Online Interval Skyline Queries on Time Series

Bin Jiang, Jian Pei

School of Computing Science, Simon Fraser University, Burnaby, BC, Canada

{bjiang, jpei}@cs.sfu.ca

***Abstract*— In many applications, we need to analyze a large number of time series. Segments of time series demonstrating dominating advantages over others are often of particular interest. In this paper, we advocate *interval skyline queries*, a novel type of time series analysis queries. For a set of time series and a given time interval $[i : j]$, an interval skyline query returns the time series which are not dominated by any other time series in the interval. We illustrate the usefulness of interval skyline queries in applications. Moreover, we develop an on-the-fly method and a view-materialization method to online answer interval skyline queries on time series. The on-the-fly method keeps the minimum and the maximum values of the time series using radix priority search trees and sketches, and computes the skyline at the query time. The view-materialization method maintains the skylines over all intervals in a compact data structure. Through theoretical analysis and extensive experiments, we show that both methods only require linear space and are efficient in query answering as well as incremental maintenance.**

I. Introduction

In many applications such as environment surveillance, telecommunication, and finance market analysis, we need to analyze a large number of time series. More often than not, segments of time series demonstrating dominating advantages over others are of particular interest.

Example 1 (Motivation): A power supplier needs to analyze the electricity consumption of different regions in the service area. The power consumption in a region over time can be captured by a time series. For illustration, Figure 1 shows three synthesized time series of three regions in a monitoring period June 15-23. An analyst may ask, "*In the week of June 16-22, which regions have high power consumption?*" Here, June 16-22 is a query interval.

Region 2 is interesting to the analyst since it has the highest average power consumption in the query interval. Moreover, region 1 is also interesting since it has the highest power consumption on June 20. The analyst may want to take a closer look to find out the cause of the burst. On the other hand, region 3 should not be returned to the analyst for this query since the power consumption of region 3 on every day in the query interval is lower than that of region 2. ■

To compare multiple time series in a query interval, it is interesting to find the time series that are not subsumed by any other time series. Technically, a time series s is interesting if, in the query interval, there does not exist another time series s' such that s' is better than s on at least one timestamp and s' is not worse than s on every timestamp. In order words, s is a skyline point [1] if we consider each timestamp in the query interval as a dimension and each time series as a point in the space constructed as such. In this paper, we model such queries as interval skyline queries.

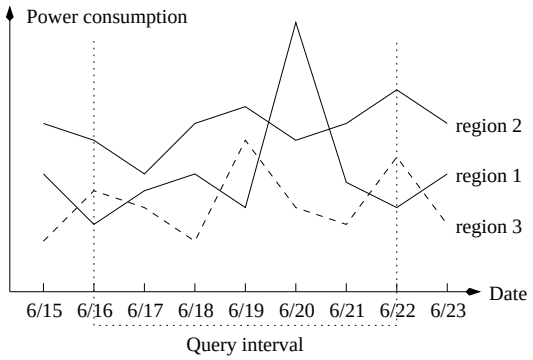


Fig. 1. A set of power consumption time series.

We can easily give more application examples where interval skyline queries are useful. For instance, consider finding popular web pages in terms of the number of hits. A web analytics manager may be interested in the popular pages in a specific time interval. Page p_1 is more popular than page p_2 if the number of hits of p_1 is larger than or equal to the number of hits of p_2 on every day in the query interval, and p_1 has strictly more hits than p_2 on at least one day.

Since the skyline operator was introduced to the database community by Börzsönyi, *et al.* [1], a large amount of research has been dedicated to developing efficient skyline computation method. Can interval skyline queries be answered straightforwardly using an existing skyline computation method?

A naïve method is that, every time a query interval is given, we apply an efficient skyline computation method on the time series in the space formed by the query interval. However, such a method is not efficient due to three reasons.

First, most of the existing skyline computation methods, as analyzed in Section V, consider answering skyline queries from scratch and thus cannot be applied directly for online query answering. They also do not consider the incremental maintenance against updates on adding and removing dimensions, i.e., timestamps in the time series context.

Second, for two overlapping query intervals, the naïve method does not explore any sharing in computation. For example, in Figure 1, after observing the skyline regions in interval June 16-22, an analyst may further be interested in the skyline regions in some other intervals such as June 19-22 and June 1-30. While the naïve method has to apply the skyline computation on those intervals from scratch, can we

build some compact data structure so that the computation for different intervals can share?

Last, most of the existing skyline computation methods can handle low dimensional cases well, but may not be capable in high dimensional cases. It is not rare at all that time series contain at hundreds of or even more timestamps. Therefore, how to efficiently answer interval skyline queries on intervals of tens or hundreds of timestamps is far from trivial.

To address the above challenges, can we develop some effective data structures and methods so that interval skyline queries can be answered efficiently? Moreover, time series data is often collected incrementally. The data structures have to be maintainable with low cost for incremental updates.

There are a few recent studies on skyline computation on data streams [2], [3], which consider how to efficiently maintain the skyline points against a stream of data points in a fixed space. Different from those studies, incremental maintenance of data structures for online interval skyline queries on time series have to address an orthogonal problem: data on new timestamps as new dimensions keeps arriving while the number of time series (i.e., data points in the context of data streams) is fixed in our problem. Thus, the methods in [2], [3] are not applicable here.

To the best of our knowledge, we are the first to model and tackle the problem of interval skyline queries on time series data. We make several contributions in this paper.

First, we model interval skyline queries, a novel type of queries on time series data. We show that interval skyline queries are interesting. Second, we devise online query answering methods. Particularly, we develop an on-the-fly method and a view-materialization method to answer interval skyline queries on time series. The on-the-fly method keeps the minimum and the maximum values of the time series using radix priority search trees and sketches, and computes the skyline at the query time. The view-materialization method maintains the skylines over all intervals in a compact data structure. We also consider the incremental maintenance of the data structures. Through theoretical analysis and extensive experiments, we show that both methods only require linear space and are efficient in query answering as well as incremental maintenance.

The rest of the paper is organized as follows. In Section II, we define interval skyline queries formally, and explore their properties. We develop the on-the-fly method in Section III and the view-materialization method in Section IV, followed by an experimental study in Section VI. Finally, Section VII concludes the paper.

II. Interval Skyline Queries

A *time series* s consists of a set of $(value, timestamp)$ pairs. The data values are ordered with respect to the timestamps. To keep our discussion simple, we assume that all timestamps take positive integer values. We denote the value of s at timestamp i by $s[i]$, and write a time series s as a sequence of values $s[1], s[2], \ldots$. We assume that all time series are synchronized. That is, each time series s has a value $s[i]$ on a timestamp $i > 0$.

A *(time) interval* $[i : j]$ $(i \leq j)$ specifies a range in time. Let $s[i : j] = s[i], s[i+1], \ldots, s[j]$ $(i \leq j)$ denote the subsequence of time series s in interval $[i : j]$. We write $k \in [i : j]$ if $i \leq k \leq j$. For two intervals $[i_1 : j_1]$ and $[i_2 : j_2]$, $[i_1 : j_1] \subseteq [i_2 : j_2]$ if $i_2 \leq i_1$ and $j_1 \leq j_2$.

Time series s is said to *dominate* time series q in interval $[i : j]$, denoted by $s \succ_{[i:j]} q$, if $\forall k \in [i : j]$, $s[k] \geq q[k]$; and $\exists l \in [i : j]$, $s[l] > q[l]$.

Given a set S of time series and interval $[i : j]$, the *interval skyline* is the set of time series that are not dominated by any other time series in $[i : j]$, denoted by

$$Sky[i : j] = \{s \in S | \nexists s' \in S, s' \succ_{[i:j]} s\}.$$

Interval skyline has the following property.

