Efficient Maintenance of all k-Dominant Skyline Query Results for Frequently Updated Database

Md. Anisuzzaman Siddique*
University of Rajshahi
Rajshahi-6205, Bangladesh
Email: anis_cst@yahoo.com

Yasuhiko Morimoto
Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
Email: morimoto@mis.hiroshima-u.ac.jp

***Abstract*—Skyline queries are useful to multi-criteria decision making as they represent the set of all solutions that the user can safely take without fear that something better is out there. It can act as a filter to discard sub-optimal objects. However, a major drawback of skylines is that, in datasets with many dimensions, the number of skyline objects becomes large and no longer offer any interesting insights. To solve the problem, k-dominant skyline queries have been introduced, which can reduce the number of retrieved objects by relaxing the definition of the dominance. Though it can reduce the number of retrieved objects, the k-dominant skyline objects are difficult to maintain if the database is updated. This paper addresses the problem of maintaining k-dominant skyline objects for frequently updated database. We propose an algorithm for maintaining k-dominant skyline objects. An extensive performance evaluation using both real and synthetic datasets demonstrated that our method is efficient and scalable.**

***Keywords*-Skyline, k-Dominant Skyline, Database Update.**

I. Introduction

Abundance of data has been both a boon and a curse, as it has become increasingly difficult to process data in order to isolate useful and relevant information. In order to compensate, the research community has invested considerable effort into developing tools that facilitate the exploration of a data space. One such successful tool is the skyline query. The set of skyline objects presents a scale-free choice of data objects worthy for further considerations in many application contexts. Figure 1 shows a typical example of skyline. The table in the figure is a list of hotels, each of which contains two numerical attributes: distance to beach and price, for online booking. A tourist chooses a hotel from the list according to her/his preference. In this situation, her/his choice usually comes from the hotels in skyline, i.e., any of h_1, h_3, h_4 (see Figure 1(b)). Many approaches have been proposed for the efficient computation of skylines in the literature [2], [3], [5], [6], [8], [9], [10], [12].

The skyline query can greatly help user to narrow down the search range. It is always assumed that all the attributes are involved in the skyline queries, that is, the dominating relationship is evaluated based on every dimensions of the dataset. However, a major drawback of skylines is that, in datasets with many dimensions, the number of skyline objects becomes large and no longer offer any interesting insights. The reason is that as the number of dimensions increases, for any object O_1, it is more likely there exists another object O_2 where O_1 and O_2 are better than each other over different subsets of dimensions. If our tourist, cared not just about price and distance to beach, but also about the distance to airport, distance to downtown, rank, and internet charge then most hotels may have to be included in the skyline answer since for each hotel there may be no one hotel that beats it on all criteria.

To deal with this dimensionality curse, one possibility is to reduce the number of dimensions considered. However, which dimensions to retain is not easy to determine, and requires intimate knowledge of the application domain. To reduce the number of dimensions without any intimate knowledge of the application domain, Chan, *et al.* considered k-dominant skyline query [4]. They relaxed the definition of "dominated" so that an object is more likely to be dominated by another. Given an n-dimensional database, an object O_i is said to k-dominates another object O_j $(i \neq j)$ if there are k $(k \leq n)$ dimensions in which O_i is better than or equal to O_j. A k-dominant skyline object is an object that is not k-dominated by any other objects. Note that conventional skyline objects are n-dominant objects.

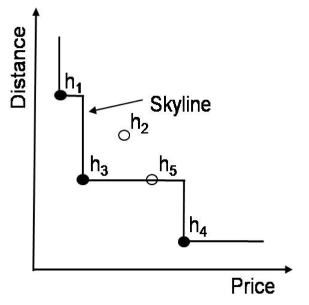
A. Motivating Example

Assume we have a symbolic dataset containing six attributes $(D_1, ..., D_6)$ as listed in Table I. Without loss of generality, we assume smaller value is better in each dimension. Conventional skyline query for this database returns five objects: O_2, O_3, O_5, O_6, and O_7. On the other, if we apply the k-dominant skyline query for this dataset it can control the selectivity by changing k. For example, if $k = 5$, the 5-dominant skyline query returns two objects: O_5 and O_7. Objects O_1, O_2, O_3, O_4, and O_6 are not in 5-

*This work was done when the author was in Hiroshima University.

ID	Price	Distance
h_1	3	8
h_2	5	6
h_3	4	4
h_4	9	2
h_5	7	4

(a) Hotels



(b) Skyline

Figure 1. Skyline example

Table I
SYMBOLIC DATASET

Obj.	D_1	D_2	D_3	D_4	D_5	D_6
O_1	7	3	5	4	4	3
O_2	3	4	4	5	1	3
O_3	4	3	2	3	5	4
O_4	5	3	5	4	1	2
O_5	1	4	1	1	3	4
O_6	5	3	4	5	1	5
O_7	1	2	5	3	1	2

dominant skyline because they are 5-dominated by O_7. The 4-dominant skyline query returns only one object, O_7 and the 3-dominant skyline query returns empty.

Though the k-dominant skyline query can control selectivity, but unfortunately, there exist no efficient methods that can handle k-dominant skyline queries under continuous updates. Therefore, we consider continuous queries, that is, queries that are executed when there are some changes in the dataset that affects their result. Continuous k-dominant skyline queries are very useful, since they allow users to monitor the dataset and get informed about new interesting objects. However, the maintenance of k-dominant skyline for an update is much more difficult due to the well-known intransitivity of the k-dominance relation. Assume that "A" k-dominates "B" and "B" k-dominates "C". However, "A" does not always k-dominates "C". Moreover, "C" may k-dominate "A". Because of the intransitivity property, we have to compare each object against every other object to check the k-dominance. To illustrate this problem consider the 5-dominant example again. In the dataset, objects O_5, O_7 are in 5-dominant skyline. If we insert a new object $O_{new} = (6, 4, 4, 2, 2, 3)$ into the dataset, we can compare O_{new} with the 5-dominant skyline objects (i.e., $\{O_5, O_7\}$) to maintain the 5-dominant skyline and after comparisons we may find that O_{new} is in the 5-dominant skyline. But it is not true, because O_{new} is 5-dominated by O_2. Like this example, for each insertion in the dataset, we have to perform domination check of each new object against all k-dominant as well as non-k-dominant skyline objects. This procedure is cost effective and we have to reduce the cost in order to handle frequent updates in a large dataset.

