データ更新が頻繁な環境におけるスカイラインデータ計算アルゴリズムの提案

Kamalas Udomlamlert[1,a)] Takahiro Hara[1,b)] Shojiro Nishio[1,c)]

Abstract: 2 つのデータが与えられたとき，すべての属性値が劣っておらず，かつ少なくとも一つの属性値が優っている場合，一方のデータはもう一方のデータを支配しているという．スカイラインクエリは，どのデータにも支配されていないデータのみを検索する．ここで，データ更新が頻繁に起こる環境について考える．この時、長期間スカイラインであるデータは多くのアプリケーションで用いられる場合が多い．このようなデータを検索するために，更新が発生するごとにすべてのデータを候補として再計算する単純なアプローチが考えられるが，計算量が非常に大きいという問題がある．そこで，本稿ではこの問題を解決する効率的なアルゴリズムを提案する．提案アルゴリズムでは，Minimum bounding rectangles (MBRs) を用いて，スカイラインとならないデータを候補から除外し，計算時間を短縮する．シミュレーション実験の結果より，提案アルゴリズムは単純なアプローチよりも高速にスカイラインデータを検索できることを確認した．

1. Introduction

Recently, many query methods have been developed and gained a lot of attentions in database researches in order to deliver most satisfactory results to various classes of end-users. Considering the dominance relations among objects (the competitiveness of each object), skyline computation [1], which represents a result set which each result item is not worse than others, is also one of popular query methods so far. For a common example, given that a hotel is represented by 2 attributes including price per night and rating, the system has to give a set of preferable hotels to end-users. However, each user may have different criteria to evaluate the same set of these hotels, for example, the first user may want a hotel which is high-rated regardless of price while the second user may prefer to standard rating hotels with reasonable price. Obviously, hotels which have higher price per night and lower rating than others cannot attract any users, and they should be removed from the final result set. A skyline set is useful here because it returns hotel candidates that each result will not contain such non-preferable choices.

In some applications, multiple data objects may change their data values according to the time, e.g., financial data where stock prices and fundamental criteria of many stocks can be dynamically changed every time-tick. Another example application is to analyzing historical data archives, for example sport data where player statistics are changed partially in each match. Even though, there are many continuous skyline processing methods previously proposed in the research community, there are still a couple of differences as follows:

(1) Continuous skyline processing, e.g., [5], [11], [17], mainly aims to deal with a single data update at each time while in our assumption, a bulk of data updates is raised at a time.

(2) Continuous skyline processing tries to find how to fast answer a new skyline set upon a single update regardless of maintenance time (only query response time) [5] while in our assumption, we also analyze the historical archive of data, so overall processing time including skyline computation time as well as maintenance time are both also taken into accounts.

In skyline monitoring, a single data update can totally change a final skyline set, so handling multiple data updates at a time is challenging. Without any technique, to guarantee the correctness, the new skyline set must be computed from the entire set of data objects in each time snapshot.

In this paper, we propose an efficient method based on the properties of a bounding box (a minimum bounding rectangle in the case of 2 dimensions). We use bounding boxes to capture and prune unnecessary data candidates as well as neglect no-effect data updates. Therefore, we can identify a smaller candidate set in skyline computation in consecutive data snapshots resulting in saving overall execution time.

In summary, the contributions of this paper are as follow:

- We formulate the problem definition of skyline computation on a bulk of data updates as well as illustrate example applications of this problem.
- We propose an efficient algorithm and index structures to identify a smaller set of data candidates before skyline calculation, and the cost of maintenance is in according with

[1] Graduate School of Information Science and Technology, Osaka University, Japan
[a)] kamalas.u@ist.osaka-u.ac.jp
[b)] hara@ist.osaka-u.ac.jp
[c)] nishio@ist.osaka-u.ac.jp

degree of data changes (pay as you go).

- We conduct some experiments in various settings by using both synthetic and real datasets to show that our proposed method can run faster than the baseline.

2. Related work

Skyline computation in database research was firstly introduced in [1]. The authors proposed two skyline algorithms including Block-Nested-Loop skyline and Divide-and-Conquer skyline algorithms. After that, numerous research papers tried to enhance the performance by using more complicated index structures such as Branch-and-Bound skyline algorithm [12].

Apart from the traditional skyline processing in databases, skyline processing for distributed systems has been studied. Many techniques were proposed as described in the survey [4]. Moreover, many interesting variants of skyline processing methods have also been studied for example, reverse skyline query [3], fragmented skyline [13], subspace skyline [16] and interval skyline [7].

On the other hand, finding skyline sets on time series is closely related to continuous skyline query processing which aims to monitor the latest skyline in response to new data updates over the course of time. There are many variants of continuous skyline processing methods. The papers in [2], [6], [8] assume a kinetic model of moving data objects aiming to find the skyline objects (static attributes) when some dynamic attributes (locations or query points) are movable. These are quite different from ours, and their techniques are not suitable to solve our context's problems because a kinetic model is not assumed.

Another group of related work is continuous skyline on data streams [9], [11], [15], [19]. These aim to efficiently monitor the latest skyline set in the sliding window where window-range, data arrival time and data expiration time are given. Nevertheless, a single data modification (data update) can be taken as two operations - insertion and deletion, but it does not work well in our assumed problem because of high maintenance cost.

Apart from skyline computation for data streams, [5] used the pre-computed second skyline to improve the query response time to answer the final skyline (the first skyline) on new data updates. However, the procedure to update focuses only one single update at a time, so the performance degrades in our assumption where the update ratio is high. [17] assumed a very close problem to ours according to its data model. This work tries to monitor the latest modification of the skyline set when each data object is updated by the information from update streams. Its main contribution relies on allocating data into grid cells and consider the dominance relations between those grids to prune unnecessary candidates from skyline calculation. Using grid indexes is very common for pruning unnecessary candidates as found in [15], [19].

3. Preliminaries

3.1 Data model

We assume that the system analyzes a skyline set over a fixed number of data objects. Each data object is comprised of m-numerical values as attributes and data id *id* as an object's identifier. Data attributes of data object i at the initialization (the first snapshot in the historical archive, snapshot 0) are represented as a tuple $p_i^0 = (p_i^0[1], p_i^0[2], \ldots, p_i^0[m])$. Let N be the number of all data objects in the system, a set of all data objects at snapshot t is denoted by $P^t = \{p_1^t, p_2^t, \ldots, p_N^t\}$, where $t = \{0, 1, \ldots, T\}$ and T is the total number of snapshots in the archive.