Property 1 (Interval skyline): Given a set S of time series and an interval $[i : j]$, time series $s \in S$ is in $Sky[i : j]$ if there exist timestamps $k_1, \ldots, k_l$ $(i \leq k_1 < \cdots < k_l \leq j)$ such that $\frac{\sum_{x=1}^{l} s[k_x]}{l}$ achieves the maximum among S on those timestamps, and s is the only such a time series. That is,

$$s = \arg\max_{s' \in S} \{\frac{\sum_{x=1}^{l} s'[k_x]}{l}\}.$$

Proof: We prove by contradiction. Assume $\frac{\sum_{x=1}^{l} s[k_x]}{l}$ is the maximum and s is the only time series achieving the maximum, but $s \notin Sky[i : j]$. Then, there must be another time series $s' \in S$, $s' \neq s$ such that s' dominates s. Thus, $s'[k_x] \geq s[k_x]$ for $1 \leq x \leq l$ and there exists $1 \leq x_0 \leq l$ such that $s'[k_{x_0}] > s[k_{x_0}]$. As a consequence,

$$\sum_{x=1}^{l} s[k_x] < \sum_{x=1}^{l} s'[k_x].$$

Thus,

$$\frac{\sum_{x=1}^{l} s[k_x]}{l} < \frac{\sum_{x=1}^{l} s'[k_x]}{l}.$$

In other words, $\frac{\sum_{x=1}^{l} s[k_x]}{l}$ is not the maximum, which leads to a contradiction. ■

Property 1 is particularly interesting for interval skyline queries on time series. Interval average aggregates are often important for time series analysis. The answer to an interval skyline query contains all time series which achieve the highest average aggregate values on any subsets of timestamps. As one application, an interval skyline query with an interval of m timestamps can be used to obtain the answers to all average aggregate queries on $2^m - 1$ timestamp subsets. Interval skyline queries can be very effective in time series analysis.

We note that the converse proposition of Property 1 does not always hold, that is, a skyline time series may not have the maximum sum value on any combination of timestamps.

Many time series applications involve continuous updates over time. Recent data is often considered more important. Therefore, it is practical to assume that we are always asked

TABLE I

A SUMMARY OF FREQUENTLY USED NOTIONS.

Notion	Meaning
s, q	time series.
$[i:j]$ $(i \leq j)$	an interval.
$Sky[i:j]$	the skyline in interval $[i:j]$.
$NRSky[i:j]$	the non-redundant skyline in $[i:j]$.
t_c	the most recent timestamp.
w	the size of the base interval.
$W = [t_c - w + 1 : t_c]$	the base interval of time series.
n	the number of time series in the data set.

to maintain the most recent w timestamps, where w can be a large number. Let t_c be the most recent timestamp. We call interval $W = [t_c - w + 1 : t_c]$ the base interval. Whenever a new timestamp t_c+1 comes, the oldest one t_c-w+1 expires. Consequently, the base interval becomes $W' = [t_c - w + 2 : t_c + 1]$. This is similar to the sliding window model for data stream analysis. In this paper, we tackle the following problem.

Problem Definition Given a set of time series S such that each time series is in the base interval $W = [t_c - w + 1 : t_c]$, we want to maintain a data structure $\mathcal{D}$ such that any interval skyline queries in interval $[i:j] \subseteq W$ can be answered efficiently using $\mathcal{D}$. ■

Table I summaries the notions frequently used in this paper.

III. AN ON-THE-FLY METHOD

In this section, we give a simple on-the-fly method to answer interval skyline queries. The on-the-fly method keeps the minimum and the maximum values for each time series and computes the interval skyline at query time.

We first present an interval skyline query answering algorithm and then describe how to make it online.

A. An Interval Skyline Query Answering Algorithm

For time series s, let $s.max$ denote the maximum value of s in the base interval W, i.e., $s.max = \max_{k \in W}\{s[k]\}$. Let $s.min[i:j]$ denote the minimum value of s in interval $[i:j] \subseteq W$, i.e., $s.min[i:j] = \min_{k \in [i:j]}\{s[k]\}$.

Using the maximum values and the minimum values in intervals of the time series, we can determine the domination relation between some time series without checking the details.

Lemma 1 (Max-Min): For two time series p, q and interval $[i:j] \subseteq W$, if $s.min[i:j] > q.max$, then s dominates q in $[i:j]$.

Proof: $\forall k \in [i:j]$, $s[k] \geq s.min[i:j] > q.max \geq q[k]$. Thus, s dominates q in $[i:j]$. ■

To some extent, for a time series s, $s.max$ reflects the capability of s not being dominated by other time series. If $s.max$ is small, likely s may be dominated by some other time series since s has a value at most $s.max$ on every timestamp.

Lemma 1 leads to Algorithm 1, a simple interval skyline query answering algorithm. We iteratively process the time series in S in their max value descending order. This order is maintained in a sorted list L (line 1). We also keep the skyline candidates in Sky. Let $maxmin$ be the maximum $min[i:j]$ value of time series in Sky. For a time series s, if s is not dominated by any time series in Sky, then s is a skyline candidate (line 8); otherwise, s is discarded. Meanwhile, we also remove false positives in Sky (line 7). $maxmin$ is updated accordingly (line 9). The algorithm terminates as early as $maxmin$ is greater than the max value of the next time series in list L (line 5). Because the remaining time series in the list all have max values less than $maxmin$, according to Lemma 1, they are dominated.

Algorithm 1 The on-the-fly query algorithm.

Input: a set S of time series, an interval $[i:j]$;
Output: the skyline in $[i:j]$;
Description:

1: L = a sorted list of the time series in S in the descending order of their $s.max$.
2: $Sky = \emptyset$;
3: $maxmin = -\infty$;
4: let s be the first time series in L;
5: **while** L is not empty and $maxmin \leq s.max$ **do**
6: **if** no time series in Sky dominates s in $[i:j]$ **then**
7: remove the time series from Sky dominated by s in $[i:j]$;
8: $Sky = Sky \cup \{s\}$
9: $maxmin = \max_{q \in Sky}\{q.min[i:j]\}$;
10: **end if**
11: s = the next time series in L;
12: **end while**
13: **return** Sky;

TABLE II

A SET OF TIME SERIES DATA.

Time series ID	Timestamps					
	1	2	3	4	5	...
s_1	4	3	2	5	5	...
s_2	5	5	1	5	5	...
s_3	2	2	5	3	4	...
s_4	1	1	3	4	2	...
s_5	3	4	4	1	3	...

TABLE III

THE SORTED LIST L IN ALGORITHM 1.

Time series	s_2	s_3	s_5	s_1	s_4
max	5	5	4	4	3
$min[2:3]$	1	2	4	2	1

Example 2 (The on-the-fly query algorithm): Table II shows 5 time series. Suppose currently the data in interval $W = [1:3]$ is maintained. Let us compute the skyline in interval $[2:3]$.

Table III shows the sorted list L of the 5 time series along with their max and $min[2:3]$ values.

We first put s_2 in the skyline candidate list and update $maxmin$ to 1. Then we process s_3. s_2 and s_3 do not dominate each other in $[2:3]$. Thus, s_3 is pushed into the candidate list.

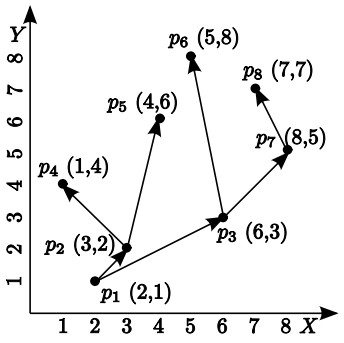


Fig. 2. An example of a radix priority search tree.

Consequently, $maxmin = 2$.

Next, s_5 is processed and becomes a candidate. $maxmin$ is updated to 4. The next time series in list L is s_1, which is dominated by s_5 and thus discarded. Now $maxmin$ is larger than $s_4.max$, thus the algorithm terminates. $Sky[2:3] = \{s_2, s_3, s_5\}$. ■

Algorithm 1 is not ready for online query answering yet. The algorithm needs to check the max value for each time series. Moreover, the algorithm needs to check the $min[i:j]$ value for the query interval $[i:j]$. Checking those values on-the-fly is apparently costly. Can we maintain a succinct data structure so that those values can be obtained efficiently online? Moreover, can the data structure be incrementally maintained with low cost? We answer these questions in the rest of this section.

B. Online Interval Skyline Query Answering

We use a radix priority search tree for each time series to maintain the $min[i:j]$ values for all possible intervals. Moreover, we maintain a sketch to keep the max values for each time series. In this subsection, we describe these data structures and their incremental maintenance methods.

1) Radix Priority Search Tree: A radix priority search tree [4] is a two-dimensional data structure that allows efficient range queries where the ranges on at least one dimension are unbounded. It is a hybrid of a heap on one dimension and a binary search tree on the other dimension.