If an update is a deletion, we have to recompute the entire k-dominant skyline from the scratch. Because some objects that are not in the current k-dominant skyline objects may be "promoted" as k-dominant skyline objects by a deletion. Suppose, if we delete object O_2, this deletion will "promote" O_{new} as a 5-dominant skyline object. Again, if we want to revise or update the attributes values of an object. Then similar to deletion operation, we have to recompute the entire k-dominant skyline from the scratch. After this type of modification, three cases can be happened: some k-dominant skyline objects may be "removed", some objects that are not in the current k-dominant skyline objects may be "promoted" as k-dominant skyline objects, and both may be occurred at a time. For example, if we select object O_7 for update and revise the values of D_1 and D_5 from 1 to 6, then the 5-dominant skyline result updated as $\{O_5, O_4, O_7, O_2\}$. The focus of this paper is on developing an efficient algorithm for continuous k-dominant skyline objects.

This paper is the journal version of our paper [1]. The main contributions of this paper are as follows:

- We propose an algorithm to compute k-dominant skyline objects for all k at a time.
- We addresses the problem of continuous k-dominant skyline objects of frequently updated database.
- We develop algorithms for continuous k-dominant skyline objects.
- We conduct the extensive performance evaluation using both real and synthetic datasets and compare our method with the adaptive version of Two-Scan Algorithm (TSA) technique [4], which is currently considered the most efficient k-dominant skyline method. Our evaluation shows that the proposed method is significantly faster than the adaptive version of TSA technique.

The remaining sections of this paper are organized as follows: Section II presents the notions and properties of k-dominant skyline computation, we provide detailed examples and analysis of proposed k-dominant skyline computation and maintenance methods in Section III, Section IV discusses related work, we experimentally evaluate our algorithm in Section V by comparing with other existing algorithms under a variety of settings, finally, Section VI concludes the paper.

II. Problem Definition

Assume there is an n-dimensional database DB and D_1, D_2, $\cdots$, D_n be the n attributes of DB. Let O_1, O_2, $\cdots$, O_r be r objects (tuples) of DB. We use $O_i.D_s$ to denote the s-th dimension value of O_i. An object $O_i \in DB$ is said to dominate another object $O_j \in DB$, denoted as $O_i \prec O_j$, if (1) for every $s \in \{1, \cdots, n\}$: $O_i.D_s \leq O_j.D_s$; and (2) for at least one $t \in \{1, \cdots, n\}$: $O_i.D_t < O_j.D_t$. The skyline of DB is a set of objects $Sky_n(DB) \subseteq DB$ which are not dominated by any other objects. Skyline query for Table I dataset returns five objects: O_2, O_3, O_5, O_6, and O_7. Objects O_1 and O_4 are not in skyline because they are dominated by O_7.

In the remaining sections, we first define the k-dominance relationship using above notation, then introduce k-dominant skyline based on the definition of the k-dominance relationship.

A. k-dominance Relationship

An object O_i is said to *dominate* another object O_j, which we denote as $O_i \prec O_j$, if $O_i.D_s \leq O_j.D_s$ for all dimensions D_s $(s = 1, \cdots, n)$ and $O_i.D_t < O_j.D_t$ for at least one dimension D_t $(1 \leq t \leq n)$. We call such O_i as *dominant object* and such O_j as *dominated object* between O_i and O_j.

By contrast, an object O_i is said to *k-dominate* another object O_j, denoted as $O_i \prec_k O_j$, if (1) $O_i.D_s \leq O_j.D_s$ in k dimensions among n dimensions and (2) $O_i.D_t < O_j.D_t$ in one dimension among the k dimensions. We call such O_i as *k-dominant object* and such O_j as *k-dominated object* between O_i and O_j.

An object O_i is said to have *δ-domination power* if there are δ dimensions in which O_i is better than or equal to all other objects of DB.

B. k-dominant Skyline

An object $O_i \in DB$ is said to be a *skyline object* of DB if O_i is not dominated by any other object in DB. Similarly, an object $O_i \in DB$ is said to be a *k-dominant skyline object* of DB if O_i is not k-dominated by any other object in DB. We denote a set of all k-dominant skyline objects in DB as $Sky_k(DB)$.

Theorem 1: *Any object in $Sky_{k-1}(DB)$ must be an object in $Sky_k(DB)$ for any k such that $1 < k \leq n$. Any object that is not in $Sky_k(DB)$ cannot be an object in $Sky_{k-1}(DB)$ for any k such that $1 < k \leq n$.*

Proof: Based on the definition, a $(k-1)$-dominant skyline object O_i is not $(k-1)$-dominated by any other objects in DB. It implies that O_i is not k-dominated by any other objects. Therefore, we can say that O_i is a k-dominant skyline object. Similarly, if an object O_j is k-dominated by another object, it must be $(k-1)$-dominated by the object. Therefore, any k-dominated object cannot be a $(k-1)$-dominant skyline object. $\diamondsuit$

The conventional skyline is the k-dominant skyline where $k = n$. If we decrease k, more objects tend to be k-dominated by other objects. As a result, we can reduce the number of k-dominant skyline objects. Using above properties, we can compute $Sky_{k-1}(DB)$ from $Sky_k(DB)$ efficiently. For example, O_1 and O_4 of Table I are not in $Sky_6(DB)$ because they are 6-dominated by O_7. Therefore, they cannot be a candidate of k-dominant skyline object for $k < 6$. We can prune such non-skyline objects for further procedure of the k-dominant query. If we consider 5-dominant query, then O_2, O_3, and O_6 are 5-dominated objects. Therefore, we can prune all of those five objects in 4-dominant query computation. Thus, by decreasing k, more dominated objects can be pruned away.

Theorem 2: *Every object that belongs to the k-dominant skyline also belongs to the skyline, i.e., $Sky_k(DB) \subseteq Sky_n(DB)$.*

Proof: Immediate from theorem 1. $\diamondsuit$

III. k-dominant Skyline Computation and Maintenance

In this section, we present our algorithm for computing k-dominant skyline objects and how to maintain the result when update occurs. We illustrate how to compute k-dominant skyline for all k at a time in section III-A. Next in section III-B, we present three types of maintenance solution. They are insertion, deletion, and update operation.