Definition 1. *(Point dominance) A data point p_i^t dominates p_j^t ($p_i^t \prec p_j^t$) if and only if $\forall k \in \{1, 2, \ldots, m\} : p_i^t[k] \leq p_j^t[k]$, and $\exists l \in \{1, 2, \ldots, m\} : p_i^t[l] < p_j^t[l]$.*

Definition 2. *(Weakly point dominance) A data point p_i^t weakly dominates p_j^t ($p_i^t \preceq p_j^t$) if and only if $\forall k \in \{1, 2, \ldots, m\} : p_i^t[k] \leq p_j^t[k]$.*

3.2 Data update model

In this research, we assume that at each timestamp (snapshot) t, only a partial set of data objects changes their attributes' values from the previous timestamp $t-1$. An update model like this can be often found in pull-based data delivery model that the server pulls new updates from data sources periodically.

How data change is described by an update tuple which can be defined in many ways based on the applications, for example, a new value update defined by a 3-tuple $u = (id, t, (p[1], p[2], \ldots, p[m]))$ that means $p_{id}^t = (p[1], p[2], \ldots, p[m])$ and a modification update defined by a 3-tuple $u = (id, t, (\Delta p[1], \Delta p[2], \ldots, \Delta p[m]))$ that means $p_{id}^t = (p_{id}^{t-1}[1] \otimes \Delta p[1], p_{id}^{t-1}[2] \otimes \Delta p[2], \ldots, p_{id}^{t-1}[m] \otimes \Delta p[m])$ where $\otimes$ is an operator, for example, addition, multiplication and average. This changes the corresponding data object p_{id}^{t-1} to p_{id}^t.

A list of updates of snapshot t (update streams) is a list of update tuples $U^t = \{u_1^t, u_2^t, \ldots\}$ where $|U| \leq N$. Therefore, the data objects which are not modified by any update tuples remain the same values that is $p_{id}^t = p_{id}^{t-1}$. Since our model embodies both multidimensional attributes (space) and time-series data (temporal data), it also works with spatio-temporal applications (e.g., location-aware services).

3.3 Skyline calculation

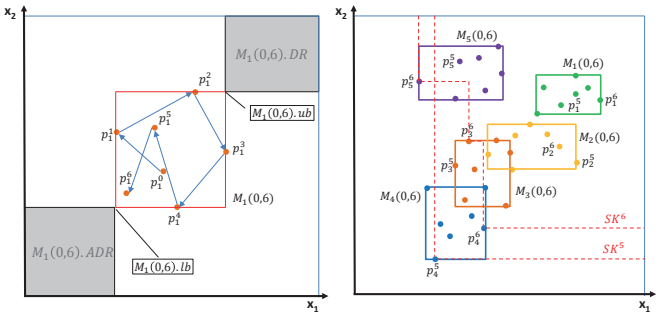
In this research, we aim to continuously calculate a set of skyline (SK^t) efficiently from P^t at each consecutive snapshot t, i.e., the next snapshot from $t-1$ ($\forall t \in \{0, 1, \ldots, T\}$).

Definition 3. *(Skyline set) Given a set of data points at snapshot t (P^t), $p_i^t \in P^t$ is included in the skyline set SK^t if and only if $\forall p_j^t \in (P^t \backslash \{p_i^t\})$, p_j^t does not dominate p_i^t ($p_j^t \not\prec p_i^t$).*

3.4 Summarizing consecutive data snapshots with minimum bounding rectangles (MBRs)

A minimum bounding rectangle (MBR) is the smallest oriented rectangle enclosing a set of points which is a 2 dimensional case of a minimum bounding box in a coordinate system. Our proposed solution can deal with any number of dimensions by using the same idea. According to all examples in this paper illustrated in a 2-dimensional space, for simplicity, we use the term *MBRs* to refer to this expression in general.

While each data object possibly changes its attributes' values at every timestamp t, those tracing data points can be seen as a set of points which can be summarized (represented) as a MBR.



(a) A example MBR $M_1(0,6)$ (b) An example of 2 skylines in different snapshots

Fig. 1: An example of MBRs and the space

Therefore, we use a MBR to summarize the space that a data object changes its values between consecutive snapshot a to b, i.e., $\{p_i^a, p_i^{a+1}, \ldots, p_i^b\}$ where $a \leq b$.

A MBR of data object id from consecutive snapshot a to b is represented by a 3-tuple $M_{id}(a,b) = (id, ub, lb)$ where $ub[l] = \max_{j\in[a,b]} p_{id}^j[l]$ and $lb[l] = \min_{j\in[a,b]} p_{id}^j[l]$ when $l \in \{1,2,\ldots,m\}$. We represent a list of MBRs which ends at snapshot t as $M^t = \{M_1(a_1,t), M_2(a_2,t), \ldots, M_N(a_N,t)\}$ where a_{id} is a number of the start snapshot of MBR id. Fig.1a exemplifies 7 consecutive data snapshots of a data object ($id = 1$) denoted by $\{p_1^0, p_1^1, \ldots, p_1^6\}$ while the arrows express their trajectories between two consecutive data snapshots. Their MBR is a box $M_1(0,6)$ shown in the figure. In this paper, for short, M_{id}^t refers to the latest MBR of object id at timestamp t regardless of the beginning timestamp ($M_{id}(*,t)$).

3.5 Dominance region and Anti-dominance region

A dominance region of a MBR ($M_i^t.DR$) is a subspace where $x[l] \geq M_i^t.ub[l]$ for all $l \in \{1,2,\ldots,m\}$. Any data points or MBRs that fully fall within this region will be weakly dominated by M_i^t. In the contrary, an anti-dominance region of MBR ($M_i^t.ADR$) is a subspace where $x[l] \leq M_i^t.lb[l]$ for all $l \in \{1,2,\ldots,m\}$. Any data points or MBRs that fully fall within this region weakly dominate M_i^t. To illustrate, in Fig.1a, the dominance region and anti-dominance region of $M_1(0,6)$ are the gray area in $M_1(0,6).DR$ and $M_1(0,6).ADR$ respectively.