A radix priority search tree contains a set of points in a two dimensional space (X, Y). A tree node contains a range $[x_1, x_2]$ on dimension X and a data point p of the smallest value on dimension Y among all points in the range $[x_1, x_2]$ on dimension X and not in any ancestor node. The range of the root on dimension X is the domain of dimension X (we assume that dimension X has a finite domain). Recursively, the range of a node on dimension X is divided exactly in half. The left half range on dimension X becomes the range of the left child, and the right half goes to the right child. It is easy to see that a radix priority search tree uses linear space with respect to the number of points indexed.

Example 3 (Radix Priority search tree): Figure 2 shows a radix priority search tree with 8 points. The root corresponds to range $[1, 8]$ on dimension X and contains p_1 that has the smallest value on dimension Y. The rest of the points are divided into two partitions. p_2, p_4 and p_5 are in the left partition with range $[1, 4]$ on dimension X; p_3, p_6, p_7 and p_8 are in the right partition with range $[5, 8]$ on dimension X. Similarly, p_2 and p_3 are in the roots of the left and the right subtrees, respectively. The tree is built recursively. ■

Using a radix priority search tree, we can find the point of the smallest value on dimension Y among those whose values on X dimension falling in a query range $[lowX, highX]$. We start with the root node N. If the point at N lies in $[lowX, highX]$ on dimension X, then the Y value of this point is the solution. Otherwise, we recursively search the subtrees whose ranges on X dimension overlap with the query range until we find such a point. If both subtrees of N are searched and two points are returned from the two subtrees, the point with the smaller value on dimension Y is the answer. [4] proves that finding the smallest Y value in a radix priority search tree with h levels takes time $O(h)$. Besides, an insertion or a deletion operation only moves the existing nodes in the radix priority search tree down or up along a single path in the tree. Thus, those operations can be done in $O(h)$.

2) Maintaining a Radix Priority Search Tree for Each Time Series: A radix priority search tree involves one heap dimension Y and one binary search tree dimension X. To process a time series, we use the time dimension (i.e., the timestamps) as the binary tree dimension X and the data values as the heap dimension Y.

Since the base interval W always consists of w timestamps represented by w consecutive natural numbers, we can apply the module w operation to the timestamps of W, i.e., $\forall k \in W$, we map k into $k \mod w$. By doing this, we map W into a fixed domain of X, which is $\{0, \ldots, w-1\}$. Under the mapping, the value at each timestamp of a time series corresponds to one point in the radix priority search tree. This fixed X domain results in a balanced radix priority search tree with minimum height $\lceil \log w \rceil$. Thus, we can obtain the optimal performance in query answering and update using the radix priority search tree.

Let t_c be the most recent timestamp. Let $w_t = t_c \mod w$. Then, $W = [t_c - w + 1 : t_c]$ is mapped into $w_t + 1, w_t + 2, \ldots, w-1, 0, 1, \ldots, w_t$ with the timestamp order preserved. When the new timestamp $t_c + 1$ comes, the base interval becomes $W' = [t_c - w + 2 : t_c + 1]$. The mapping of timestamps $k \in [t_c - w + 2 : t_c]$ does not change. Thus, those points in the radix priority search tree corresponding to the values at timestamps $t_c - w + 2, \ldots, t_c$ do not need to be changed. The new timestamp $t_c + 1$ and the expired timestamp $t_c - w + 1$ are mapped to the same value $w_t + 1$ in the X dimension. Thus, they correspond to the same point in the radix priority search tree. We only need to substitute the Y value of this point by the value at the new timestamp. This can be done using at most one insertion and one deletion on the tree which take $O(\log w)$ time. The tree remains balanced after the update.

Example 4 (Radix Priority search tree mapping): Suppose the current base interval is $W = [1:3]$ and $w = 3$. The time series s_1 in Table II is mapped into 3 points $(1, 4)$, $(2, 3)$, and $(3, 2)$ in the radix priority search tree. When the base interval becomes $W' = [2:4]$, s_1 is mapped into $(1, 5)$, $(2, 3)$, and $(3, 2)$. The Y value of the point with $X = 1$ is changed from 4 to 5. ■

3) Retrieving $min[i:j]$ Values: In order to retrieve the $min[i:j]$ value in interval $[i:j]$, we map $[i:j]$ into the X domain. Let $w_i = i \mod w$ and $w_j = j \mod w$. One of the following two cases may arise.

- If $w_i \leq w_j$, then $[i:j]$ is mapped into a single X-interval $[w_i, w_j]$. We query the radix priority search tree for the point with the minimum Y value among the points whose X values falling into range $[w_i, w_j]$. The Y value returned is $min[i:j]$.
- If $w_i > w_j$, then $[i:j]$ is mapped into two separate X-intervals, $[0, w_j]$ and $[w_i, w-1]$. We issue two queries to the radix priority search tree, respectively, for the points with the minimum Y values among the points whose X values falling into ranges $[0, w_i]$ and $[w_j, w-1]$. $min[i:j]$ is the smaller one between the two returned Y values.

Clearly, in each of the above two cases, the cost of finding $min[i:j]$ is $O(\log w)$.

4) Maintaining max Values Using Sketches: The space requirement of maintaining the maximum element of a streaming time series s in the base interval W is $\log w$ [5]. We represent s as a set of $(value, timestamp)$ pairs. We only maintain pairs that have a value larger than the values in all pairs with newer timestamps. That is, a pair (v, t) is maintained if there is no other pair (v', t') such that $v' \geq v$ and $t' > t$. In this way, we only keep $\log w$ pairs on average [6].

In fact, the set of w value-timestamp pairs can be viewed as a set of w points in two dimensional space $(value, timestamp)$. The pairs maintained form the skyline of this set of points if we prefer large values and large timestamps on these two dimensions. Given w independently distributed points, the expected number of points in the skyline is $O(\log w)$ [6].

With the sketches, we can find the maximum value in the base interval W in $O(1)$ time. The average time cost for update when a new timestamp arrives is $O(\log w)$.

Example 5 (Sketches for max values): Suppose the current base interval is $W = [1:3]$. For time series s_1 in Table II, its sketch of max values consists of 3 value-timestamp pairs $(4, 1)$, $(3, 2)$ and $(2, 3)$. When the base interval becomes $W' = [2:4]$, the updated sketch contains only one pair $(5, 4)$. ■

In Algorithm 1, all time series are sorted in the descending order of their maximum values in W. When a new timestamp arrives, the base interval changes. The maximum values of some time series may also change. In real applications, the variance of the values of a time series is often small within a time interval. The variance of its maximum values is even smaller. Thereby, after update, the previous sorted list is substantially sorted with a few inversions. In this case, we use insertion sort to re-sort the previous sorted list. Insertion sort takes linear time $O(w + d)$ where d is the number of inversions.

For every time series, we use $O(w)$ space to store a radix priority search tree and $O(\log w)$ space to maintain the sketch of the max values. The total space for n time series is $O(nw)$.

The time costs for updating the radix priority search tree and updating the sketch are both $O(\log w)$. Thus, the amortized update cost is $O(n \log w)$.

IV. A View-materialization Method

The on-the-fly method presented in the previous section stores some critical statistics to answer interval skyline queries. In this section, we tackle the interval skyline queries from an angle other than the on-the-fly method. We explore how materialization of skylines can help to answer queries efficiently.

The view-materialization method maintains a set of *non-redundant interval skylines* in the base interval. The base interval with width w has $\frac{w(w-1)}{2}$ distinct sub-intervals. A time series s is called a *non-redundant skyline time series* in interval $[i:j]$ if (1) s is in the skyline in interval $[i:j]$; and (2) s is not in the skyline in any subinterval $[i':j'] \subset [i:j]$. In this case, we also call $[i:j]$ a *minimal skyline interval* of s. The non-redundant interval skyline of $[i:j]$, denoted by $NRSky[i:j]$, consists of all non-redundant skyline time series in $[i:j]$.

Non-redundant interval skylines have the following property.

Theorem 1 (Number of minimal skyline intervals): In a base interval W of w timestamps, a time series s has at most w minimal skyline intervals.