A. Algorithm for k-dominant Skyline

Chan, *et al.* sort the whole objects with a monotonic scoring function sum in their One-Scan Algorithm (OSA), algorithm for k-dominant query [4]. By using the ordered objects, we can eliminate some of non-skyline objects easily. However, this ordered objects is not effective for k-dominant query computation, especially, when values of each attribute is not normalized. For example, assume $O_i = (1, 2, 3, 3, 3, 2)$ and $O_j = (7, 1, 3, 2, 3, 1)$ are two objects in 6-dimensional space. Although sum of O_i's values is smaller than that of O_j's, O_i does not 5-dominant of O_j. Instead, O_i is 5-dominated by O_j.

In order to prune unnecessary objects efficiently in the k-dominant skyline computation, we compute *domination power* of each object. Algorithm 1 represents the domination power calculation procedure. We sort objects in descending order by domination power. If more than one objects have the same domination power then sort those objects in ascending order of the sum value. This order reflects how likely to k-dominate other objects.

Algorithm 1: *Compute Domination Power, DP*

```
01. for each object $O_i(i = 1 \cdots r)$ do
02.     $O_i$.DP := 0 (initialize DP for each object)
03. for each attribute $D_s(s = 1 \cdots n)$ do
04.     minValue := $O_1.D_s$
05.     for each object $O_i(i = 2 \cdots r)$ do
06.         if ($O_i.D_s <$ minValue) then
07.             minValue := $O_i.D_s$
08.     for each object $O_i(i = 1 \cdots r)$ do
09.         if (minValue == $O_i.D_s$) then
10.             $O_i$.DP := $O_i$.DP + 1 (Increment DP)
11. Sort dataset, $DB$, in descending order by DP and Sum
```

Table II is the sorted object sequence of Table I, in which the column "DP" is the domination power and the column "Sum" is the sum of all values. In the sequence, object O_7 has the highest domination power 4. Note that object O_7 dominates all objects lie below it in four attributes, D_1, D_2, D_5, and D_6.

After computing the sorted object sequence, we compute dominated counter (DC) and dominant index (IDX) by using the algorithm 2. The dominated counter (DC) indicates the maximum number of dominated dimensions by another object in DB. The dominant index (IDX) is the strongest dominator. That means IDX keeps the record of the corresponding strongest dominator for each object.

For example, O_7 is 3-dominated by O_5 and there exist no other object which can 4-dominate O_7. In the algorithm (8-13), we count the number of dominated dimensions for each pair of objects. In the algorithm, we denote the i-th object in the sorted sequence as O_i. During the procedure, we keep the max value and its dominator object in DC and IDX for each object.

The column DC and IDX of Table II shows the result of the procedure. $Sky_k(DB)$ is a set of objects whose DC is less than k. According to the dominated counter, we can see that $Sky_6(DB) = \{O_7, O_5, O_2, O_6, O_3\}$, $Sky_5(DB) = \{O_7, O_5\}$, and $Sky_4(DB) = \{O_7\}$. Since there is no object whose DC value is less than 3, thus $Sky_3(DB) = \{\emptyset\}$.

B. *k-dominant Skyline Maintenance*

In this section, we discuss the maintenance problem of $Sky_k(DB)$ after an update is occurred in DB. In the maintenance phase, we keep a vector that contains the minimal value for each dimension, which we call the minimal vector. The minimal vector of Table II is $\{1, 2, 1, 1, 1, 2\}$. We also keep an inverted index of the dominant index column (IDX). Table III is the inverted index of Table II. We use a multi-hash data structure for the records in the inverted index so that we can look up efficiently.

Lemma 1: Assume $O_1.D_s \leq O_2.D_s$ for all dimensions D_s $(s = 1, \cdots, n)$ for $O_1, O_2 \in DB$. If O_1 is not deleted

Algorithm 2: *Compute DC and IDX*

```
01. for each object $O_i$ (i := 1 $\cdots$ r)
02.     $O_i.DC := 0$ (initialize DC)
03. for each i := 1 $\cdots$ r
04.     if $O_i.DC == n$ then
05.         skip the i-th and continue
06.     for each j := i+1 $\cdots$ r
07.         if $O_j.DC == n$ then
08.             skip the j-th and continue
09.         $k_i := 0; k_j := 0$
10.         for each attribute $D_s(s := 1 \cdots n)$
11.             if $O_i.D_s \leq O_j.D_s$ then
12.                 $k_j++$
13.             if $O_j.D_s \leq O_i.D_s$ then
14.                 $k_i++$
15.         if $k_j > O_i.DC$ then
16.             $O_i.DC := k_j$ and $O_i.IDX := O_j$
17.         if $k_i > O_j.DC$ then
18.             $O_j.DC := k_i$ and $O_j.IDX := O_i$
```

Table II
ORDERED DOMINATION TABLE

Obj.	D_1	D_2	D_3	D_4	D_5	D_6	DP	Sum	DC	IDX
O_7	$\underline{1}$	$\underline{2}$	5	3	$\underline{1}$	$\underline{2}$	4	14	3	O_5
O_5	$\underline{1}$	4	$\underline{1}$	$\underline{1}$	3	4	3	14	4	O_7
O_4	5	3	5	4	$\underline{1}$	$\underline{2}$	2	20	6	O_7
O_2	3	4	4	5	$\underline{1}$	3	1	20	5	O_7
O_6	5	3	4	5	$\underline{1}$	5	1	23	5	O_7
O_3	4	3	2	3	5	4	0	21	5	O_7
O_1	7	3	5	4	4	3	0	26	6	O_7

Algorithm 3: *Insertion Procedure*

```
01. $O_I$.DC := 0 (initialize DC for inserted object $O_I$)
02. for each i := 1 $\cdots$ r
03.     if $O_i$.DC == n then
04.         skip the i-th and continue
05.     $k_i := 0; k_I := 0$
06.     for each attribute $D_s(s := 1 \cdots n)$
07.         if $O_i.D_s \leq O_I.D_s$ then
08.             $k_I++$
09.         if $O_I.D_s \leq O_i.D_s$ then
10.             $k_i++$
11.     if $k_i > O_i.DC$ then
12.         $O_I.DC := k_i$ and $O_I.IDX := O_i$
      (update inverted index)
13.     if $O_I.DC == n$ then
14.         break loop
15.     if $k_I > O_i.DC$ then
16.         $O_i.DC := k_I$ and $O_i.IDX := O_I$
      (update inverted index)
```