Definition 4. *(MBR Dominance) A MBR M_i^t dominates M_j^t ($M_i^t \prec M_j^t$) if and only if $\forall l \in \{1,2,\ldots,m\} : M_i^t.ub[l] \leq M_j^t.lb[l]$, i.e., $M_i^t.ub \preceq M_j^t.lb$ (M_j^t fully falls in $M_i^t.DR$)*

Due to $M_{id}^t.lb[l] = \min_{j\in[a,b]} p_{id}^j[l]$ when $l \in \{1,2,\ldots,m\}$, we conclude that a point $M_{id}^t.lb$ weakly dominates every point p_{id}^k where $k = \{a, a+1, \ldots, b\}$. We further define a definition of a set of skyline MBRs at snapshot t (SKM^t).

Definition 5. *(Skyline MBR) Given a set of MBRs at snapshot t (M^t), $M_i^t \in M^t$ is included in the set of skyline MBRs SKM^t if and only if $\forall M_j^t \in (M^t \backslash \{M_i^t\})$, M_j^t does not dominate M_i^t.*

In addition, we denote a set of MBRs that do not belong to SKM^t as a set of non-skyline MBRs ($NSKM^t = M^t \backslash SKM^t$).

Lemma 1. *$\forall M_i^t \in NSKM^t$: there must be at least one MBR which is inside $M_i^t.ADR$.*

Proof. (Proof by contradiction) Assume that $M_i^t \in NSKM^t$, but there is no MBR inside $M_i^t.ADR$. Due to $M_i^t \in NSKM^t$ and Definition 5, there exists at least one MBR M_j^t dominating M_i^t. As a result, $M_j^t.ub[l] \leq M_i^t.lb[l]$ for $\forall l \in \{1,2,\ldots,m\}$. From the explanation in Section 3.5, this leads to the contradiction because M_j^t must be contained in $M_i^t.ADR$. □

3.6 Pruning candidates for skyline calculation using MBRs

At each snapshot, instead of finding a skyline set from all data points P^t, we find the skyline by using only candidates in the skyline MBRs ($\{p_i^t / M_i^t \in SKM^t\}$). The cardinality of SKM^t is likely to be much smaller than that of P^t, so skyline calculation can be computed faster.

Lemma 2. *Skyline calculation from $\{p_i^t / M_i^t \in SKM^t\}$ produces the correct skyline set (SK^t) as same as using all data points P^t.*

Proof. (Proof by contradiction) In order to produce incorrect SK^t, there must be at least a point p_i^t where $M_i^t \in NSKM^t \wedge p_i^t \in SK^t$. By Lemma 1, there exists $M_j^t : M_j^t \prec M_i^t$. This leads to contradiction that $p_i^k \notin SK^t$ because $\forall k : M_j^t.ub \preceq p_i^k$. □

Running Example in Fig.1b

Fig.1b illustrates series of 7 data snapshots (data points) of 5 data objects ($id = \{1,2,3,4,5\}$) with their MBRs ($M_{id}(0,6)$). By Definition 5, $M_1(0,6)$ and $M_2(0,6)$ are not skyline MBRs because they are dominated by $\{M_3(0,6), M_4(0,6)\}$ and $\{M_4(0,6)\}$ respectively. Therefore, d_1^t and d_2^t are guaranteed not to include the final skyline at $t \in \{0,1,\ldots,6\}$ that we can safely remove them from skyline calculation. As in the example, consider only $t = \{5,6\}$, while the final skyline at $t = 5$ (SK^5) includes only $\{p_4^5\}$, SK^6 includes $\{p_3^6, p_4^6, p_5^6\}$.

3.7 Changes of an MBR due to new updates

We start considering how a bulk of new data updates at the next snapshot (U^{t+1}) affects the current MBRs M_i^t. Certainly, including a new snapshot of data can make M_i^t changed in size as well as their properties, i.e., lb, ub, DR and ADR. Consider a MBR of data object id at snapshot t and its update tuple u_j^{t+1} where $u_j^{t+1}.id = id$, where $l \in \{1,2,\ldots,m\}$

$$M_i^{t+1}.ub[l] = \max(M_i^t.ub[l], p_i^{t+1}[l]) \quad (1a)$$

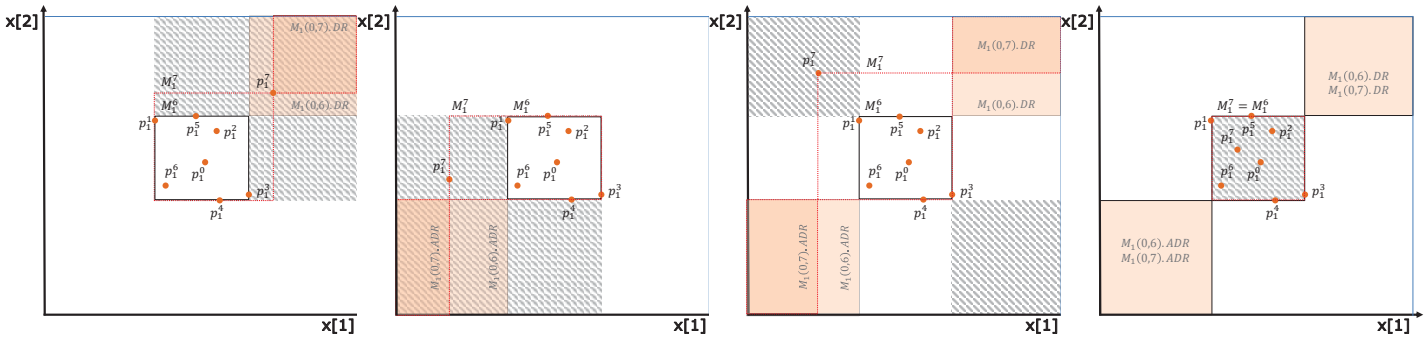
$$M_i^{t+1}.lb[l] = \min(M_i^t.lb[l], p_i^{t+1}[l]) \quad (1b)$$

The effects of the data updates to M_{id}^{t+1} can be classified into 4 cases as follows:

(Case I) $M_{id}^{t+1}.lb = M_{id}^t.lb$ and $M_{id}^{t+1}.ub \succ M_{id}^t.ub$
This case happens when p_{id}^{t+1} falls in gray-shaded area at the right-top corner illustrated in Fig.2a. $M_{id}^{t+1}.ADR$ remains the same as M_{id}^t, but $M_{id}^{t+1}.DR$ becomes smaller.

