memory usage and maximal memory usage increase with the increment of the sliding window size.

6.2.2. Time efficiency

We evaluate the time efficiency of our continuous query processing techniques, SSKY and MSKY, as well as ad hoc query processing.

Evaluating SSKY and MSKY. Fig. 13 records the delay (processing costs) of each element when it arrives, where n

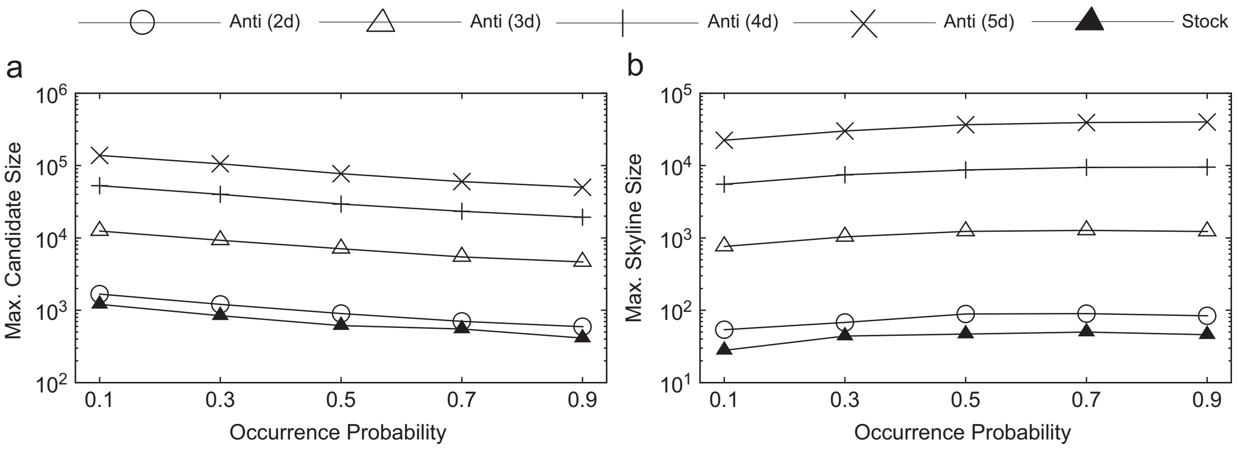

Fig. 10. Space usage vs occurrence probability. (a) Max. candidate size. (b) Max. skyline size.

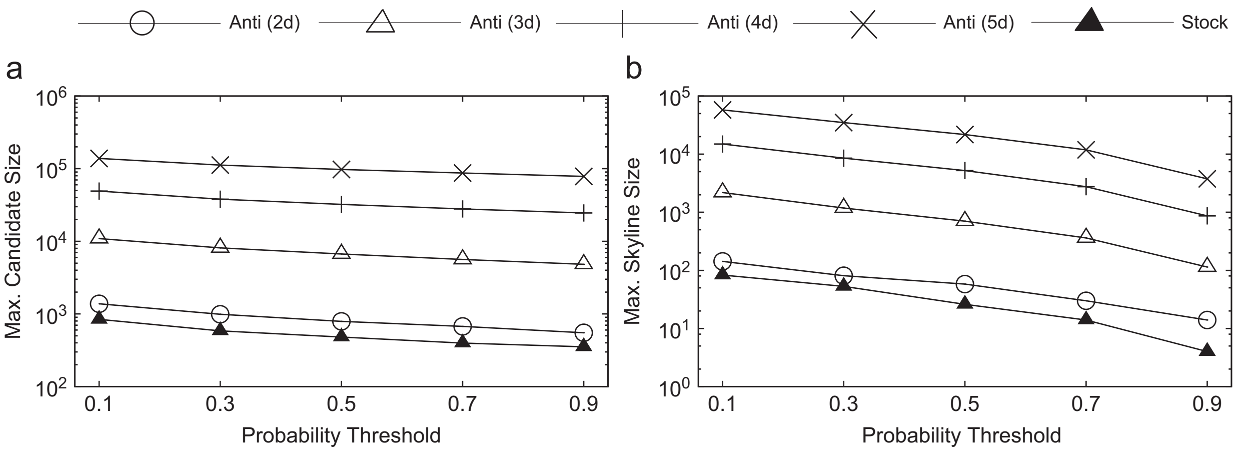

Fig. 11. Space usage vs probability threshold. (a) Max. candidate size (*uniform*). (b) Max. skyline size (*uniform*).