Table III
INVERTED INDEX

Obj.	dominated
O_5	O_7
O_7	O_5, O_4, O_2, O_6, O_3, O_1

Table IV
ORDERED DOMINATION TABLE AFTER INSERTIONS

Obj.	DP	Sum	DC	IDX
O_7	4	14	3	O_5
O_5	3	14	4	O_7
O_4	2	20	6	O_7
O_2	1	20	5	O_7
O_6	1	23	5	O_7
O_9	0	18	5	O_2
O_3	0	21	5	O_7
O_1	0	26	6	O_7
O_8	0	36	6	O_7

Obj.	DP	Sum	DC	IDX
O_7	4	14	3	O_5
O_5	3	14	4	O_7
O_{10}	2	13	4	O_7
O_4	2	20	6	O_7
O_2	1	20	5	O_7
O_6	1	23	5	O_7
O_9	0	18	6	O_{10}
O_3	0	21	5	O_7
O_1	0	26	6	O_7
O_8	0	36	6	O_7

from DB, O_2 will never be in the k-dominant skyline of DB.

Proof: Since O_2 is n-dominated by O_1, it will also k-dominated by O_1 because $(k \leq n)$. Therefore, while O_2 is in the DB, there will be at least one object, namely O_1, that k-dominate it and consequently cannot become a k-dominant skyline. $\diamondsuit$

The lemma implies that only $Sky_n(DB)$ objects are relevant for the k-dominant skyline maintenance task, since according to theorem 1 other objects can never become part of $Sky_k(DB)$. Because, they are already n-dominated by $Sky_n(DB)$ objects.

Insertion

Assume O_I is inserted into DB. We maintain $Sky_k(DB)$ by using algorithm 3. In the algorithm, we examine the dominated counter (DC) by scanning the sorted object sequence. If the DC value of an object is updated, we also update the inverted index. According to theorem 2, any data object added to k-dominant skyline during the execution of the algorithm is guaranteed to be a skyline object. Thus, during the update procedure, if O_I is n-dominated by another object, then we can stop the procedure immediately.

After updating DC and IDX, we insert O_I in the sorted object sequence. We use a skip list data structure for the sequence so that we can insert efficiently.

Assume we insert $O_8 = (6, 6, 6, 6, 6, 6)$ in the running example. By comparing with the first object O_7, we can find that O_8 is 6 dominated by O_7. Therefore, we immediately complete the update of DC and IDX. Then, we insert O_8 into the last position of the sorted sequence. Next, we insert $O_9 = (3, 4, 4, 2, 2, 3)$. O_9 is not 6 dominated by any object and we can find that the strongest dominator of O_9 is O_2. Then, DC and IDX are updated as in Table IV (left) after inserting O_9. Next, we insert $O_{10} = (2, 2, 3, 1, 2, 3)$. O_{10} is not 6 dominated by any object and the strongest dominator of O_{10} is O_7. Moreover, O_{10} becomes the strongest dominator of O_9. Then, DC and IDX are updated as in Table IV (right) after inserting O_{10}.

Deletion

Assume O_D is deleted from DB. We check the inverted index to examine whether there is O_D's record in the index.

Note that in Table III "Obj." column in each record is the strongest dominator for objects in the "dominated" column.

Algorithm 4: *Deletion Procedure*

```
1. for each $O_D$ do
2.   if $O_D \notin IDX$
3.     no change in Dominant Counter
4.     break loop
5.   else
6.     for each affected objects do
7.       Compute $DC$ and $IDX$ by using algorithm 2
```

Therefore, if there is no O_D's record in the inverted index, we do not have to change the dominated counter (DC) for other objects. Otherwise, we initialize DC for each object in the O_D's index record. Then, we perform the pairwise comparison like in algorithm 2. In this case, we do not have to examine DC for objects that are not in the O_D's index record and therefore it is not costly.

Consider the running example again. Assume we delete O_{10}. We examine DC of O_9 by scanning Table IV (right). The updated result is Table IV (left).

Update

In this section, we propose maintenance solution for update operation. Although one can handle update operation with consecutive deletion and insertion. However, single update operation is usually faster than consecutive delete and insert operation. Assume O_U is updated from DB. Then, same as delete, we have to check the inverted index to examine whether there is O_U's record in the index. If there is no O_U's record in the inverted index, then we need one scan to revise the dominated counter (DC) for updated object as well as all other data objects. Otherwise, we initialize DC for each object from scratch. However, in this situation we argue that compare with non-IDX objects, the number of IDX objects is very few and therefore update operation also not very costly.

Consider the Table II and select a non-IDX object such as O_3 for update. Assume we update D_5 value of this object from 5 to 1. Then after dominance checking with other objects we find that O_3 becomes the strongest dominator of object O_6. This update procedure is shown in Table V (left). Again from Table II if we select an IDX object such as O_7. In this case, we update the values of D_1 and D_5

Table V
ORDERED DOMINATION TABLE AFTER UPDATES

Obj.	DP	Sum	DC	IDX
O_7	4	14	3	O_5
O_5	3	14	4	O_7
O_4	2	20	6	O_7
O_3	1	17	5	O_7
O_2	1	20	5	O_7
O_6	1	23	6	O_3
O_1	0	26	6	O_7

Obj.	DP	Sum	DC	IDX
O_5	3	14	3	O_4
O_4	2	20	4	O_7
O_7	2	24	4	O_5
O_2	1	20	4	O_5
O_6	1	23	5	O_4
O_3	0	21	5	O_5
O_1	0	26	6	O_4

from 1 to 6. The revised result after this update is shown in Table V (right).

IV. RELATED WORK

Our work is motivated by previous studies of skyline query processing as well as k-dominant skyline query processing, which are reviewed in this section. Section IV-A and IV-B, respectively discuss about the related work on skyline query processing and k-dominant skyline query processing.