Proof: Assume that s has more than w minimal skyline intervals. There are only w timestamps in the base interval W. Due to the pigeonhole principle, there must be two different skyline intervals $[i:j_1]$ and $[i:j_2]$ which share the same starting timestamp. Since either $j_1 < j_2$ or $j_1 > j_2$, one of the two intervals is not a minimal skyline interval. ■

Based on Theorem 1, the space to store all non-redundant interval skylines is $O(nw)$. When w is large, this is significantly better than the $O(nw^2)$ space used to store the exact interval skylines.

A. Query Answering Algorithm

Following with the definitions of non-redundant skyline and minimal skyline interval, we immediately have the following result.

Proposition 1 (Sub-interval skylines): Given a time series s and an interval $[i:j]$, if there does not exist an interval $[i':j'] \subseteq [i:j]$ such that s is a non-redundant skyline in $[i':j']$, then s is not a skyline in interval $[i:j]$. ■

According to Proposition 1, to find skylines in interval $[i:j]$, we only need to consider the non-redundant skylines whose

Algorithm 2 The view-materialization query algorithm.

Input: a set of non-redundant interval skylines, an interval $[i : j]$;
Output: the skyline in $[i : j]$;
Description:

```
1: Sky = ∅;
2: for each interval [i' : j'] ⊆ [i : j] do
3:    for each time series s ∈ NRSky[i' : j'] do
4:       if no time series in NRSky[i' : j'] dominates s in
         [i : j] then
5:          Sky = Sky ∪ {s};
6:       end if
7:    end for
8: end for
9: return Sky;
```

minimal skyline intervals are within $[i : j]$. The following rule can be used to qualify the interval skyline-ship of such time series.

Lemma 2 (Interval skyline): Given a time series s and an interval $[i : j]$, if for all interval $[i' : j'] \subseteq [i : j]$ such that $s \in NRSky[i' : j']$, $s \not\succ_{[i:j]} s'$ for any time series $s' \in NRSky[i' : j']$, then $s \in Sky[i : j]$.

Proof: Assume $s \notin Sky[i : j]$. Then, there exists another time series q dominating s in $[i : j]$. Therefore, $q[k] \geq s[k]$ for $\forall k \in [i : j]$. For any interval $[i' : j']$ such that $s \in NRSky[i' : j']$, either $q \succ_{[i':j']} s$ or q is also in $NRSky[i' : j']$. A contradiction. ■

Suppose all non-redundant interval skylines are materialized. Using Proposition 1 and Lemma 2, Algorithm 2 gives a method to retrieve the skyline in interval $[i : j]$ from the set of non-redundant interval skylines. To compute the skyline in interval $[i : j]$, Algorithm 2 only unions the non-redundant interval skylines over all intervals contained in $[i : j]$ and removes the false positives: those that fail Lemma 2. We note that Proposition 1 and Lemma 2 are similar to Lemmas 1 and 2 in [7], respectively.

In general, the query time is linear to the number of time series in $Sky[i : j]$ with a small overhead for removing false positives using Lemma 2.

Example 6 (The view-materialization query algorithm): Suppose the current base interval is $W = [2 : 4]$. Table IV shows the non-redundant interval skylines in all sub-intervals for the set of time series in Table II. To compute the interval skyline in $[3 : 4]$, we union the non-redundant interval skylines in $[3 : 3]$, $[4 : 4]$ and $[3 : 4]$ to get the result $\{s_1, s_3, s_4\}$. We see that s_2 is a false positive since it is dominated by s_1 in $[3 : 4]$. Clearly, s_1 and s_2 are both in $NRSky[4 : 4]$. ■

Among a set of long time series, there can be many non-redundant interval skylines. Now, the problem is how to maintain those non-redundant interval skylines, which is the topic of the next subsection.

TABLE IV
AN EXAMPLE OF NON-REDUNDANT SKYLINES.

Interval	Non-redundant skyline
[2:2]	$\{s_2\}$
[3:3]	$\{s_3\}$
[4:4]	$\{s_1, s_2\}$
[2:3]	$\{s_5\}$
[3:4]	$\{s_4\}$
[2:4]	$\emptyset$

B. Maintaining Non-Redundant Interval Skylines

We divide all $\frac{w(w-1)}{2}$ intervals of the base interval W into w exclusive partitions. The m-th partition consists of $w-m+1$ intervals whose left-end timestamps are $t_c - w + m$. That is, the m-th partition includes intervals $[t_c - w + m : t_c - w + m], \ldots, [t_c - w + m : t_c]$ $(1 \leq m \leq w)$.

When a new timestamp $t_c + 1$ arrives, the base interval becomes $W' = [t_c - w + 2 : t_c + 1]$. We classify the updates of non-redundant interval skylines into three cases.

- **Case 1:** all intervals in the first partition are discarded, since $t_c - w + 1$ expires.
- **Case 2:** all existing non-redundant interval skylines in the intervals in the m-th partition $(2 \leq m \leq w)$ remain unchanged.
- **Case 3:** one new interval $[t_c - w + m : t_c + 1]$ is added to each partition except for the first one due to the new timestamp.

In Case 3, we need to compute the non-redundant skyline in the new intervals. Here, we employ a shared divide-and-conquer algorithm [8] (SDC for short). However, with very high dimensionality (e.g. $w = 1,000$) and large data set (e.g., $100,000$ time series), SDC is inefficient. Moreover, we have to remove redundant time series to obtain the non-redundant interval skylines. We develop a three-step procedure to improve SDC.

- **Step 1:** We compute the skyline over the new base interval $W' = [t_c - w + 2 : t_c + 1]$ and find possible false negatives if there is any.
- **Step 2:** We apply the shared divide-and-conquer algorithm to compute the interval skylines in all new intervals with the result of Step 1 as input, which is often much smaller than the whole set of time series.
- **Step 3:** We convert interval skylines into non-redundant interval skylines.

We explain each step in a subsequent section in detail.

1) Step 1: We can use the on-the-fly algorithm to obtain the interval skyline in the new base interval W'. However, instead of computing it from scratch, we compute it from the interval skyline in the previous base interval W. Intuitively, if a time series is in the interval skyline in W, it is very likely to be in the interval skyline in W' because only one attribute changes. In fact, if s is in the interval skyline in W, we can quickly eliminate time series that cannot dominate s in W' by checking the values at the expiring timestamp $t_c - w + 1$ and the new timestamp $t_c + 1$ using the following rule.

Lemma 3 (Interval skyline update): Given two time series s and q such that s is in the skyline of the previous base interval $W = [t_c - w + 1 : t_c]$. If $q[t_c + 1] < s[t_c + 1]$ or $q[t_c - w + 1] > s[t_c - w + 1]$, then q cannot dominate s in the new base interval $W' = [t_c - w + 2 : t_c + 1]$.

Proof: If $q[t_c+1] < s[t_c+1]$, obviously, by the definition of the dominance relation, q does not dominate s in W'.

If $q[t_c - w + 1] > s[t_c - w + 1]$, suppose q dominate s in W', then $q[k] > s[k]$ for $\forall k \in [t_c - w + 2 : t_c]$. Therefore, q dominates s in W, contradicting that s is in the interval skyline in W. ■

If s is in the interval skyline in W, when determining the skyline membership of s in W', by Lemma 3, we do not compare s with time series q if $q[t_c + 1] < s[t_c + 1]$ or $q[t_c - w + 1] > s[t_c - w + 1]$. Suppose the probability that $q[t_c + 1] < s[t_c + 1]$ is p, and the values at different timestamps are independent, then the probability that $q[t_c - w + 1] > s[t_c - w + 1]$ is $1-p$. Therefore, $1-p(1-p) \times 100\%$ time series do not need to compare with s. Apparently, $1-p(1-p) \geq 0.75$. In order words, Lemma 3 provides a good pruning power.

Given an interval $[i : t_c + 1] \subset W'$, is a time series in $Sky[i : t_c + 1]$ also in $Sky[W']$?

Lemma 4 (False negatives): Given an interval $[i : t_c+1] \subset W' = [t_c-w+2 : t_c+1]$ and a time series $s \in Sky[i : t_c+1]$, $s \in W'$ if there does not exist another time series $q \in Sky[i : t_c + 1]$ such that s and q have the same value at timestamp $t_c + 1$.