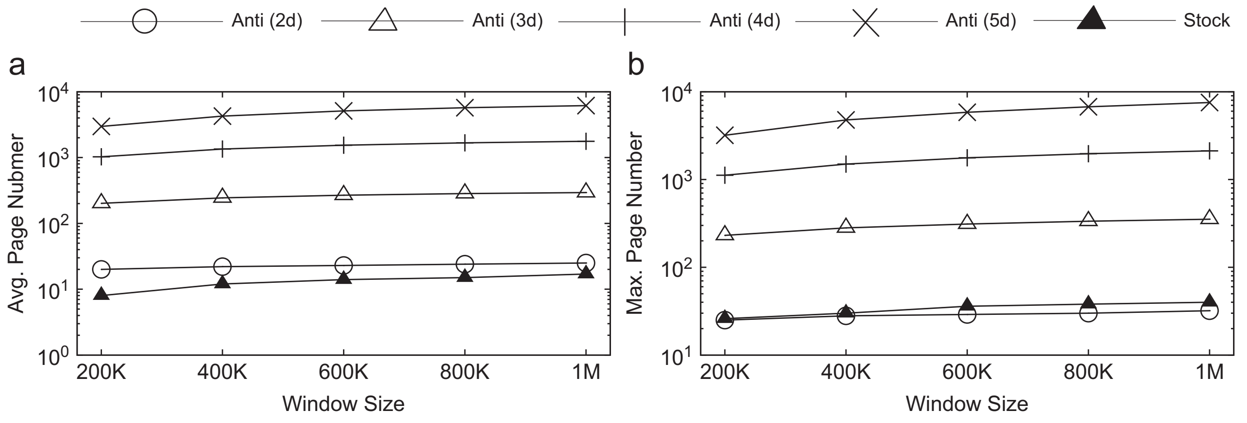

Fig. 12. Memory size vs sliding window size. (a) Avg. memory size. (b) Max. memory size.

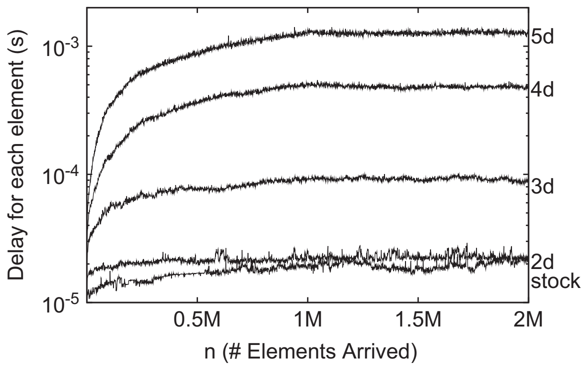

Fig. 13. Time efficiency vs n.

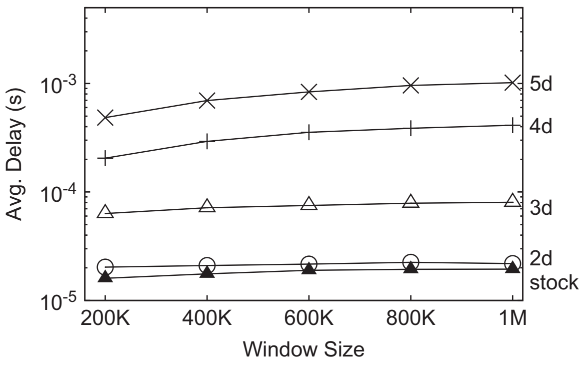

Fig. 14. Avg. delay vs N.

indicates the number of elements arrived so far. As shown, the delay for each element increases before the window is full, and becomes stable after the window is full (number of arrived elements reaches 1 M). This is because in the initialization phase, the sizes of $S_{N,q}$ and $SKY_{N,q}$ are smaller. Also before the window is full, we only need to handle the insertion of new elements, and do not need to process the expiration of elements. Moreover, 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 38 K per second even for *stock*and *anti-correlated* dataset. For 5*d* *anti-correlated* data, our algorithm can still support up to 728 elements per second, which is a medium speed for data streams.

Fig. 14 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 the window size N, as reported in Fig. 9.

Fig. 15 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.