A. Skyline Query Processing

Borzsonyi, *et al.* first introduce the skyline operator over large databases and proposed three algorithms: $Block\text{-}Nested\text{-}Loops(BNL)$, $Divide\text{-}and\text{-}Conquer(D\&C)$, and B-tree-based schemes [2]. BNL compares each object of the database with every other object, and reports it as a result only if any other object does not dominate it. A window W is allocated in main memory, and the input relation is sequentially scanned. In this way, a block of skyline objects is produced in every iteration. In case the window saturates, a temporary file is used to store objects that cannot be placed in W. This file is used as the input to the next pass. $D\&C$ divides the dataset into several partitions such that each partition can fit into memory. Skyline objects for each individual partition are then computed by a main-memory skyline algorithm. The final skyline is obtained by merging the skyline objects for each partition. Chomicki, *et al.* improved BNL by presorting, they proposed $Sort\text{-}Filter\text{-}Skyline(SFS)$ as a variant of BNL [3]. SFS requires the dataset to be pre-sorted according to some monotone scoring function. Since the order of the objects can guarantee that no object can dominate objects before it in the order, the comparisons of tuples are simplified.

Among index-based methods, Tan, *et al.* proposed two progressive skyline computing methods Bitmap and Index [13]. Both of them require preprocessing. In the Bitmap approach, every dimension value of an object is represented by a few bits. By applying bit-wise and operation on these vectors, a given object can be checked if it is in the skyline without referring to other objects. The index method organizes a set of d-dimensional objects into d lists such that an object O is assigned to list i if and only if its value at attribute i is the best among all attributes of O. Each list is indexed by a B-tree, and the skyline is computed by scanning the B-tree until an object that dominates the remaining entries in the B-trees is found. Kossmann, *et al.* observed that the skyline problem is closely related to the nearest neighbor (NN) search problem [8]. They proposed an algorithm that returns skyline objects progressively by applying nearest neighbor search on an R*-tree indexed dataset recursively. The current most efficient method is $Branch\text{-}and\text{-}Bound\ Skyline(BBS)$, proposed by Papadias, *et al.*, which is a progressive algorithm based on the best-first nearest neighbor (BF-NN) algorithm [10]. Instead of searching for nearest neighbor repeatedly, it directly prunes using the R*-tree structure. Balke, *et al.* show how to efficiently perform distributed skyline queries and thus essentially extend the expressiveness of querying current Web information systems [14]. Kapoor studies the problem of dynamically maintaining an effective data structure for an incremental skyline computation in a 2-dimensional space [15]. Tao and Papadias studied sliding window skylines, focusing on data streaming environments [16]. Huang, *et al.* studied continuous skyline queries for dynamic datasets [17].

B. k-dominant Skyline Query Processing

Chan, *et al.* introduce k-dominant skyline query [4]. They proposed three algorithms, namely, One-Scan Algorithm (OSA), Two-Scan Algorithm (TSA), and Sorted Retrieval Algorithm (SRA). OSA uses the property that a k-dominant skyline objects cannot be worse than any skyline object on more than k dimensions. This algorithm maintains the skyline objects in a buffer during the scan of the dataset and uses them to prune away objects that are k-dominated. TSA retrieves a candidate set of dominant skyline objects in the first scan by comparing every object with a set of candidates. The second scan verifies whether these objects are truly dominant skyline objects or not. This method turns out to be much more efficient than the one-scan method. A theoretical analysis is provided to show the reason for its superiority. The third algorithm, SRA is motivated by the rank aggregation algorithm proposed by Fagin, *et al.*, which pre-sorts data objects separately according to each dimension and then merges these ranked lists [7].

Another study on computing k-dominant skyline is $k\text{-}ZSearch$ proposed by Lee, *et al.* [18]. They introduced a concept called filter-and-reexamine approach. In the filtering phase, it removes all k-dominant objects and retain possible skyline candidates, which may contain false hits. In the reexamination phase, all candidates are reexamined to eliminate false hits.

For any static dataset in case of insertions and deletions the k-dominant skyline result should be updated accordingly. But in a dynamic dataset insertions and deletions are very frequent and the above schemes [4], [18] are not efficient to solve the frequent update problem. Because they need to recompute k-dominant skyline result from scratch. On the other hand, algorithms developed for skyline maintenance are not easily adapted for the maintenance of k-dominant skyline, except for the obvious case where $k = n$. This is because existing skyline computation algorithms do not satisfy the requirement of k-dominant skyline computation. Moreover, they can not compute k-dominant skyline for all k at a time. To overcome frequent update problem, our proposed method introduce two new concepts $dominated\ counter$ and $inverted\ index$. From the discussion and experimental results it has been seen that those are very helpful to retrieve the k-dominant skyline query result efficiently.

Recently, more aspects of skyline computation have been explored. Vlachou, *et al.* introduce the concept of extended skyline set, which contains all data elements that are necessary to answer a skyline query in any arbitrary subspace [19]. Fotiadou, *et al.* mention about the efficient computation of extended skylines using bitmaps in [20]. Chan, *et al.* introduce the concept of *skyline frequency* to facilitate skyline retrieval in high-dimensional spaces [5]. Tao, *et al.* discuss skyline queries in arbitrary subspaces [11]. There exist more work addressing *spatial skyline* [21], [22], skylines on partially-ordered attributes [6], *dada cube* for analysis of dominance relationships [23], *probabilistic skyline* [24], skyline search over small domains [9], and *reverse skyline* [25].

V. Performance Evaluation

We conduct a series of experiments to evaluate the effectiveness and efficiency of our proposed methods. In lack of techniques dealing directly with the problem of maintaining k-dominant skyline in this paper, we compare our methods against TSA, which was the most efficient k-dominant skyline search algorithm proposed in Ref. 4). To handle updates, we adapt a variant of the TSA called ATSA (Adaptive Two-Scan Algorithm). Let r be the total number of objects in DB. ATSA takes $O(r^2)$ to compute all k-dominant objects from scratch. If an object is inserted in the DB, ATSA has to perform k-domination check of the inserted object against all objects. Therefore, for each insertion ATSA takes $O(r)$. If an object is deleted from the DB, ATSA has to recompute entire k-dominant skyline objects because some objects that are not in the current k-dominant skyline objects may be "promoted" as k-dominant skyline objects. Therefore, for each deletion ATSA requires $O(r^2)$ time. Moreover, for each update it also requires $O(r^2)$ time for the recomputation of k-dominant skyline.