(Case II) $M_{id}^{t+1}.lb \prec M_{id}^t.lb$ and $M_{id}^{t+1}.ub = M_{id}^t.ub$
This case happens when p_{id}^{t+1} falls in the gray-shaded area at the left-bottom corner illustrated in Fig.2b. In addition, $M_{id}^{t+1}.ADR$ becomes smaller than that of M_{id}^t while $M_{id}^{t+1}.DR$ remains the same as M_{id}^t.

(Case III) $M_{id}^{t+1}.lb \prec M_{id}^t.lb$ and $M_{id}^{t+1}.ub \succ M_{id}^t.ub$
If p_{id}^{t+1} falls in the gray-shaded areas at the right-top and left-bottom corners illustrated in Fig.2c., this changes both lb



(a) Case I (b) Case II (c) Case III (d) Case IV

Fig. 2: MBR updates before including p_1^7 and after including p_1^7

Table 1: Notation Summary

Notation	Description
p_i^t	a data tuple of data object i at snapshot t
P^t	a set of data tuples at snapshot t
u_i^t	an i-th new update tuple at snapshot t
U^t	a set of new update tuples at snapshot t
$M_i(a,b)$	a MBR summarizing consecutive data snapshots between a and b of data object i
M_i^t, $M_i(*,t)$	refer to a current $M_i(a,b)$ where $b = t$ and a is neglected
SK^t	a set of skyline points at snapshot t
SKM^t	a list of skyline MBRs at snapshot t
$NSKM^t$	a list of non skyline MBRs at snapshot t

and ub. In the same way, both $M_{id}^{t+1}.DR$ and $M_{id}^{t+1}.ADR$ are degraded compared to that of M_{id}^t.

(Case IV) $M_{id}^{t+1}.lb = M_{id}^t.lb$ and $M_{id}^{t+1}.ub = M_{id}^t.ub$
M_{id}^{t+1} and M_{id}^t are identical if p_{id}^{t+1} falls inside M_{id}^t (gray-shaded area) illustrated in Fig.2d.

According to Definition 5, the membership of MBRs in SKM^t and $NSKM^t$ possibly no longer holds for snapshot $t+1$ due to the changes of its DR and ADR. Therefore, our proposed method introduces an efficient method to maintain the consistency in order to identify SKM^{t+1} as well as $NSKM^{t+1}$. We describe details in the next section. The notations and symbols used for the explanation are summarized in Table 1.

4. Overview

From the preliminaries in Section 3, we proved that only the data objects whose current MBR belongs to SKM^t are a sufficient candidate set for skyline calculation of each snapshot t. This can significantly reduce the cardinality of data candidates to be calculated in skyline computation. Therefore, we propose an efficient method to maintain those MBRs by keeping two different lists including SKM^t and $NSKM^t$ where $M^t = SKM^t \cup NSKM^t$ and $SKM^t \cap NSKM^t = \phi$.

Our proposed method can be roughly divided into 3 steps.

(1) **Pre-computation maintenance (PRE)**
According to new data updates from the stream at t, the MBRs at $t-1$ can be possibly changed in terms of physical MBRs and their pruning capability. This step tries to identify the correct SKM^t by paying low maintenance cost as much as necessary before skyline calculation.

(2) **Skyline calculation (SKY)**
This process is straight-forward. We calculate a final skyline result (SK^t) by one of many state-of-the-art skyline computation methods but using a smaller set of candidates, i.e., a set of data objects whose MBR belongs to SKM^t.

(3) **Post-computation maintenance (POST)**
Regarding to the final skyline result, we are able to detect some data objects whose MBR belongs to SKM^t but does not appear in SK^t. In other words, these MBRs produce unpleasant false positives degrading overall the pruning capability. In this process, we propose a heuristic rule to solve this problem.

5. Proposed algorithms

5.1 Intialization ($t = 0$)

At the initialization ($t = 0$), we have to construct the initial MBRs (M^0) of all data objects (P^0). That means, in each MBR M_i^0, $M_i^0.lb = M_i^0.ub = p_i^0$ for all $i \in \{1,2,\ldots,N\}$. Hence, we calculate the first skyline set of P^0 and classify MBRs into 2 lists as $SKM^0 = \{M_i^0 / p_i^0 \in SK^0\}$ and $NSKM^0 = M^0 \backslash SKM^0$.

Moreover, we additionally introduce two important elements to help easily identifying the relations between MBRs in SKM^t and $NSKM^t$ including an id list of MBRs in a dominance region of each MBR ($M_i^t.DRM$) and a single id of MBR which is in an anti-dominance region ($M_i^t.adrm$). These relations can be established while executing skyline calculation by using the following rules:

(1) If M_i^t is dominated by M_j^t on skyline computation, then $M_i^t \in NSKM^t$, $M_i^t.admr = j$ and $i \in M_j^t.DRM$.

(2) If M_i^t is not dominated by any other MBRs, then $M_i^t \in SKM^t$ and $M_i^t.admr = nil$ (not applicable).

According to Definition 4 and Lemma 1, we conclude that $\forall M_i^t \in NSKM^t : M_i^t \neq nil$ while $\forall M_i^t \in SKM^t : M_i^t = nil$, and as long as $M^t_{M_i^t.admr}$ dominates M_i^t, M_i^t must belong to $NSKM^t$.

5.2 On a bulk of new data updates at $t > 0$

As a set of bulk updates $U^t = \{u_1^t, u_2^t, \ldots\}$, each update tuple describes how a new data tuple p^t can be generated while the rest (not indicated in U^t) $p_i^t = p_i^{t-1}$. For those unchanged data tuples, their MBRs are also unchanged in the same way, so the system does nothing in this case.

However, we need to verify some affected MBRs both in SKM^{t-1} and $NSKM^{t-1}$ whether they are still in the correct lists at snapshot t due to new data updates. Therefore, we create an additional list called a verification list (VL) containing MBRs which are needed to be further investigated. There are only 2 cases that need to be checked. Otherwise can be neglected. This process is included in the pre-calculation maintenance.