Fig. 16 evaluates the effect of probability threshold q on SSKY. Since both the size of candidate set and the size of

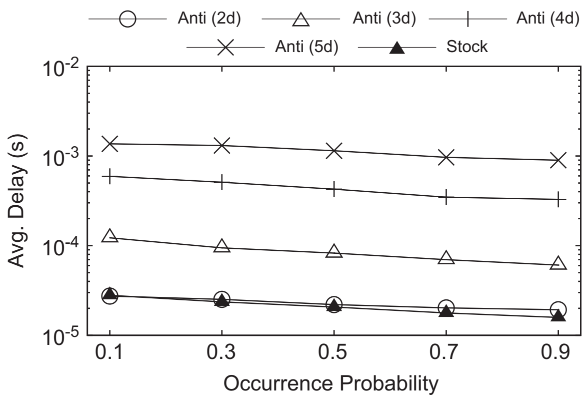

Fig. 15. Avg. delay vs P_μ.

skyline set are small when q is large as depicted in Fig. 11, SSKY is more efficient when q increases.

We study the effectiveness of the index structure and statistics in Fig. 17. If an entry E (in R_1 or R_2) is not accessed based on the dominance relationship (i.e., full dominance or partial dominance) derived by R-tree index, we increase the number of elements pruned by the index by the number of elements in E. Similarly, we increase the number of elements pruned by the statistics by the number of elements in E if E is pruned based on the statistics information. Fig. 17 shows the

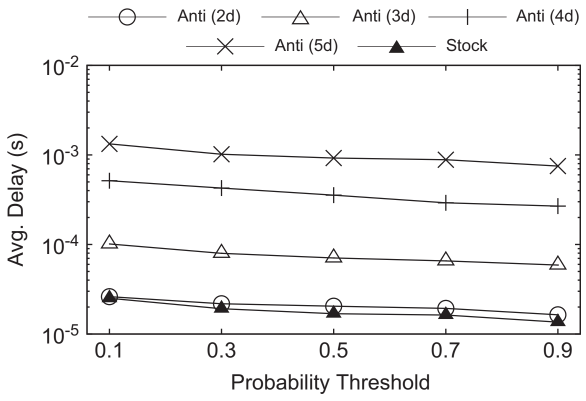

Fig. 16. Avg. delay vs *q*.

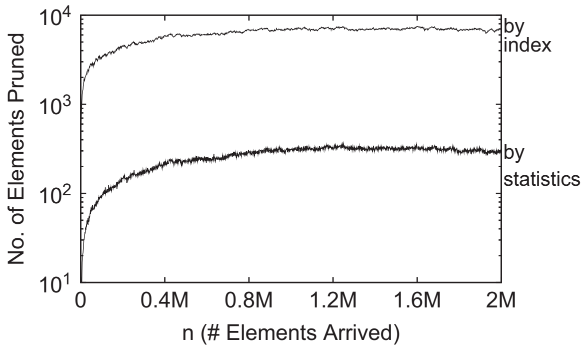

Fig. 17. Effectiveness of pruning.

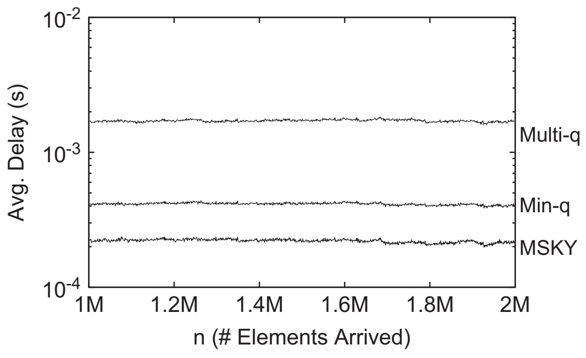

Fig. 18. Time efficiency vs *n*.

number of elements pruned by the index structure and statistics information, respectively. As expected, both the index structure and the statistics information are very useful and hence significantly reduce the number of elements probed.

Fig. 18 illustrates the performance of MSKY, and two naive implementations of MSKY, named *Multi-q* and *Min-q* respectively. As described in Section 5.1, in *Multi-q*, instead of maintaining $|Q|$ disjoint *R*-trees for $SKY_{N,q}$ and one *R*-tree for $S_{N,q}$, we maintain $S_{N,q}$ and $SKY_{N,q}$ for each $q \in Q$. In *Min-q*, we maintain $S_{N,q}$ and $SKY_{N,q}$ for the smallest $q \in Q$ only; besides the regular update cost, whenever the probability or elements change in $SKY_{N,q}$, we scan R_1 to output the skyline results. The results in Fig. 18 show that MSKY is much more efficient than these two implementations.