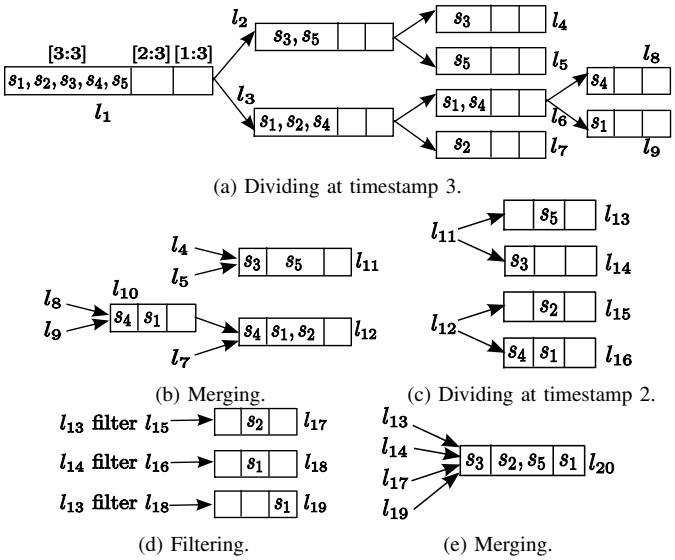
Proof: If $s \notin Sky[W']$, there exists another time series q in the interval skyline in W' such that $q \succ_{W'} s$. Then, $\forall k \in W', q[k] \geq s[k]$. If $q[t_c+1] > s[t_c+1]$, then $q \succ_{[i:t_c+1]} s$ since $[i : t_c + 1] \subset W'$, contradicting that s is in the interval skyline in $[i : t_c + 1]$. Thus, $q[t_c + 1] = s[t_c + 1]$. ■

Example 7 (False negatives): In the example shown in Table II, the interval skyline of interval $[3 : 5]$ consists of s_1, s_3, s_4, and s_5. s_2 is not in this interval skyline since s_1 dominates s_2 in $[3 : 5]$. However, s_2 is in the skyline of interval $[5 : 5] \subset [3 : 5]$. So s_2 is a false negative when we perform the update. Clearly, s_2 has the same value (i.e., 5) as s_1 at timestamp 5. ■

By Lemma 4, to compute the interval skylines in intervals $[t_c - w + 2 : t_c + 1], [t_c - w + 3 : t_c + 1], \ldots, [t_c + 1 : t_c + 1]$, we only need to consider the interval skyline in W' and the time series with the same value as those skyline time series in W' at the new timestamp. Therefore, the input for SDC in step 2 is significantly reduced.

2) Step 2: Now, we present the shared divide-and-conquer algorithm (SDC for short) that can share the computation when we compute the interval skylines for multiple new intervals $[t_c - w + 2 : t_c + 1], \ldots, [t_c + 1 : t_c + 1]$.

The shared divide-and-conquer [8] is the extension of the divide-and-conquer algorithm (DC for short) for computing the skyline in a single space. In our problem of interval skyline, a space is defined as a time interval. Both DC and SDC process in two steps – dividing and merging. In SDC, the computation of the two steps for multiple related spaces is shared. Therefore, it is much more efficient than computing the skyline in each space separately.



(a) Dividing at timestamp 3.

(b) Merging.

(c) Dividing at timestamp 2.

(d) Filtering.

(e) Merging.

Fig. 3. An example of SDC.

The related spaces (intervals) have to be organized as a path. For example, a set of intervals $[j : j], [j - 1 : j], \ldots, [i : j]$ $(i < j)$ form an interval path where each interval contains all preceding intervals. A *skylist* data structure is developed to store the intermediate results of the skylines over a path of intervals during the computation. The skylist for the above example has $j - i + 1$ elements where the k-th element $(0 \leq k \leq j - i)$ stores time series that are in the interval skyline in interval $[j-k : j]$ and do not appear in the preceding elements.

Let us describe SDC through an example.

Example 8 (SDC): Table II lists 5 time series. We are going to compute the skylines in intervals $[3 : 3]$, $[2 : 3]$, and $[1 : 3]$. Figure 3 shows the procedure of SDC. At the beginning, we put all 5 time series in the first element of skylist l_1 corresponding to interval $[3 : 3]$.

Next, SDC recursively divides l_1 on timestamp 3 (Figure 3(a)). We have 5 skylists that contain only one time series thus cannot be further partitioned. Then we go to the merging phase (Figure 3(b)). It is trivial to merge l_4 and l_5 since only one comparison is required. In this iteration, s_3 dominates s_5 in $[3 : 3]$, therefore s_5 cannot be in the skyline of $[3 : 3]$. We push s_5 into the next element $[2 : 3]$. Similarly, l_8 and l_9 are merged into l_{10} that is further merged with l_7 into l_{12}, and s_1 and s_2 are pushed into the second element.

Then, we divide the previous skylists on timestamp 2 (Figure 3(c)). Figure 4 shows the projection of the 5 time series on timestamps 2 and 3 and the corresponding skylists after the partitioning. Clearly, we do not need to compare the time series in l_{14} and the time series in l_{15}. We only need to compare l_{13} and $_{15}$, l_{14} and l_{16}, l_{13} and l_{16}, respectively. This

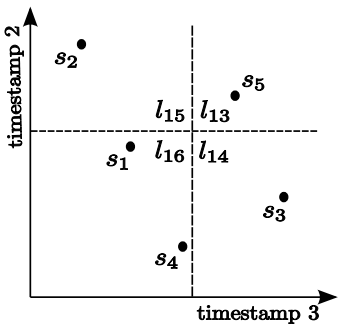


Fig. 4. The projection after the dividing of Figure 3(c).

is how DC and SDC avoid redundant comparisons.

Figure 3(d) shows that we use l_{13} to filter l_{15}. That is, each time series in l_{15} is compared with all time series in l_{13}. s_5 does not dominate s_2 over $[2:3]$, so s_2 stays in the second element in l_{17}. When filtering l_{16}, s_3 dominates s_4 over $[1:3]$, s_4 is removed from the skylist l_{18}. Then l_{18} is filtered by l_{13}.

Finally, in Figure 3(e) we merge l_{13}, l_{14}, l_{17} and l_{19} to get the final result l_{20}. Thus, $Sky[1:3] = \{s_1, s_2, s_3, s_5\}$, $Sky[2:3] = \{s_2, s_3, s_5\}$ and $Sky[3:3] = \{s_3\}$. ■

By the definition of skylist, a time series can only appear in one element. However, an element of the final skylist is not the non-redundant interval skyline of the corresponding time interval. That is, a skylist still may contain redundancy. For example, s_2 is in the second element of the final skylist, namely, s_2 is in the interval skyline in $[2:3]$. However, we can easily see that s_2 is in the interval skyline of $[2:2]$, $s_2[2] = 5$ which is the largest value at timestamp 2 among the 5 time series. Thereby, s_2 is not in the non-redundant interval skyline in $[2:3]$. We develop a method to remove the redundant time series in Step 3.

3) Step 3: Given the interval skyline (or the corresponding element of a skylist) in an interval $[k : t_c + 1]$, we have to remove "redundant time series" $s \in Sky[k : t_c + 1]$ such that $s \in Sky[i : j]$ and $[i : j] \subset [k : t_c + 1]$. This can be done in $O(1)$ time for each time series by comparing $[k : t_c + 1]$ with the *most recent minimal skyline interval* of s.

Recall that an interval is a minimal skyline interval of s if s is in the non-redundant interval skyline in this interval. The most recent minimal skyline interval of s is the minimal skyline interval of s whose left-end timestamp is the newest (i.e., the largest) among all minimal skyline intervals of s. Formally, $[i : j]$ is the most recent minimal skyline interval of s if $s \in NRSky[i, j]$ and there exists no interval $[i' : j']$ such that $s \in NRSky[i' : j']$ and $i' > i$.

Example 9 (The most recent minimal skyline interval): Suppose the base interval is $W = [2 : 4]$. From Table IV, we see that time series s_2 has two minimal skyline intervals, $[2 : 2]$ and $[4 : 4]$. So $[4 : 4]$ is the most recent minimal skyline interval of s_2. ■

Whether a time series is redundant in a new interval $[k : t_c + 1]$ can be determined using the following rule.