(1) $M_i^{t-1} \in NSKM^{t-1}$ and M_i^t affected by p_i^t fall in either Case II or Case III (referred to Section 3.7).
Because $M_i^t.ADR$ is deteriorating, $M_{M_i^t.admr}^{t-1}$ may no longer dominate M_i^t. Therefore, we check if it still dominates, and if not, M_i^t is pushed to VL.

(2) $M_i^{t-1} \in SKM^{t-1}$ and M_i^t affected by p_i^t fall in either Case I or Case III (referred to Section 3.7).
Because the dominance capability of M_i^t ($M_i^t.DR$) has been reduced, some MBRs in $M_i^{t-1}.DRM$ may no longer be dominated by M_i^t. We move $j \in M_i^{t-1}.DRM$ which is not dominated by M_i^t to VL.

(3) Otherwise
No change of MBRs' status.

VL Refinement

We can see that all listed MBRs in VL used to be in $NSKM^{t-1}$, but possibly they are no longer able to be in $NSKM^t$. This will increase the number of MBRs in SKM^t resulting in increasing in the number of candidates in skyline calculation. At this process, we try to check these MBRs again whether there exists at least a MBR in SKM^t dominating them before including them into SKM^t. Therefore, for each $M_i^t \in VL$, we search for any first $M_j^t \in SKM^t$ that dominates M_i^t. If found, M_i^t is pushed back to $NSKM^t$ and set $M_i^t.admr = j$. Otherwise, it is swapped to SKM^t.

It is noted that there may be more than one M_j^t dominating M_i^t, but we simply choose the first found MBR that dominates M_i^t. In best practice, M_j^t to be chosen should be a MBR in SKM^t which gives the longest distance between $M_i^t.lb$ and $M_j^t.ub$ ($M_i^t.admr = \max_{\forall j: M_j^t \prec M_i^t} d(M_i^t.lb, M_j^t.ub)$). However, to do this approach consumes more time because we cannot avoid scanning the entire list of SKM^t while simply choosing the first found dominating MBR can perform early termination.

5.3 Skyline calculation

Because the main objective of this paper aims to reduce the number of candidates to be calculated in skyline computation regardless of skyline computation algorithms. Therefore, we simply adopt the state-of-the-art Block-Nested-Loop skyline computation as default. According to Lemma 2, we calculate the final skyline set at snapshot t (SK^t) by using a set of data points whose MBRs belong to SKM^t. Nevertheless, other complicated skyline computation algorithms can be used for further improvements, but this is out of the scope of this paper.

5.4 Post-compuation maintenance

After SK^t has been calculated, it is possible that some of candidates from SKM^t do not finally belong to SK^t (false positives). If $M_i(*,t) = M_i^t$ usually incurs a false positive for a long period (too many consecutive snapshots), it is worth considering paying maintenance cost to reconstruct and newly start a MBR from the current snapshot t ($M_i^t = M_i(t,t)$) because of the possible higher gains of pruning capability in the next iteration.

Lemma 3. *The dominance and anti-dominance regions of a newly-reconstructed MBR $M_i' = M_i(t,t)$ are not smaller than the old $M_i = M_i^t(a,t)$ where $a < t$.*

Proof. The dominance region of M_i ($M_i.DR$) is $x[l] \geq M_i.ub[l]$ for $l \in \{1,2,\ldots,m\}$. However, $M_i'.ub = p_i^t$ and $p_i^t[l] \leq \max_{k\in[a,t]} p_i^k[l] = M'.ub$. Hence, $M_i'.DR$ must not be smaller than that of M_i. In the same way, the anti-dominance region of M_i ($M_i.ADR$) is $x[l] \leq M_i.lb[l]$ for $l \in \{1,2,\ldots,m\}$. However, $M_i'.lb = p_i^t$ and $p_i^t[l] \geq \min_{k\in[a,t]} p_i^k[l] = M'.lb$. Hence, $M_i'.ADR$ must not be smaller than that of M_i. □

MBR Reconstruction Strategy

MBR M_i^t that belongs to SKM^t but p_i^t is not included in SK^t for a long period should be lowered the rank to $NSKM^t$ to decrease the cardinality of skyline calculation in each snapshot. In this section, we discuss about a heuristic rule to decide which MBR should be reconstructed followed by a running example.

Firstly, a record of the number of consecutive false positives of each MBR should be tracked by adding a new MBR attribute called $M_i^t.cfp$. At each iteration we calculate this parameter for all MBRs in SKM^t as follows:

$$M_i^t.cfp = \begin{cases} M_i^{t-1}.cfp + 1 & ; M_i^t \in SKM^t \wedge p_i^t \notin SK^t \\ 0 & ; Otherwise \end{cases}$$

We decide to reconstruct a MBR M_i^t when

$$M_i^t.cfp \geq cfp_{ths} \tag{2}$$

where cfp_{ths} is a false positive tolerance threshold, i.e., $M_i^t : \forall k \in [t - cfp_{ths}, t] : M_i^k \in SKM^k \wedge p_i^k \notin SK^k$.

Lemma 4. *If M_i^t is reconstructed, a list of MBRs M_i^t dominates ($M_i^t.DRM$) remains the same.*

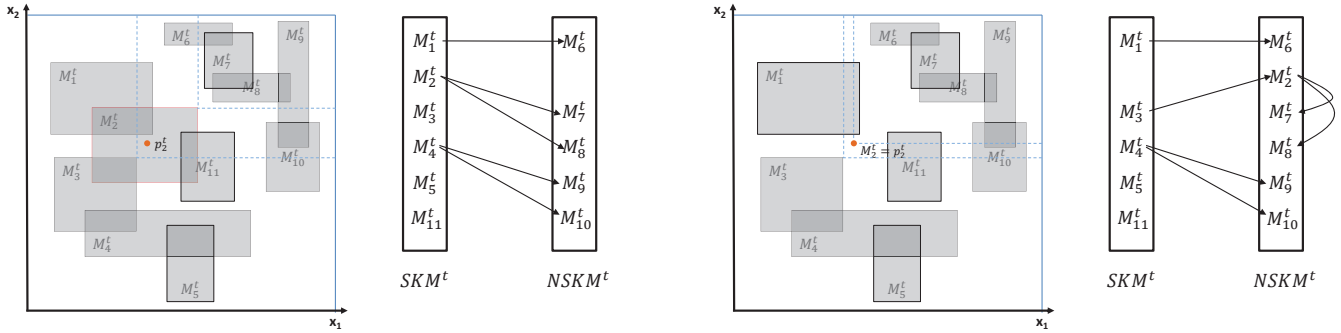
Proof. Due to Lemma 3, the dominance region of M_i^t does not become smaller. Therefore, all MBRs that M_i^t dominated before reconstruction are still dominated by M_i^t after the reconstruction. □