The last experiment in this subsection evaluates the efficiency of our multi-probability thresholds based continuous query processing techniques MSKY and ad hoc query processing techniques QSKY. Results are reported in Fig. 19(a) and (b), respectively. As expected, Fig. 19(a) shows that the average maintenance cost for the update of each element in MSKY increases when $|Q|$ increases because we need to maintain more *R*-trees. As the multiple ad hoc queries with different probability threshold (q) may share computation if queries are issued at the same time, the average response time of each query decreases with the growth of $|Q|$. This is illustrated in Fig. 19(b).

6.3. *Evaluating TOPK*

The space usage of TOPK is the same as that of SSKY - because we only consider elements with skyline probabilities not smaller than *q*. Therefore, we only evaluate the time efficiency of TOPK in this subsection.

We evaluate the performance of TOPK on different datasets when the window slides from 1 M to 2 M. As shown in Fig. 20, the algorithm is very efficient, especially when the dimensionality is low because of the small candidate and skyline sizes and the use of statistic information. It is shown that the performances of the algorithms are stable because the number of elements in the sliding window remains unchanged after the window becomes full (i.e., $n \geq 1$ M).

We also investigate the impact of the sliding window size *N* which varies from 200 K to 1 M. Fig. 21 shows that the performance of the algorithm slightly degrades when the size of the window increases. As shown in Fig. 9, that is because the number of elements in the skyline and candidate sets grows with the window size.

The impact of the occurrence probability on time efficiency of TOPK is studied in Fig. 22. Probability values follow *normal* distribution and the expected value P_μ varies from 0.1 to 0.9. Similar to SSKY, larger P_μ leads to smaller skyline and candidate set sizes and hence higher time efficiency. The effect of probability threshold *q* is evaluated in Fig. 23. As depicted, the performance of the algorithm is better when *q* is set to larger values because the number of elements in skyline and candidate sets decreases with *q* as shown in Fig. 11.

Figs. 24 and 25 report the effect of the value *k* on TOPK when the occurrence probability follows *uniform* and *normal* distributions, respectively. As shown, the performance of TOPK is not sensitive to the value of *k* because the skyline and candidate set sizes are not dependent on *k*.

6.4. *Evaluating TIME STAMP model*

In this subsection, we evaluate the space and time efficiency of TIME STAMP algorithm.

6.4.1. *Space efficiency*

We first evaluate the space usage of TIME STAMP against different datasets, Inde-Uniform (spatial locations of

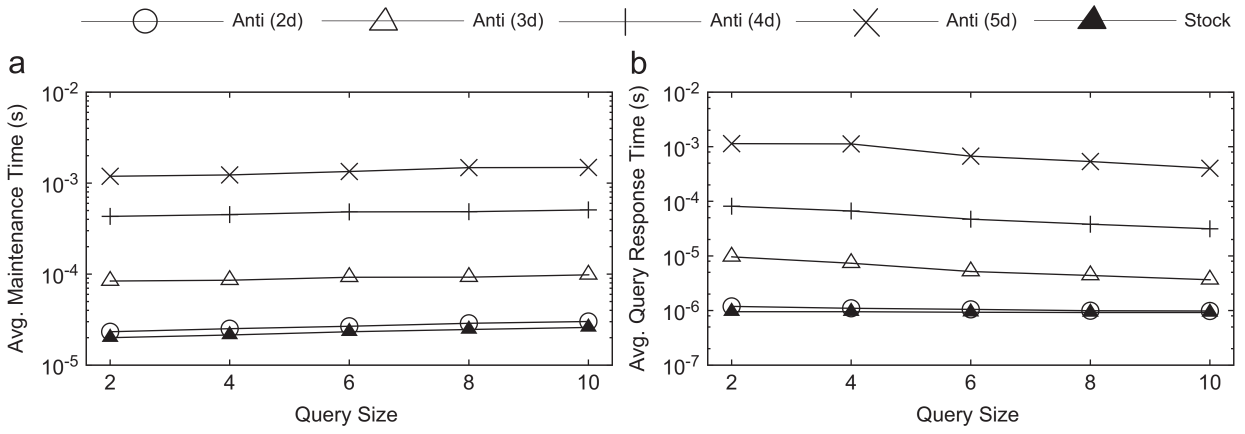

Fig. 19. Query cost vs $|Q|$. (a) Continuous. (b) Ad hoc.

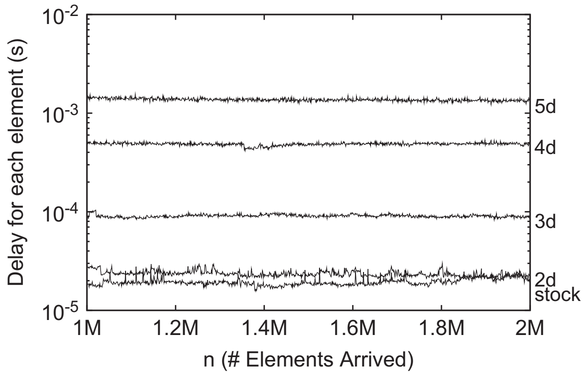

Fig. 20. Time efficiency vs *n*.