Lemma 5 (Non-reduandant interval skyline): For a given time series s in the skyline in a new interval $[k : t_c + 1]$, let $[i : j]$ be the most recent minimal skyline interval of s. $s \in NRSky[k : t_c + 1]$ if and only if $i < k$.

Proof: (Direction only-if) If $s \in NRSky[k : t_c + 1]$, since $t_c + 1$ is the newest timestamp, $t_c + 1 > j$. If $i \geq k$, then $[i : j] \subset [k : t_c + 1]$, resulting in a contradiction.

(Direction if) If $i < k$, suppose $s \notin NRSky[k : t_c+1]$, then there exists an interval $[i' : j']$ such that $s \in NRSky[i' : j']$ and $[i' : j'] \subset [k : t_c + 1]$. Thus, $i' \geq k > i$ which contradicts that $[i : j]$ is the most recent minimal skyline interval. ■

C. Complexity Analysis

We show before that the space complexity of the view-materialization method is $O(nw)$, linear with respect to the total number of data elements. The update operation is efficient because Step 1 significantly reduces the number of time series involved in the shared divide-and-conquer algorithm. The query time is proportional to the number of time series in the solution plus a small overhead for false positive detection.

V. Related Work

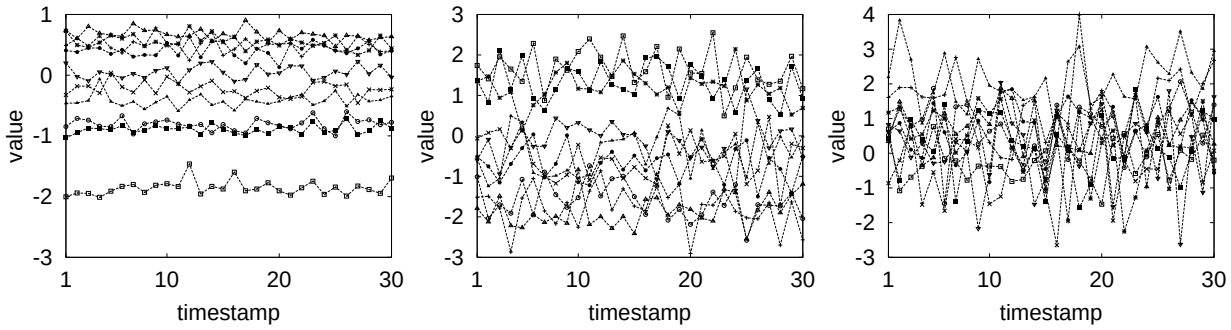
In computational geometry, Kung *et al.* [9] first investigate skylines referred as maxima. Börzsönyi *et al.* [1] introduce the concept of skylines in the context of databases and propose a SQL syntax for skyline queries. They also develop the block-nested-loop and the divide-and-conquer algorithms. Chomicki *et al.* [10] propose the sort-filter-skyline algorithm (SFS for short) to take the advantages of pre-sorting. The SFS algorithm is further improved by Godfrey *et al.* [11]. Tan *et al.* [12] develop a progressive skyline algorithm. Kossmann *et al.* [13] present an algorithm based on the nearest neighbor search. Papadias *et al.* [14] propose a branch-and-bound algorithm using the R-tree index.

Recently, skyline computation has been extended from full space to multiple subspaces [15], [8], [16], [7], [17]. Tao *et al.* [18] propose the SUBSKY method to index multidimensional data points using a B^+-tree for subspace skyline queries.

In the context of data stream processing, Lin *et al.* [2], Tao *et al.* [3], and Morse *et al.* [19], [20] consider the problem of continuously processing skyline queries against a data stream of multidimensional data points.

Most of the previous studies focus on data sets of low dimensionality, and assume that the set of attributes (dimensions) are given and do not change over time. Moreover, they only consider updates at the object level, that is, updating the skyline when some points are inserted or deleted. However, in our problem, the dimensionality is extremely high (e.g., an interval of hundreds of timestamps), and an attribute is generated at each new timestamp, resulting in the updates of all time series. Thus, time series data raises several new challenges which cannot be addressed efficiently by the previous work.

Skyline analysis is also extended in various aspects, including computing skylines in a distributed environment [21],



(a) $\sigma = 0.1$. (b) $\sigma = 0.5$. (c) $\sigma = 0.9$.

Fig. 5. Synthetic data sets with different standard deviation σ.

[22] and for partially-ordered domains [23], approximate skyline computation [24], high dimensional skylines [25], [26], handling and learning flexible user preferences in skyline queries [27], [28], [29], and materializing dominance relationships [30].

Last, in [31], a group of time series are approximated and represented by a "skyline bounding region", which is a region surrounded by a "top skyline" and a "bottom skyline". Here, the top skyline is a synthesized time series whose value at each timestamp is the maximum value of all time series in the group. The bottom skyline takes the minimum values at time stamps. [31] shows that using the skyline bounding regions in a skyline index can facilitate similarity search substantially. Clearly, the notion of "skyline" in [31] is completely different from our problem definition. Consequently, the methods are completely different.

VI. Empirical Study

We conduct extensive experiments to study the query cost and the update cost of the on-the-fly algorithm and the view-materialization algorithm, using both synthetic data sets and a stock data set. All algorithms are implemented in C++ and compiled by GCC. We conduct experiments on an Intel Core 2 Duo 2.2GHz PC with 1GB memory running Ubuntu Linux 7.04 operating system.

A. Synthetic Data Sets

A synthetic data set consists of n time series. We first generate the means μ_i $(1 \leq i \leq n)$ of all n time series using a standard normal distribution $\mathcal{N}(0, 1)$. Then each time series s_i follows a normal distribution $\mathcal{N}(\mu_i, \sigma)$ with the mean μ_i and standard deviation σ. In our experiments, we vary the number of time series n from $20,000$ to $100,000$ and the standard deviation σ from 0.1 to 0.9. By default, $n = 60,000$ and $\sigma = 0.5$. The size w of the base interval is between 100 and 500, we use 300 as the default value. To study the query efficiency, we issue interval skyline queries with interval size m from 50 to 250, and set 150 as the default value. Table V summaries the above experiment parameters, the default values are shown in bold font. In the following report, we run every experiment 100 times on different data sets with the same setting and report the average result.

TABLE V
A summary of experiment parameters

Parameter	Values
n	$20k, 40k, \mathbf{60k}, 80k, 100k$
σ	$0.1, 0.3, \mathbf{0.5}, 0.7, 0.9$
w	$100, 200, \mathbf{300}, 400, 500$
m	$50, 100, \mathbf{150}, 200, 250$

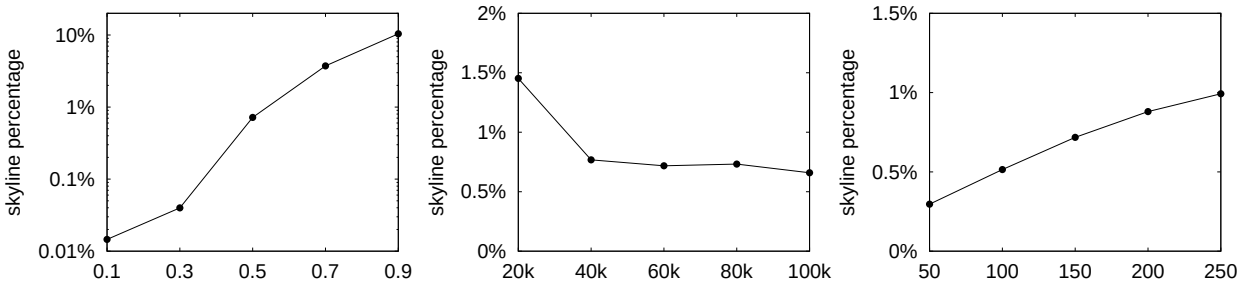
1) Data Sets Properties: We first study the properties of the synthetic data sets in order to understand the difficulty of interval queries on time series. Figures 5(a), (b), and (c) plot three data sets with the standard deviation σ of time series 0.1, 0.5, and 0.9, respectively. Each data set has 10 time series of 30 timestamps.