Lemma 5. *After $M_i^t \in SKM^t$ is reconstructed, M_i^t may change its membership to $NSKM^t$*

Proof. According to Lemma 3, the anti-dominance region of M_i^t may become larger. Therefore, it is possible that some $M_j^t \in SKM^t$ can dominate M_i^t. □

Running example

Fig.3 illustrates an example of a MBR reconstruction. In Fig 3a, there are 11 different MBRs in the space which can be classified to SKM^t and $NSKM^t$. An arrow from M_i^t to M_j^t



(a) Before M_2^t reconstruction

(b) After M_2^t reconstruction

Fig. 3: Running example of an MBR reconstruction

shows the relation that M_i^t dominates M_j^t, i.e., $M_j^t \in M_i^t.DRM$ and $M_j^t.admr = M_i^t$. Assume that M_2^t is decided to be reconstructed at snapshot t and the recent data point p_2^t is as shown in Fig.3a. After M_2^t's reconstruction (Fig.3b), both dominance and anti-dominance region of M_2^t have changed, and M_2^t is no longer in SKM^t because it is dominated by M_3^t while $M_2^t.DMR$ ($\{7, 8\}$) remains the same (no additional cost of finding). Note that $M_i^t \in NSKM^t$ can be dominated by some $M_j^t \in NSKM^t$ (not only limited to $M_j^t \in SKM^t$).

In summary, while the process in pre-calculation maintenance swaps some MBRs from $NSKM^t$ to SKM^t, the process in post-calculation maintenance dynamically swaps back some MBRs from SKM^t to $NSKM^t$. This responses to behaviors of data movement in an adaptive way.

6. Performance Evaluation

6.1 Datasets

In this experiment, we use both synthetic and real datasets to simulate and show our proposed method's performance.

(1) **Synthetic dataset (SYN):** Firstly, each data record p_i^0 is uniformly random on each dimension as a point on the m-dimensional data space $[0, 100]^m$. We model a data value on each dimension as a Gaussian random walk pattern following $p_i^t[l] = p_i^{t-1}[l] + u_i^t[l]$ where $u_i^t[l] = \lambda \cdot e_t[l]$, $e_t[l] \sim N(0, 0.5)$ (normal distribution), $1 \le l \le m$ and

$$\lambda = \begin{cases} 1 & \text{, with probability } p. \\ 0 & \text{, with probability } 1 - p. \end{cases} \quad (3)$$

Hence, in each snapshot, the values of a data object (a data point) change in some/all attributes with the probability $1 - (1 - p)^m$. Therefore, given N_D data objects in the entire system, the number of update tuples is expected to be $(1 - (1 - p)^m) \cdot N_D$ records in each snapshot, and the steps of the changed values in each attribute is according to a normal distribution. We generate different synthetic datasets by varying some parameters as shown in Table 2, which also shows their default values.

(2) **NBA Dataset (NBA):** This NBA dataset aggregated by the authors in [14] consists of all historical NBA information both game plays and player statistics between 1991 and 2004. We aim find the skyline of players being active over that time period (only players who play more than 30 matches). By taking the end of each game as one snapshot, we need to compute the skyline after every game play. In summary, there are 1225 players to be monitored, 16423 matches played (snapshots) and 312086 update tuples in total. We selected 5 useful attributes from a record of each player in each match including play time in minutes (MIN), points made (PTS), total rebounds (TOT), field goal made (FGM) and field goal attempts (FGA). However, we extracted 3 attributes to evaluate players including $PTSA = \sum PTS / \sum MIN$, $TOTA = \sum TOT / \sum MIN$ and $FGR = \sum FGM / \sum FGA$. In this dataset, we can see that there are only at most 24 players changed (about 2% of entire monitored players) their statistics in a consecutive snapshot.

Table 2: Parameter table for synthetic datasets

Parameter	Default	Range
N	5000	500-10000
m	3	2-8
T	10000	1000-50000
p	2.5%	0.5% - 20%

(3) **Stock Dataset (STK):** This Stock dataset aggregated from Yahoo! Finance[*1] consists of the daily information of all stocks in NYSE including open price, high price, low price, close price and volume between 2004 and 2013. For the scenario that we want to form a defensive investment portfolio, stocks that have lower beta (not fluctuate with the market) and a trend of increasing in price) than other stocks for a long period of time in the market are preferable. Therefore, we extract only 2 attributes including 200-day beta (β) and a 200-day slope of a regression line of close price (2 decimal precision). The system monitors which stock acts or holds this characteristic for a long period of time and no better other choices in the market (skyline). Therefore, we take a daily change of these attributes as a snapshot. After data cleansing, this dataset contains 1704 stocks, 2000 snapshots and 1621833 update tuples (averagely 47.6% of entire monitored stocks).

[*1] Yahoo! Finance: http://finance.yahoo.com/

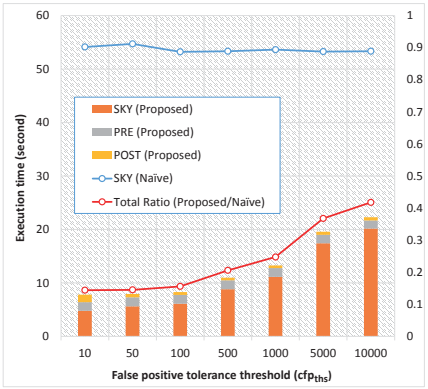


Fig. 4: Preliminary result on cfp_{ths}

6.2 Experiment setup

We conduct some experiments by implementing algorithms using C++ on a single commodity PC (Core i5 2.7GHz, 8GB of RAM, 64-bit Windows 7). We test our proposed algorithm as well as other competitive methods on the same environment. The system is to process a large file dataset which contain a series of $(D^0, U^1, U^2, \ldots, U^T)$ and find the desired output which is $(SK^0, SK^1, \ldots SK^T)$. Only synthetic datasets are generated 3 times in each case, and the results report the average case of them.