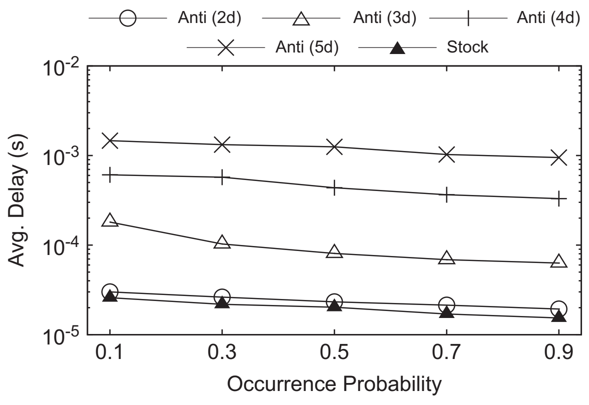

Fig. 22. Avg. delay vs P_μ.

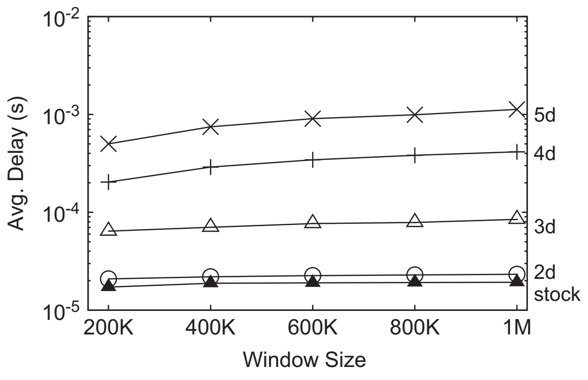

Fig. 21. Avg. delay vs *N*.

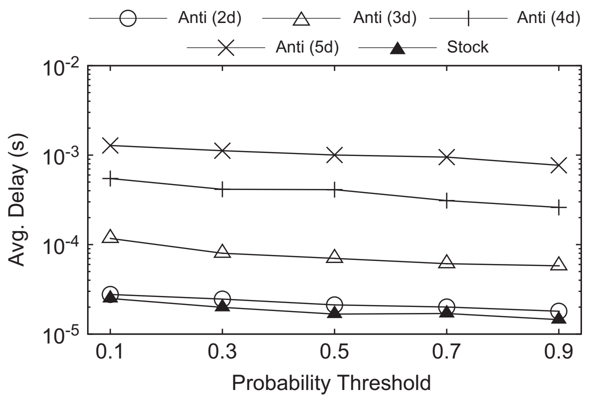

Fig. 23. Avg. delay vs *q*.

elements follow independent distribution and occurrence probability follows uniform distribution), Anti-Uniform, Anti-Normal, and Stock-Uniform (real dataset with uniform occurrence probability distribution). We capture and report the maximum sizes of candidate and skyline sets over the whole data stream. As shown in Fig. 26, TIME STAMP is very space efficient and only a very small portion of the elements need to be kept as candidates. Both candidate and skyline sets grow as dimensionality increases. Nevertheless, even in the most challenging scenario of Anti-Normal distribution where dimensionality is 5, a 95% space saving can be achieved.

As the space usage of the TIME STAMP is similar to that of SSKY against various potential parameters such as the window size, the occurrence probabilities and the probability threshold, we omit the figures due to the space limitation.

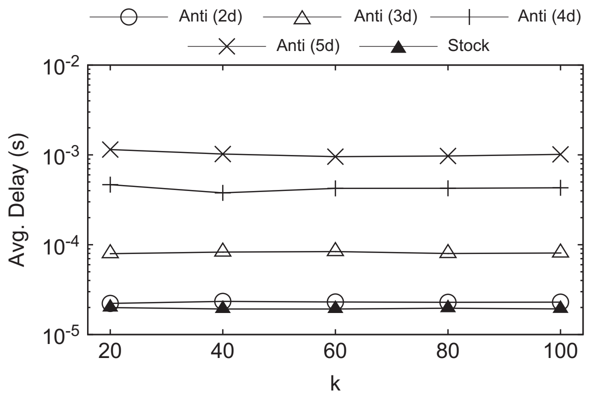

Fig. 24. Avg. delay vs *k* (uniform).

