IPSJ Transactions on Databases Vol. 3 No. 2 48–60 (June 2010)

Regular Paper

Continuous *k*-Dominant Skyline Computation by Using Divide and Conquer Strategy

Md. Anisuzzaman Siddique[†1]
and Yasuhiko Morimoto[†1]

Skyline objects in a database are objects that are not dominated by any other object in the database. Skyline queries retrieve a set of skyline objects so that the user can choose promising objects from them and make further inquiries. Therefore, such skyline queries are important for several database applications. However, a skyline query often retrieves too many objects to analyze intensively especially for high-dimensional dataset. Recently, k-dominant skyline queries have been introduced, which can reduce the number of retrieved objects by relaxing the definition of the dominance. On the other hand, the maintenance of k-dominant skyline objects under continuous updates is much more difficult compared to conventional skyline objects. This paper addresses the problem of efficient maintenance of k-dominant skyline objects of frequently updated database. We propose an algorithm based on divide and conquer strategy for maintaining k-dominant skyline objects. Intensive experiments using real and synthetic datasets demonstrated that our method is efficient and scalable.

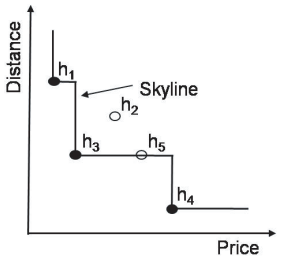
1. Introduction

Skyline objects in a database are objects that are not dominated by any other object in the database. Skyline queries retrieve a set of skyline objects so that the user can choose promising objects from them and make further inquiries. Therefore, such skyline query functions are important for several database applications, including customer information systems, decision support, data visualization, and so forth.

Given an n-dimensional database DB, an object O_i is said to be in skyline of DB if there is no other object O_j $(i \neq j)$ in DB such that O_j is better than O_i in all dimensions. If there exist such O_j, then we say that O_i is dominated by O_j or O_j dominates O_i. **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 and price, for online booking. A user 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 Fig. 1 (b)). Hence, computing skyline from a database is helpful for users' decision-making. A number of efficient algorithms for computing skyline objects have been reported in the literature [1)–5)].

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

Fig. 1 Skyline example.

However, a skyline query often retrieves too many objects to analyze intensively especially for high-dimensional dataset. To reduce the number of returned objects and to find more important and meaningful objects, Chan, et al. considered k-dominant skyline query [6)]. 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 object. Therefore, conventional skyline objects are n-dominant objects.

Chan, et al. proposed three algorithms for k-dominant skyline query computation. These three algorithms are executed on static datasets and do not consider online maintenance of the k-dominant skyline objects to reflect upcoming updates of the database. Hence, each time when an update (insertion/deletion) occurs, k-dominant skyline objects have to be re-computed from scratch.

Towards an efficient continuous skyline computation the following challenge must be addressed: an effective incremental k-dominant skyline query result update mechanism that is needed provides a fast response time of reporting the

†1 Graduate School of Engineering, Hiroshima University

Table 1 Symbolic data set.

Object	D_1	D_2	D_3	D_4	D_5	D_6
O_1	7	7	4	5	4	2
O_2	7	6	5	6	3	2
O_3	1	2	3	4	5	5
O_4	3	3	4	5	1	2
O_5	4	4	1	2	6	3
O_6	2	5	2	3	3	4
O_7	3	2	3	2	6	5
O_8	1	1	6	3	2	1
O_9	5	6	4	3	3	1
O_{10}	2	6	4	5	1	3
O_{11}	1	2	6	5	4	4
O_{12}	2	3	6	5	8	4

current query results. However, the maintenance of k-dominant skyline objects under continuous updates is much difficult. This is because, sometimes, we have to recompute the entire k-dominant skyline objects from scratch if some kinds of update occur. We, therefore, consider an efficient method to compute and to maintain the k-dominant skyline in the presence of database updates.