Though the time complexity of our proposed method is substantially the same, we can drastically reduce comparisons for k-dominant skyline computation. For each new insertion, the time complexity of proposed method varies in between $O(1)$ and $O(r)$ to perform k-domination check. For each deletion, if the deleted object is not in the dominant objects list then the proposed method takes $O(1)$. Otherwise, if we assume the number of dominant objects is x, then it takes $O(x^2)$. We can expect that x is much smaller than r. Finally for each update, if the updated object is not in the dominant objects list then the proposed method takes $O(r)$. Otherwise, it takes $O(r^2)$. However, the number of dominant objects is not large. So it is not costly. From the above analysis we understand that the recomputation of k-dominant skyline is not efficient than proposed maintenance solutions. The results of all experiments support our claim that using proposed techniques we can reduce the number of comparisons drastically.

We conduct simulation experiments on a PC running on MS Windows XP professional. The PC has an Intel(R) Core2 Duo 2GHz CPU and 3GB main memory. All experiments are coded in Java J2SE V6.0. Each experiment is repeated five times and the average result is considered for performance evaluation.

A. Performance on Synthetic Datasets

As benchmark synthetic datasets, we use the datasets proposed in Ref. 2). Objects are generated using one of the following three value distributions:

Anti-Correlated: an anti-correlated dataset represents an environment in which, if an object has a small coordinate on some dimension, it tends to have a large coordinate on at least another dimension. As a result, the total number of non-dominating objects of an anti-correlated dataset is typically quite large.

Correlated: a correlated dataset represents an environment in which objects with large coordinate in one dimension are also have large coordinate in the other dimensions. In a correlated dataset, few objects dominate many other objects.

Independent: for this type of dataset, all attribute values are generated independently using uniform distribution. Under this distribution, the total number of non-dominating objects in between that of the correlated and the anti-correlated datasets.

Details of the three distributions can be found in Ref. 1). The generation of the synthetic datasets is controlled by three parameters, n, "Size", and "Dist", where n is the number of attributes, "Size" is the total number of objects in the dataset, and "Dist" can be the any of the three distribution. In addition, we have generated smaller synthetic datasets for all insertion experiments. For example, to conduct insertion experiment on 100k synthetic dataset, we have also generated additional 10k dataset. As for deletion and update experiments, we choose the deleted/updated objects randomly from the experimental dataset.

1) Effect of Data Distribution: We first study the effect of data distributions on our techniques. Anti-correlated, independent, and correlated datasets with dimensionality n to 7, cardinality to 100k, and k to 6. Figure 2(a), (b), and (c) shows the time to maintain k-dominant skyline for update ranges from 1% to 5%. In the update experiments, for 100k dataset 1% update implies that we have altered 1000 data objects. Those data objects are randomly selected. However, among 1000 alterations we make sure that there