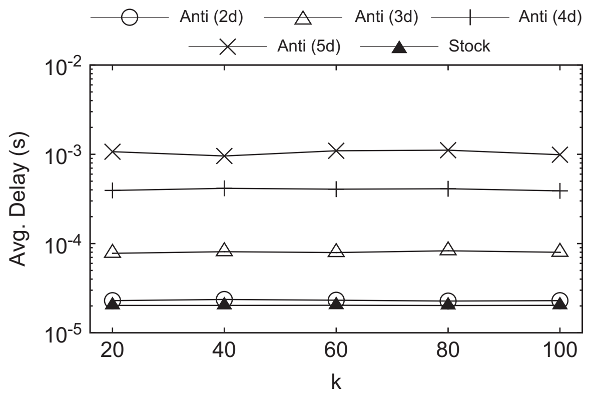

Fig. 25. Avg. delay vs *k* (normal).

6.4.2. *Time efficiency*

To evaluate the time efficiency of TIME STAMP, we record the average delay of the TIME STAMP against different datasets when the time window slides. As illustrated in Fig. 27, the algorithm is quite efficient, especially when the dimensionality is low. Fig. 28 reports the impact of the window size *N*. As shown, the average delay per element increases slowly with the growth of *N*.

The effect of the occurrence probability is depicted in Fig. 29. Elements' occurrence probabilities follow *normal* distribution where the *mean* value P_μ varies from 0.1 to 0.9. Fig. 30 reports the impact of query probability threshold *q*. As shown, the performance of the algorithm becomes more efficient when P_μ or *q* increase.

6.5. *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 and its variations over sliding windows. As expected, the performance of the algorithms, in terms of space usage and processing time, grows slowly with the sliding window size. It is sensitive to the dimensionality and spatial location distribution of the dataset because the number of candidates is large for high dimensional data and "hard" distributions. 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.

7. Related work

We review related work in two aspects, skylines and uncertain data streams.

Skylines. Börzsönyi et al. [4] first study the skyline operator in the context of databases and propose an SQL 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. [8], which takes advantage of a presorting step. SFS is then significantly improved by Godfrey et al. [11]. The *progressive* paradigm that aims to output skyline points without scanning the whole dataset is firstly proposed by Tan et al. [26]. 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. [23] 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 [3,13], skylines for partially ordered value domains [5], skyline cubes [25,28,29], reverse skylines [10], approximate skylines [6,7,17], etc.

Skyline queries processing in data streams is investigated by Lin et al. [20] against various sliding windows. Tao et al. [27] independently develop efficient techniques to compute sliding window skylines. While effectively using *R*-trees for pruning purposes, the techniques are not applicable to uncertain data stream due to the inherent difference between the two environments. To the best of our knowledge, this paper is the first one to address the problem of skyline queries on uncertain data streams.

The skyline query processing on uncertain data is firstly approached by Pei et al. [24] where *Bounding-pruning-refining* techniques are developed for efficient computation. Lian et al. [19] combine reverse skylines [10] 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 [9,14,15,31,32]. Problems such as clustering uncertain data stream [1], frequent items retrieval in probabilistic data streams [30], and sliding window top-*k* queries on uncertain streams [16] 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.