[1] proposes three benchmark distributions for skyline computation – correlated, independent, and anti-correlated. When σ is small ($\sigma = 0.1$ in Figure 5(a)), the values of a time series are stable. The data set can be considered as a correlated data set; while σ is large (Figure 5(c)), the values of a time series fluctuate dramatically. So the data set is similar to an anti-correlated data set.

Figure 6 shows that the percentage of the number of time series in the interval skyline with respect to various parameters. In Figure 6(a), the percentage of skyline increases exponentially when the standard deviation σ increases. This is similar to the case of traditional skyline study where the number of skyline points is small in correlated data sets while much larger in anti-correlated data sets.

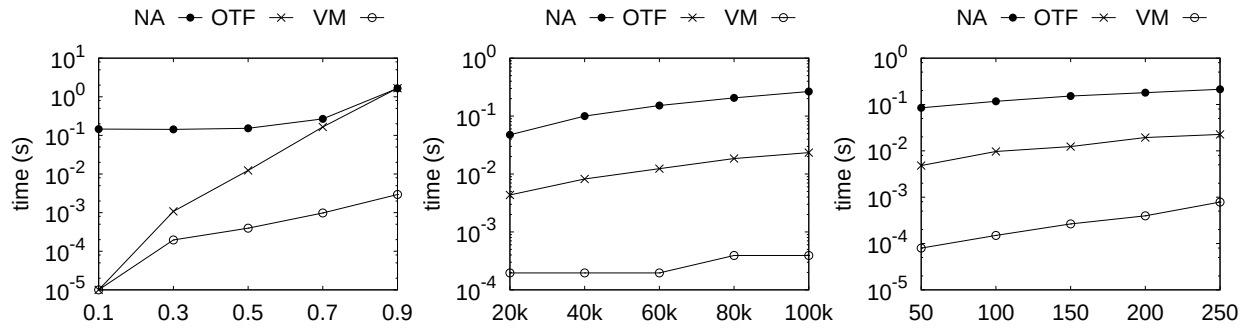
Figure 6(b) shows that, when the cardinality of the data set increases from $20,000$ to $100,000$, the skyline percentage decreases. Although the absolute number of the time series in the skyline increases, the probability of a time series being in the skyline decreases, since it is easier to be dominated by other time series in a larger data set.

Figure 6(c) shows that the skyline percentage increases linearly with respect to the query interval size. The larger the query interval, the more chances that a time series is not dominated in the interval.



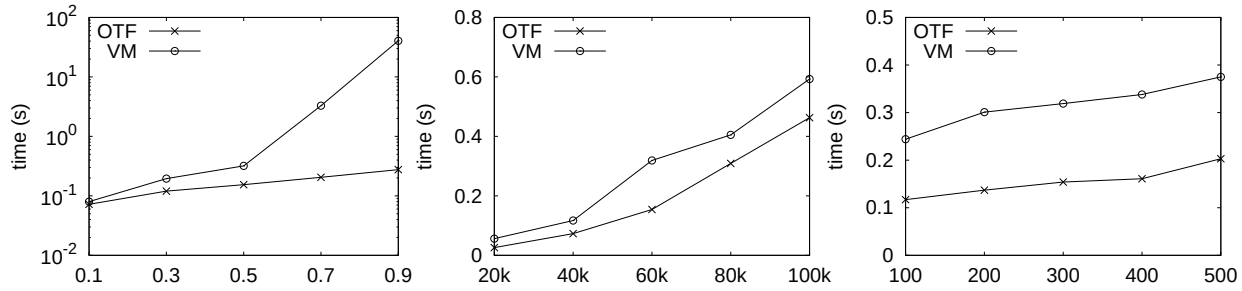
(a) Effect of the standard deviation σ. (b) Effect of the number of time series n. (c) Effect of the interval size m.

Fig. 6. The number of time series in interval skylines.



(a) Effect of the standard deviation σ. (b) Effect of the number of time series n. (c) Effect of the interval size m.

Fig. 7. The query time on synthetic data sets.



(a) Effect of the standard deviation σ. (b) Effect of the number of time series n. (c) Effect of the base interval size w.

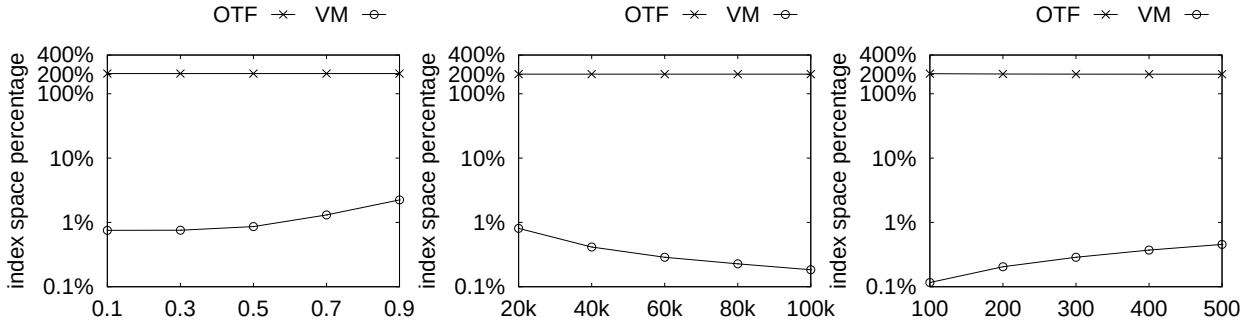
Fig. 8. The update time on synthetic data sets.

2) Query Efficiency: We compare the query efficiency of the on-the-fly algorithm (OTF), the view-materialization algorithm (VM), and a naive algorithm (NA) that computes the interval skyline using the Sort-filter-skyline algorithm (SFS) [10]. By default, we use data sets with $n = 60,000$ time series and $\sigma = 0.5$. The default size of the query interval is $m = 150$.

Figure 7 plots the running time in logarithm scale of the three algorithms in different experiment settings. It is clear that VM is the fastest among the three algorithms in every setting. VM can answer more than $1,000$ queries in 1 second. And it is always thousands of times faster than NA. OTF is also much faster than NA in most cases.

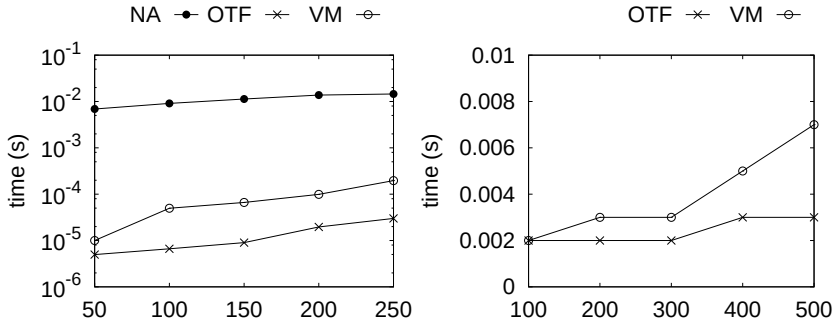
Figure 7(a) shows the trend of the query time with respect to the variation of time series. OTF is very sensitive to the variation. It is as fast as VM on data sets with low variations ($\sigma = 0.1$); while it is as slow as NA on data sets with $\sigma = 0.9$. Figure 7(b) shows the effect of the number of time series on the query time. The query time of all three algorithms increases approximately linearly. In Figure 7(c), we also see that the query time goes up slightly when the size of the query interval increases from 50 to 250.

3) Update Efficiency: Figure 8 studies the update efficiency of OTF and VM. Generally, VM has greater update cost than OTF. Figure 8(a) is in logarithm scale. It shows that the update time of OTF increases slowly when σ increases, since the change of the variation of time series does not affect the update of the radix priority search trees or the sketches of maximum values. Only the cost of the insertion sort increases because when σ is large, the ordered list may vary a lot after an update. However, the update cost of VM raises dramatically. For a



(a) Effect of the standard deviation σ. (b) Effect of the number of time series n. (c) Effect of the base interval size w.

Fig. 9. The space usage of data structures.



(a) The query time w.r.t. the interval size m. (b) The update time w.r.t. the base interval size w.

Fig. 10. Experiments on stock data sets.

data set with large σ, the number of time series in the skyline is large. Therefore, the input of the SDC algorithm is large, resulting in the performance decrease of VM update.