6.3 Benchmarks

We implemented the following methods for comparing with our proposed method.

(1) **Naive method (baseline):** compute the skyline by default skyline algorithm with entire data objects $(D^0, D^1, \ldots, D^T)$.

(2) **Our proposed method:** compute the skyline by default skyline algorithm with the data objects where their MBRs belong to SK^t described in Section 5.

6.4 Measurements

We measure the wall time clock of execution time. In the naive method, there is only computation time due to skyline computation (SKY(Naive)) while our proposed method's total computation time must include skyline computation time (SKY(Proposed)) and pre-computation maintenance time (PRE(Proposed)) and post-computation maintenance time (POST(Proposed)). In addition, a total ratio which is the total time of the proposed method divided by that of the naive method expresses the portion that our proposed method outperforms the naive method.

6.5 Parameter Tuning

The false positive tolerance threshold (cfp_{ths}) is only one system parameter described in the proposed method. Here, we study an effect of this parameter to decide a suitable value. Fig.4 shows the results of total computation time of the preliminary experiment by using a default-setting synthetic dataset. At low cfp_{ths}, POST(Proposed) is high because of frequent MBR reconstructions in that phase while SKY(Proposed) can be reduced from SKY(Naive) because of less false positives in SKM^t, and vice versa. It shows that setting low cfp_{ths} around 10-100 giving more preferable outcome than higher cfp_{ths}. In other words, paying some maintenance cost to renew non-potential MBRs in SKM^t is worthy and able to reduce the overall computation time. Therefore, we assign cfp_{ths} equal to 50 as a default parameter in all experiments.

6.6 Results of the synthetic dataset

6.6.1 Impact of *N*

Increasing the number of objects to be monitored affects the skyline computation time directly because of more candidates to be processed at each snapshot. In Fig.5a, the total execution time of the naive method tends to increase significantly because of a larger set of candidates in skyline calculation in each snapshot. While the total execution time of the proposed method also increases in the same way especially PRE(Proposed) and SKY(Proposed), the growth rate of the proposed method is quite smaller than that of the naive method noticed by a drop of the total ratio. This is because our proposed method identifies a much smaller set of candidates to be processed in skyline computation resulting in saving huge skyline computation cost. This result ensures that our proposed method is scalable on a large number of data objects.

6.6.2 Impact of *m*

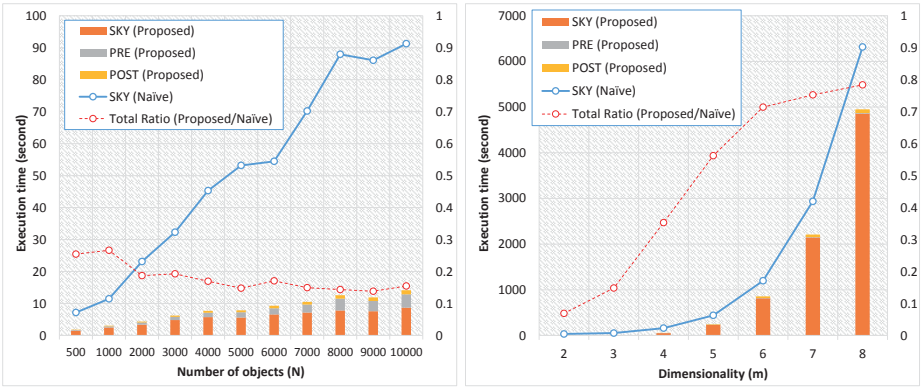
Normally the cardinality of skyline set is increasing exponentially with the number of dimensions resulting in slower skyline computation especially in the naive method shown in Fig.5b. In the same way, the total execution time of the proposed method goes up rapidly because the number of elements in SKM^t in each snapshot is likely to grow with dimensionality. Therefore, with the fixed number N, the number of data objects which can be safely pruned by MBRs becomes less in higher dimensionality. Nevertheless, our proposed method stills outperforms the naive method at least 20% even in high dimensionality, i.e., $d = 8$.

6.6.3 Impact of *T*

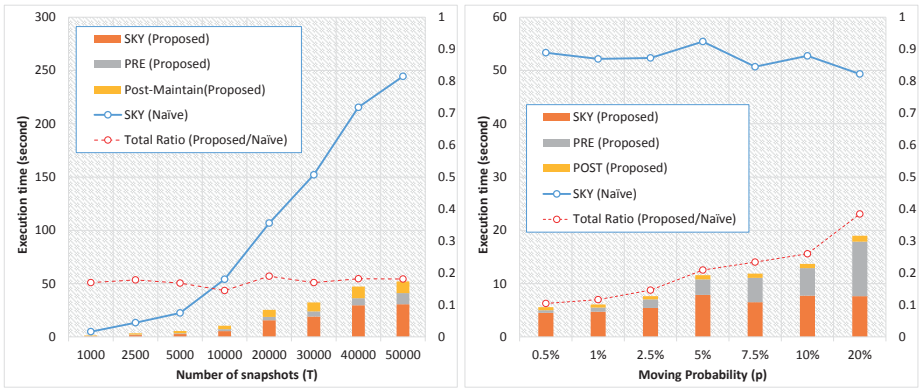
In this setting, we show the result of the proposed method when using for long periods of time. Increasing the number of snapshots directly increases the number of times of skyline calculation. Ideally, the total computation time should grow linearly with this factor. However, in practice, it also depends on the cardinality of output resulting from data distribution and data updates. The result in Fig.5c shows that the total ratio is kept constant from $T = 1000$ to $T = 50000$. This means the benefit from using our proposed method over the naive method is consistent regardless of usage duration.

6.6.4 Impact of *p*

Another factor that significantly affects the proposed method is how many data change in each snapshot; because, update tuples possibly invokes the frequent MBR inclusion checking which may lead to MBR updates (PRE(Proposed)) and MBR reconstructions (POST(Proposed)). In Fig.5d, while the total execution time of the naive method constantly fluctuates because of different data updates (different skyline output and calculation time), the total execution time of the proposed method tends to increase with p especially the cost of PRE(Proposed) as expected. However, the advantage of using the proposed method is still significant in spite of a large number of data updates at a snapshot.