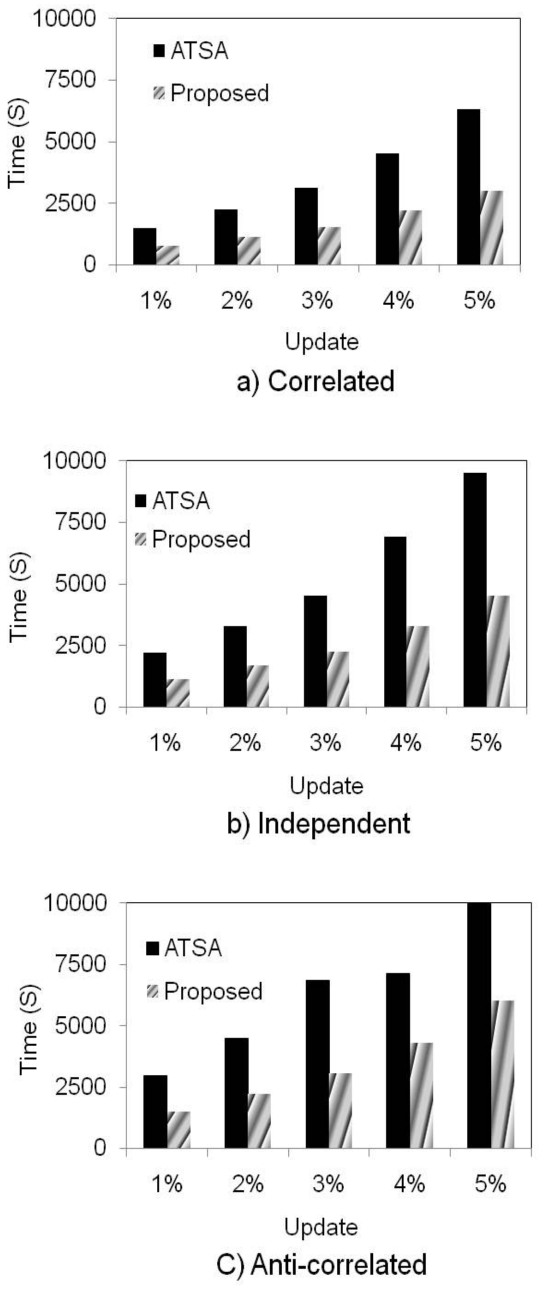


Figure 2. Update performance for different data distributions

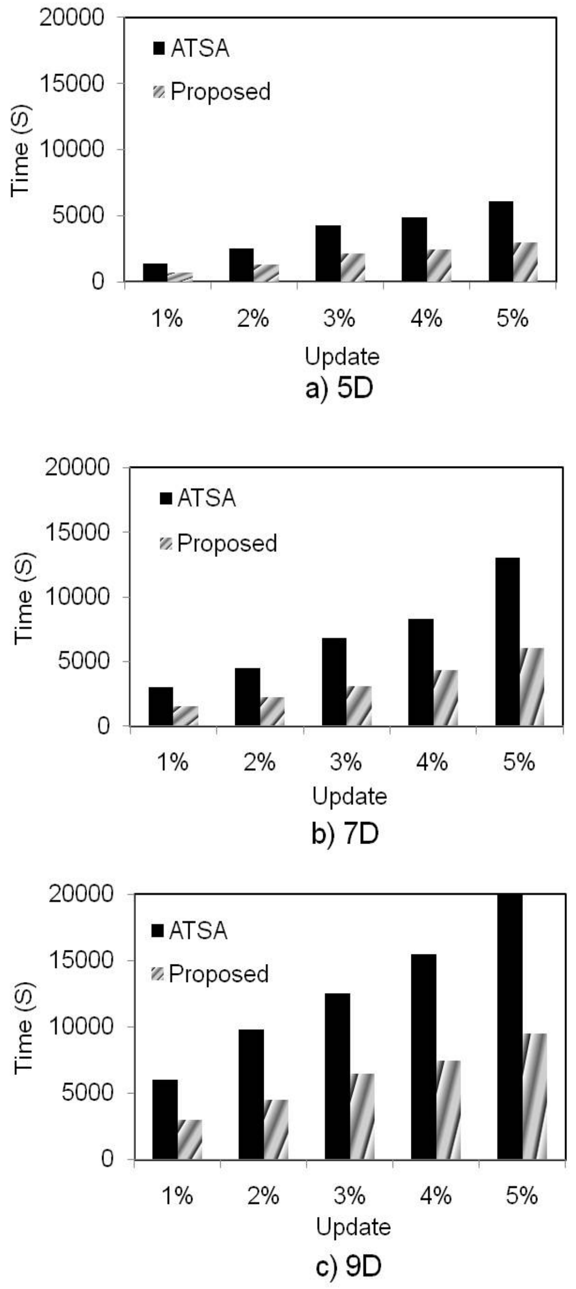


Figure 3. Update Performance for different dimensions

are 333 insertions, 333 deletions, and 334 updates (total 1k updates) occurred in the dataset. Figure 2 shows that the performance of both methods deteriorates significantly with the increase value of update size. We further notice that for anti-correlated dataset, many k-dominant skyline objects are retrieved, as a result the maintenance cost of this distribution incurs high computational overheads. On the other hand, for correlated dataset, few objects are retrieved, as a result the maintenance cost of this distribution incurs low computational overheads.

2) Effect of Dimensionality: For this experiment, we use the anti-correlated datasets. We fix the data cardinality to 100k and vary dataset dimensionality n ranges from 5 to 9 and k from 4 to 8. Figure 3(a), (b), and (c) shows the update performance. The ATSA technique is highly affected by the curse of dimensionality, i.e., as the space becomes sparser its pruning power rapidly decreases. The proposed technique also affected but to a lesser degree. The Figure 3 shows that if the dimensionality and the update ratio increase the time grows steadily, which is much less than that of ATSA.

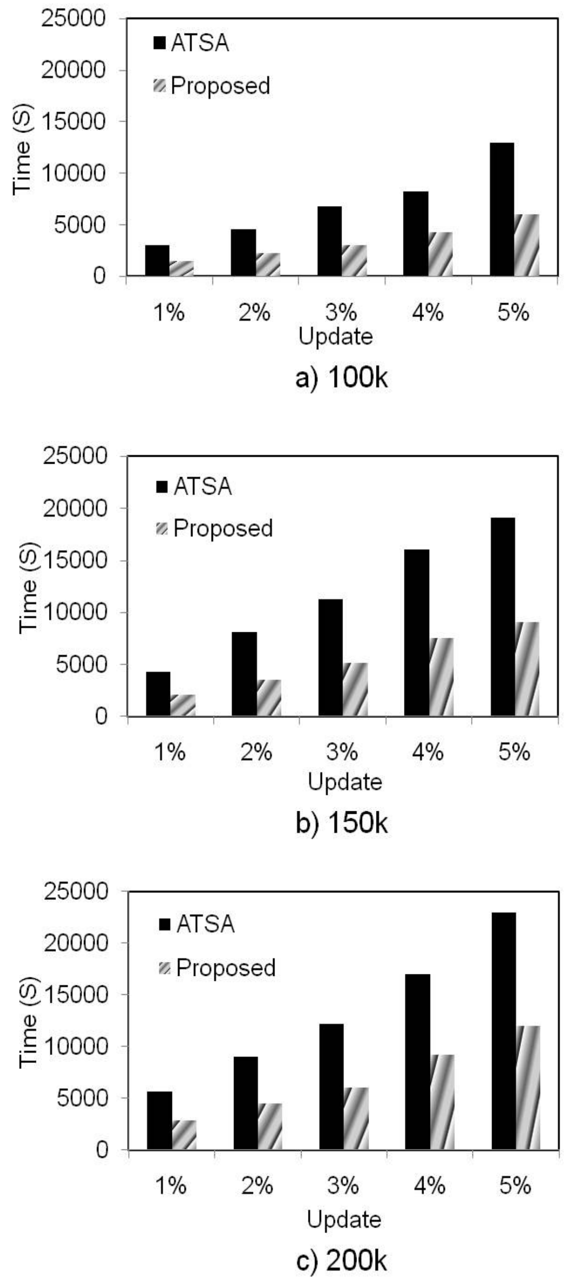


Figure 4. Update performance for different datasize

3) Effect of Cardinality: For this experiment, we use the anti-correlated datasets with varying dataset cardinality ranges from 100k to 200k and set the value of n to 7 and k to 6. Figure 4(a), (b), and (c) shows the time to maintain k-dominant skyline for update ranges from 1% to 5%. The result shows that if the update ratio and data cardinality increases the maintenance time of proposed method also increases. Even though, the time is much smaller than that of ATSA.

B. Performance on Real Datasets

To evaluate the performance for real dataset, we study two different real datasets. The first dataset is NBA statistics. It is extracted from "`www.nba.com`". The dataset contains 17k 13-dimensional data objects, which correspond to the statistics of an NBA players' performance in 13 aspects (such as points scored, rebounds, assists, etc.) and domain have range [0, 4000]. The dataset approximates a correlated data distribution. The second dataset is FUEL dataset and extracted from "`www.fueleconomy.gov`". FUEL dataset is 24k 6-dimensional objects, in which each object stands for the performance of a vehicle (such as mileage per gallon of gasoline in city and highway, etc). For this dataset attribute domain range is [8, 89]. Using both datasets we conduct the following experiment.

1) Experiments on NBA and FUEL datasets: We performed an experiment on NBA dataset. In this experiment, we study the effect of update and set the value of n to 13, and k to 12. Figure 5(a) shows the result. NBA dataset exhibits similar result to synthetic dataset, if the number of updates increases the performance of proposed algorithm becomes slower.

For FUEL dataset, we performed similar experiment like NBA dataset. For this experiment, we set the value of n to 6 and k to 5. Result is shown in Figure 5(b). In this experiment with FUEL dataset, we obtain similar result like NBA dataset that represents the scalability of the proposed method on real datasets. However, in both cases proposed method outperform than ATSA method.