Figure 8(b) and (c) show that the update cost of both algorithms increases linearly with respect to the number of time series in the data set and the size of the base interval, respectively.

4) Space Cost: In this subsection, we study the space usage of the data structures used in OTF and VM. Let r be the space to store one data element in one time series at one timestamp. Then a data set with n time series of w timestamps occupies nwr space. Figure 9 plots the ratio in percentage of the space used by an index against the size of the raw data set. We notice that the space used by OTF is always about 2 times of the size of the data set, and it is not sensitive to the change of the standard deviation, the number of time series, or the base interval size. Because for one time series, a radix priority tree needs $O(2w)$ space to maintain the links of the tree structure, and the space of the sketches is $O(\log w)$ in expectation which is relatively small.

Surprisingly, VM only uses less than 3% space because it only stores the non-redundant interval skylines. But the space usage of VM varies on different data sets since it depends on the number of time series in interval skylines. Figure 9(a) shows that the percentage of the space used to store non-redundant interval skylines increases exponentially as the variance of time series increases. In Figure 9(b), the percentage decreases when the number of time series raises, though the absolute space used by VM increases. This is similar to the trend of the number of time series in interval skylines described in Figure 6(b). Figure 9(c) shows that the percentage increases when the size of the base interval increases.

B. Stock Data Sets

We use the stock data from the Center for Research in Security Prices (`http://www.crsp.com/`) which provides historical data of stocks listed on the NYSE, AMEX, NASDAQ, and ARCA exchanges. Our data set consists of 4720 stocks. Each stock is a time series of the daily dollar volume between 2003 and 2007. The daily dollar volume is approximated by the product of the closing price and the daily volume. We say a stock s_1 dominates another stock s_2 in a time interval, if the dollar volume of s_1 is not less than that of s_2 on every day, and s_1 has strictly greater dollar volume than s_2 on one day in this time interval. A stock is in the interval skyline if it is not dominated by any other stock in this interval. It is reasonable to say that a stock is active during a period if it is in the interval skyline of this period.

Figure 10(a) shows that VM and OTF both outperform the naive algorithm by 3 orders of magnitude for query answering. However, different to the results on synthetic data sets, OTF has better query performance than VM. The reason is that there are less than 10 stocks in the interval skylines. OTF can benefit

from the early termination strategy in Algorithm 1; while VM still has to scan all sub-intervals of the query interval to retrieve the interval skyline.

Figure 10(b) shows that both VM and OTF are efficient for update on the stock data set. They can handle hundreds of updates in 1 second.

In summary, the on-the-fly algorithm and the view-materialization algorithm are efficient for query answering and incremental updates on both synthetic data sets and the stock data set. In general, the view-materialization algorithm has better query performance while the on-the-fly algorithm has smaller update cost.

VII. Conclusions

In this paper, we tackle the problem of computing skylines on time series data. We advocate interval skyline queries, a novel type of time series analysis queries. We propose two interesting methods: the on-the-fly method and the view-materialization method. Both methods use linear space. An extensive experimental study shows that both methods are efficient for query answering and increment maintenance.

As future work, it is interesting and challenging to tackle the problem of interval skyline queries on streaming time series. Moreover, it is interesting to consider summarization of skylines on time series.

Acknowledgement

This work was supported in part by an NSERC Discovery Grant, an NSERC Discovery Accelerator Supplements Grant, and a grant from the Simon Fraser University Community Trust Endowment Fund. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, Heidelberg, Germany, April 2001.

[2] X. Lin, Y. Yuan, W. Wang, and H. Lu, "Stabbing the sky: Efficient skyline computation over sliding windows," in *Proc. 2005 Int. Conf. Data Engineering (ICDE'05)*, Tokyo, Japan, April 2005.

[3] Y. Tao and D. Papadias, "Maintaining sliding window skylines on data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 2, pp. 377–391, 2006.

[4] E. M. McCreight, "Priority search trees," *SIAM J. Comput.*, vol. 14, no. 2, pp. 257–276, 1985.

[5] M. Garofalakis, J. Gehrke, and R. Rastogi, *Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[6] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson, "On the average number of maxima in a set of vectors and applications," *J. ACM*, vol. 25, no. 4, pp. 536–543, 1978.

[7] T. Xia and D. Zhang, "Refreshing the sky: the compressed skycube with efficient support for frequent updates," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data (SIGMOD'06)*. New York, NY, USA: ACM Press, 2006, pp. 491–502.

[8] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang, "Efficient computation of the skyline cube," in *Proceedings of the 31th International Conference on Very Large Data Bases (VLDB'05)*, Trondheim, Norway, August 2005.

[9] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *J. ACM*, vol. 22, no. 4, pp. 469–476, 1975.

[10] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. 2003 Int. Conf. Data Engineering (ICDE'03)*, Bangalore, India, March 2003, p. 717.

[11] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proceedings of the 31th International Conference on Very Large Data Bases (VLDB'05)*, Trondheim, Norway, August 2005.

[12] K. Tan, P. Eng, and B. Ooi, "Efficient progressive skyline computation," in *Proc. 2001 Int. Conf. on Very Large Data Bases (VLDB'01)*, 2001. [Online]. Available: citeseer.nj.nec.com/455285.html

[13] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries," in *Proc. 2002 Int. Conf. on Very Large Data Bases (VLDB'02)*, Hong Kong, China, Aug. 2002.

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proc. 2003 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'03)*, San Diego, California, June 2003.

[15] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the best views in skyline: A semantic approach," in *Proceedings of the 31th International Conference on Very Large Data Bases (VLDB'05)*, Trondheim, Norway, August 2005.

[16] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang, "Towards multidimensional subspace skyline analysis," *ACM Trans. Database Syst.*, vol. 31, no. 4, pp. 1335–1381, 2006.

[17] J. Pei, A. W. C. Fu, X. Lin, and H. Wang, "Computing compressed skyline cubes efficiently," in *Proceedings of the 23nd International Conference on Data Engineering (ICDE'07)*. Istanbul, Turkey: IEEE, April 2007.

[18] Y. Tao, X. Xiao, and J. Pei, "Subsky: Efficient computation of skylines in subspaces," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*. Atlanta, GA, USA: IEEE, April 2006.

[19] M. D. Morse, J. M. Patel, and W. I. Grosky, "Efficient continuous skyline computation," in *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, 2006, p. 108.

[20] M. D. Morse, J. M. Patel, and W. Grosky, "Efficient continuous skyline computation," *Inf. Sci.*, vol. 177, no. 17, pp. 3411–3437, 2007.

[21] W.-T. Balke, U. Güntzer, and J. X. Zheng, "Efficient distributed skylining for web information systems." in *EDBT*, 2004, pp. 256–273.

[22] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi, "Skyline queries against mobile lightweight devices in manets," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*. Atlanta, GA, USA: IEEE, April 2006.

[23] C. Y. Chan, P.-K. Eng, and K.-L. Tan, "Stratified computation of skylines with partially-ordered domains," in *SIGMOD Conference*, 2005, pp. 203–214.

[24] V. Koltun and C. H. Papadimitriou, "Approximately dominating representatives," vol. 371, no. 3. Essex, UK: Elsevier Science Publishers Ltd., 2007, pp. 148–154.

[25] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "On high dimensional skylines," in *EDBT*, 2006, pp. 478–495.

[26] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *SIGMOD Conference*, 2006, pp. 503–514.

[27] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang, "Mining favorable facets," in *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2007, pp. 804–813.

[28] B. Jiang, J. Pei, X. Lin, D. W.-L. Cheung, and J. Han, "Mining preferences from superior and inferior examples," in *KDD '08: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2008.

[29] R. C.-W. Wong, A. W.-C. Fu, J. Pei, Y. S. Ho, T. Wong, and Y. Liu, "Efficient skyline querying with variable user preferences on nominal attributes," in *VLDB '08: Proceedings of the 34th International Conference on Very Large Databases*. New York, NY, USA: ACM, 2008.

[30] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang, "Dada: a data cube for dominant relationship analysis," in *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2006, pp. 659–670.

[31] Q. Li, I. F. V. López, and B. Moon, "Skyline index for time series data," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 6, pp. 669–684, 2004.