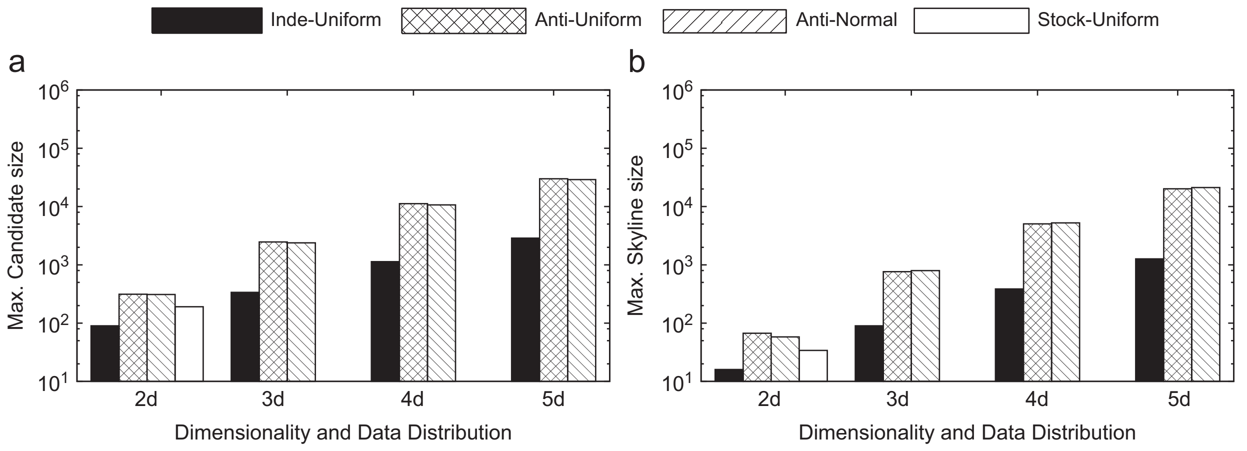

Fig. 26. Space usage vs diff. dataset. (a) Max. candidate size. (b) Max. skyline size.

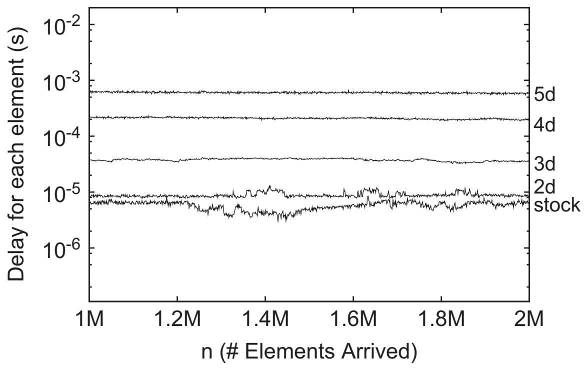

Fig. 27. Time efficiency vs n.

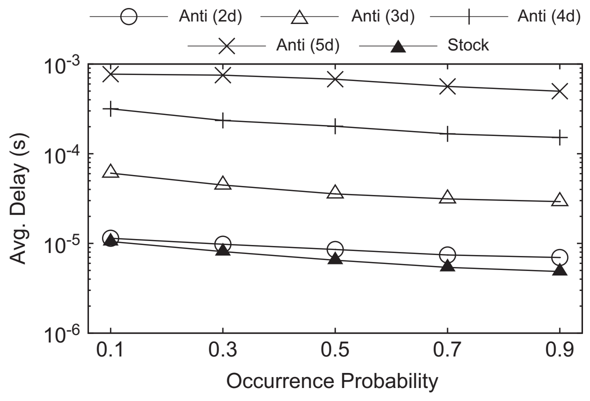

Fig. 29. Avg. delay vs P_μ.

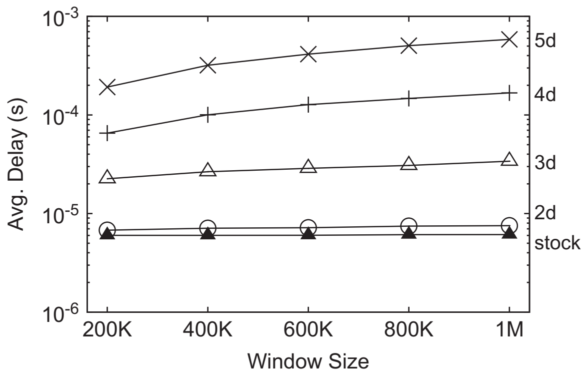

Fig. 28. Avg. delay vs N.

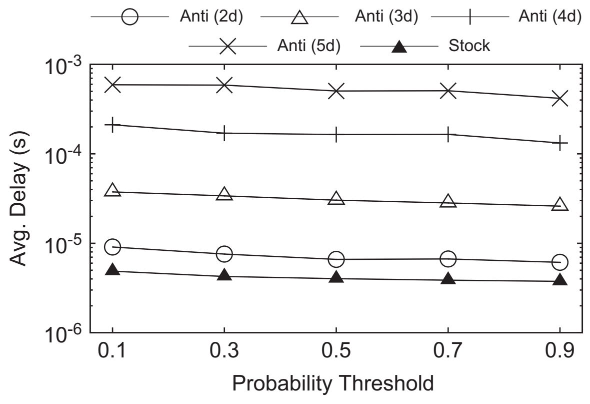

Fig. 30. Avg. delay vs q.

8. 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, as well as elements with different life-spans. Our extensive experiments demonstrate that our techniques can deal with a high-speed data stream in real time.