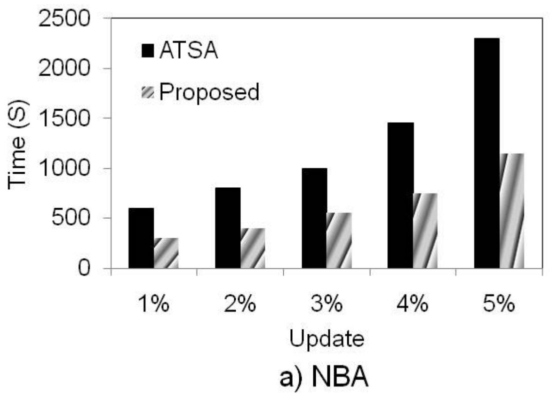
VI. Conclusion

Compared with skyline query processing, k-dominant skyline result maintenance is a relatively new research area. The k-dominant skyline objects are difficult to maintain if the database is updated. However, in lack of techniques dealing directly with the problem of maintaining k-dominant skyline in this paper we propose k-dominant skyline computation and maintenance algorithms for a frequently updated database. As shown later, this technique can produce k-dominant skyline update result for all k at a time. Besides theoretical guarantees, our comprehensive performance study indicate that the proposed maintenance framework is very effective and efficient.

We leave as future work extensions to explore precomputation techniques to further speed up the computation of k-dominant skyline query. Future works should investigate the efficient maintenance of k-dominant skyline for batch updates. To increase the pruning power is another big challenge for continuous k-dominant skyline computation.

Acknowledgements

This work was supported by KAKENHI (19500123). Md. Anisuzzaman Siddique was supported by the scholarship of MEXT Japan.



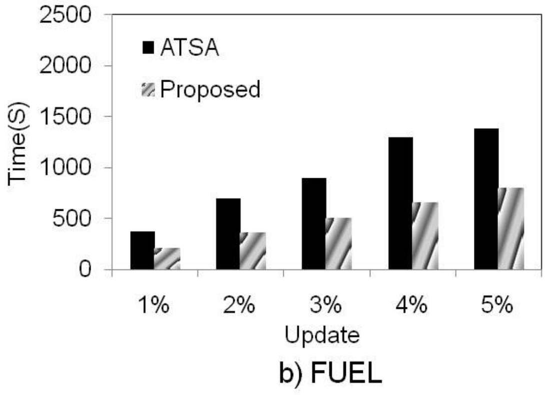


Figure 5. Experiments on NBA and FUEL datasets

REFERENCES

[1] M. A. Siddique and Y. Morimoto, "Efficient Maintenance of k-Dominant Skyline for Frequently Updated ", in: Proceedings of DBKDA, 2010, pp. 107-110.

[2] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator", in: Proceedings of ICDE, 2001, pp. 421-430.

[3] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting", in: Proceedings of ICDE, 2003, pp. 717-719.

[4] C. Y. Chan, H. V. Jagadish, K-L. Tan, A-K. H. Tung, and Z. Zhang, "Finding k-dominant skyline in high dimensional space", in: Proceedings of ACM SIGMOD, 2006, pp. 503-514.

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

[6] C.-Y. Chan, P.-K. Eng, and K.-L. Tan, "Stratified computation of skylines with partially-ordered domains", in: Proceedings of ACM SIGMOD, 2005, pp. 203-214.

[7] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware", in: Proceedings of ACM PODS, 2001, pp. 102-113.

[8] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries", in: Proceedings of VLDB, 2002, pp. 275-286.

[9] M. Morse, J. M. Patel, and H. V. Jagadish, "Efficient skyline computation over low-cardinality domains", in: Proceedings of VLDB, 2007, pp. 267-278.

[10] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems", ACM Transactions on Database Systems, vol. 30(1), pp. 41-82, March 2005.

[11] Y. Tao, X. Xiao, and J. Pei, "Subsky: efficient computation of skylines in subspaces", in: Proceedings of ICDE, 2006, pp. 65-65.

[12] T. Xia, D. Zhang, and Y. Tao, "On skylining with flexible dominance relation", in: Proceedings of ICDE, 2008, pp. 1397-1399.

[13] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient Progressive Skyline Computation", in: Proceedings of VLDB, 2001, pp. 301-310.

[14] W. T. Balke, U. Guntzer, and J. X. Zheng, "Efficient distributed skylining for web information systems", in: Proceedings of EDBT, 2004, pp. 256-273.

[15] S. Kapoor, "Dynamic Maintenance of Maxima of 2-d Point Sets", in: SIAM Journal on Computing, vol. 29(6), pp. 1858-1877, April 2000.

[16] Y. Tao and D. Papadias, "Maintaining Sliding Window Skylines on Data Streams", in: IEEE Transactions on Knowledge and Data Engineering, vol. 18(3), pp. 377-391, March 2006.

[17] Z. Huang, H. Lu, B. Ooi, and A. Tung, "Continuous skyline queries for moving objects", in: IEEE Transactions on Knowledge and Data Engineering, vol. 18(12), pp. 1645-1658, Dec. 2006.

[18] K. C. K. Lee, B. Zheng, H. Li, and W. C. Lee, "Approaching the Skyline in Z Order", in: Proceedings of VLDB, 2007, pp. 279-290.

[19] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, "SKYPEER: Efficient Subspace Skyline Computation over Distributed Data", in: Proceedings of ICDE, 2007, pp. 416-425.

[20] K. Fotiadou and E. Pitoura, "BITPEER: Continuous Subspace Skyline Computation with Distributed Bitmap Indexes", in: Proceedings of DaAMaP, 2008, pp. 35-42.

[21] K. Deng, X. Zhou, and H. T. Shen, "Multi-source Skyline Query Processing in Road Networks", in: Proceedings of ICDE, 2007, pp. 796-805.

[22] M. Sharifzadeh and C. Shahabi, "The Spatial Skyline Query", in: Proceedings of VLDB, 2006, pp. 751-762.

[23] C. Li, B. C. Ooi, A-K. H. Tung, and S. Wang, "DADA: A Data Cube for Dominant Relationship Analysis", in: Proceedings of ACM SIGMOD, 2006, pp. 659-670.

[24] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic Skylines on Uncertain Data", in: Proceedings of VLDB, 2007, pp. 15-26.

[25] E. Dellis and B. Seeger, "Efficient Computation of Reverse Skyline Queries", in: Proceedings of VLDB, 2007, pp. 291-302.