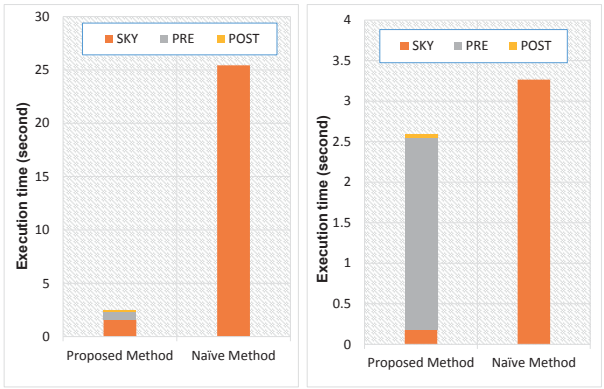


(a) Impact of N (b) Impact of m



(c) Impact of T (d) Impact of p

Fig. 5: Experimental results of the synthetic dataset (SYN)



(a) NBA dataset (NBA) (b) Stock dataset (STK)

Fig. 6: Experimental results of the real datasets

6.7 Results of the real datasets

We examine our proposed method's performance on 2 different real-life datasets which have totally different characteristics including NBA and STK. NBA is quite larger than STK in terms of the number of snapshots and the number of dimensions, but the moving probability averagely is only 2% for each snapshot. The result of NBA reported in Fig.6a show that the proposed method obviously beats the naive method an order of magnitude. We later found that, in this dataset, there is a group of outstanding players. The MBRs of this group are able to prune other candidates effectively with less MBR updates.

In the second dataset, STK, this dataset is related to stocks. Unlike NBA, the data objects are frequently changed (many update tuples) by its nature, and the moving probability is quite high averagely at 47.6%. As a consequence, the result of this dataset in Fig.6b reports the huge cost of PRE(Proposed) (mainly due to MBR checks and updates) as expected in the proposed method while the cost of SKY and POST are kept quite low. Nevertheless, our proposed method still saves the computation cost by 20% compared to the naive method.

7. Conclusion

In this paper, we proposed an efficient method for skyline calculation when there are many data updates at each data snapshot. This is useful for analyzing historical (time-series) data archive as well as continuous skyline computation on data update streams. In the assumed historical data series, the changes of data between consecutive timestamps are expressed and kept as update tuples. In practice, data insertion, deletion or any modification of a single data object between timestamp can totally change the final skyline set. Therefore, the naive method for this problem is to re-compute the new skyline set every timestamp (snapshot). This can be very expensive and time-consuming.

Basically the skyline computation'complexity largely depends the number of input (candidates), data distribution and the number of dimensions. The proposed method tries to reduce the expensive skyline computation cost by reducing the number of can-

didates in skyline computation. Our proposed method makes used of bounding boxes, i.e., MBRs to summarize and represent a series of data snapshots of each data object. Due to the properties of a MBR and our proposed data structures, we can identify a smaller set of candidates for skyline computation by pruning non-potential data objects while the accuracy can be guaranteed. Moreover, we also discuss about the maintenance of our index structures which is adaptive to data behaviors.

We compared the performance of our proposed method through the experiments by using both extensive synthetic dataset and 2 real datasets. The results obviously showed the benefits of our proposed method over the baseline by measuring the total execution time. Our proposed method can perform and process each dataset faster than the baseline because the number of candidates can be largely reduced while the maintenance cost is low resulting in the lower total execution time.

8. Acknowledgments

This research is partially supported by the Grant-in-Aid for Scientific Research (A)(26240013) of MEXT, Japan.

References

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[2] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang. A safe zone based approach for monitoring moving skyline queries. In *EDBT*, pages 275–286, 2013.

[3] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.

[4] K. Hose and A. Vlachou. A survey of skyline processing in highly distributed environments. *VLDB Journal*, 21(3):359–384, 2012.

[5] Y.-L. Hsueh, R. Zimmermann, and W.-S. Ku. Efficient updates for continuous skyline computations. In *DEXA*, pages 419–433, 2008.

[6] Z. Huang, H. Lu, B. C. Ooi, and A. Tung. Continuous skyline queries for moving objects. *IEEE TKDE*, 18(12):1645–1658, Dec 2006.

[7] B. Jiang and J. Pei. Online interval skyline queries on time series. In *ICDE*, pages 1036–1047, Washington, DC, USA, 2009. IEEE Computer Society.

[8] M.-W. Lee and S.-W. Hwang. Continuous skylining on volatile moving data. In *ICDE*, pages 1568–1575, March 2009.

[9] Y. W. Lee, K. Y. Lee, and M. H. Kim. Efficient processing of multiple continuous skyline queries over a data stream. *Information Sciences*, 221:316–337, 2013.

[10] N. Mamoulis, K. Berberich, S. Bedathur, et al. Durable top-k search in document archives. In *SIGMODS*, pages 555–566. ACM, 2010.

[11] M. Morse, J. M. Patel, and W. I. Grosky. Efficient continuous skyline computation. *Information Sciences*, 177(17):3411–3437, 2007.

[12] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 30(1):41–82, 2005.

[13] O. Papapetrou and M. Garofalakis. Continuous fragmented skylines over distributed streams. In *ICDE*, pages 124–135, 2014.

[14] A. Sultana, N. Hassan, C. Li, J. Yang, and C. Yu. Incremental discovery of prominent situational facts. 2014.

[15] S. Sun, Z. Huang, H. Zhong, D. Dai, H. Liu, and J. Li. Efficient monitoring of skyline queries over distributed data streams. *Knowledge and Information systems*, 25(3):575–606, 2010.

[16] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *ICDE*, pages 65–65, 2006.

[17] L. Tian, L. Wang, A. Li, P. Zou, and Y. Jia. Continuous skyline tracking on update data streams. In *APWeb/WAIM*, volume 4537, pages 192–197, 2007.

[18] H. Wang, Y. Cai, Y. Yang, N. Mamoulis, et al. Durable queries over historical time series data. *IEEE TKDE*, 26(3):595–607, March 2014.

[19] J. Xin, G. Wang, L. Chen, et al. Continuously maintaining sliding window skylines in a sensor network. In *DASFAA*, pages 509–521, 2007.