References

[1] C.C. Aggarwal, P.S. Yu, A framework for clustering uncertain data streams, in: ICDE, 2008.
[2] M.J. Atallah, Y. Qi, Computing all skyline probabilities for uncertain data, in: PODS, 2009.
[3] W.-T. Balke, U. Guntzer, J.X. Zheng, Efficient distributed skylining for web information systems, in: EDBT 2004.
[4] S. Borzsonyi, D. Kossmann, K. Stocker, The skyline operator, in: ICDE, 2001.
[5] C.-Y. Chan, P.-K. Eng, K.-L. Tan, Stratified computation of skylines with partially ordered domains, in: SIGMOD, 2005.
[6] C.-Y. Chan, H.V. Jagadish, K.-L. Tan, A.K.H. Tung, On high dimensional skylines, in: EDBT, 2006.
[7] C.-Y. Chan, H.V. Jagadish, K.-L. Tan, A.K.H. Tung, Z. Zhang, Finding k-dominant skylines in high dimensional space, in: SIGMOD, 2006.
[8] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with presorting, in: ICDE, 2003.
[9] G. Cormode, M. Garofalakis, Sketching probabilistic data streams, in: SIGMOD, 2007.
[10] E. Dellis, B. Seeger, Efficient computation of reverse skyline queries, in: VLDB, 2007.
[11] P. Godfrey, R. Shipley, J. Gryz, Maximal vector computation in large datasets, in: VLDB, 2005.
[12] Y.-W. Huang, N. Jing, E.A. Rundensteiner, Spatial joins using *R*-trees: breadth-first traversal with global optimizations, in: VLDB, 1997.
[13] Z. Huang, C.S. Jensen, H. Lu, B.C. Ooi, Skyline queries against mobile lightweight devices in MANETs, in: ICDE, 2006.
[14] T. Jayram, S. Kale, E. Vee, Efficient aggregation algorithms for probabilistic data, in: SODA, 2007.
[15] T.S. Jayram, A. McGregor, S. Muthukrishan, E. Vee, Estimating statistical aggregrates on probabilistic data streams, in: PODS, 2007.
[16] C. Jin, K. Yi, L. Chen, J.X. Yu, X. Lin, Sliding-window top-*k* queries on uncertain streams, in: VLDB, 2008.
[17] V. Koltun, C. Papadimitriou, Approximately dominating representatives, in: ICDT, 2005.
[18] D. Kossmann, F. Ramsak, S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: VLDB, 2002.
[19] X. Lian, L. Chen, Monochromatic and bichromatic reverse skyline search over uncertain databases, in: SIGMOD, 2008.
[20] X. Lin, Y. Yuan, W. Wang, H. Lu, Stabbing the sky: efficient skyline computation over sliding windows, in: ICDE, 2005.
[21] X. Lin, Y. Zhang, W. Zhang, M.A. Cheema, Stochastic skyline operator, in: ICDE, 2011.
[22] J. Nievergelt, H. Hinterberger, K. Sevcik, The grid file: an adaptable, symmetric mulitkey file structure, in: TODS, 1984.
[23] D. Papadias, Y. Tao, G. Fu, B. Seeger, An optimal progressive algorithm for skyline queries, in: SIGMOD, 2003.
[24] J. Pei, B. Jiang, X. Lin, Y. Yuan, Probabilistic skylines on uncertain data, in: VLDB, 2007.
[25] J. Pei, W. Jin, M. Ester, Y. Tao, Catching the best views of skylin: a semantic approach based on decisive subspaces, in: VLDB, 2005.
[26] K.-L. Tan, P. Eng, B.C. Ooi, Efficient progressive skyline computation, in: VLDB, 2001.
[27] Y. Tao, D. Papadias, Maintaining sliding window skylines on data streams, in: TKDE, 2006.
[28] T. Xia, D. Zhang, Refreshing the sky: the compressed skycube with efficient support for frequent updates, in: SIGMOD, 2006.
[29] Y. Yuan, X. Lin, Q. Liu, W. Wang, J.Y. Xu, Q. Zhang, Efficient computation of the skyline cube, in: VLDB, 2005.
[30] Q. Zhang, F. Li, K. Yi, Finding frequent items in probabilistic data, in: SIGMOD, 2008.
[31] Alfredo Cuzzocrea, Retrieving accurate estimates to OLAP queries over uncertain and imprecise multidimensional data streams, SSDBM (2011) 575–576. http://dx.doi.org/10.1007/978-3-642-22351-8_43.
[32] Alfredo Cuzzocrea, Providing probabilistically-bounded approximate answers to non-holistic aggregate range queries in OLAP, DOLAP (2005) 97–106. http://doi.acm.org/10.1145/1097002.1097020.