1.1 Motivating Example

Assume we have a symbolic dataset as listed in **Table 1**. In the table, each object is represented as a tuple containing six attributes or dimensions from D_1 to D_6. Without loss of generality, we assume smaller value is better in each dimension. Conventional skyline query for this database returns eight objects: O_3 to O_{10}. Objects O_1 and O_2 are not in skyline because they are dominated by O_4. Similarly, objects O_{11} and O_{12} are not in skyline because they are dominated by O_8. If we look at these eight skyline objects more closely, we can find that not all objects are significant in a sense. For example, compare with O_3, O_7 is survived only by its value of D_4. O_6 is in skyline because no other object fails to dominate it in all dimensions, even though it does not have any maximal feature values. In such a situation, the user naturally wants to eliminate the skyline objects by using selective criterion.

The k-dominant skyline query can control the selectivity by changing k. Consider the case where $k = 5$. Now the 5-dominant skyline query for this database returns the three objects: O_4, O_5, and O_8. Objects O_1, O_2, O_3, O_6, O_9, O_{11}, and O_{12} are not in 5-dominant skyline because they are 5-dominated by O_8. Object O_7 and O_{10} are not in 5-dominant skyline because they are respectively 5-dominated by O_3 and O_4 . Similarly, 4-dominant skyline query (i.e., $k = 4$) returns only one object, O_8 is in 4-dominant skyline. If we decrease the value of k by one, then the 3-dominant skyline will retrieve empty result.

Though we can eliminate less important skyline objects by using k-dominant skyline, the maintenance of k-dominant skyline for an update is much more difficult due to the 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, $\{O_4, O_5, O_8\}$ are in 5-dominant skyline. If we insert a new object $O_{new} = (2, 2, 3, 2, 7, 6)$ into the dataset, we can compare O_{new} with the three 5-dominant skyline objects 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_3. Like this example, for each insertion in the database, 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.

Again, if an update is 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_3, this deletion will "promote" O_7 as a 5-dominant skyline object.

In this paper, we study the problem of k-dominant skyline computation and its maintenance. We make the following contributions. First, we propose an efficient algorithm to compute k-dominant skyline objects. Second, we propose a method for handling updates. Last, we present a comprehensive performance study using both real and synthetic datasets to verify the effectiveness and the efficiency of our methods.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the notions and properties of Divide and Conquer

based Sort-Filtering k-dominant skyline computation. We provide detailed examples and analysis of Divide and Conquer based Sort-Filtering k-dominant skyline computation in Section 4. We experimentally evaluate our algorithm in Section 5 by comparing with other existing algorithms under a variety of settings. Finally, Section 6 concludes the paper.

2. 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.

2.1 Skyline Query Processing

Borzsonyi, et al. first introduce the skyline operator over large databases and proposed three algorithms: *Block-Nested-Loops* (BNL), *Divide-and-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-Filter-Skyline* (SFS) as a variant of BNL [4]. 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 [7]. Both of them require preprocessing. In the Bitmap approach, every dimension value of a point is represented by a few bits. By applying bit-wise *and* operation on these vectors, a given point can be checked if it is in the skyline without referring to other points. 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 [3]. 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-and-Bound Skyline* (BBS), proposed by Papadias, et al., which is a progressive algorithm based on the best-first nearest neighbor (BF-NN) algorithm [5]. Instead of searching for nearest neighbor repeatedly, it directly prunes using the R*-tree structure. Tao and Papadias studied sliding window skylines, focusing on data streaming environments [8].

2.2 k-Dominant Skyline Query Processing

Chan, et al. introduce k-dominant skyline query [6]. 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 [9].

Another study on computing k-dominant skyline is *k-ZSearch* proposed by Lee, et al. [10]. They introduced a concept called filter-and-reexamine approach. In the filtering phase, it remove all k-dominated 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. However, in a dynamic dataset insertions and deletions are very frequent and the above schemes [6),10)] are not efficient to solve the frequent update problem. Because they need to recompute k-dominant skyline result from scratch. To overcome frequent update problem our proposed divide and conquer based sort-filtering method localizes the side effect of updates and recomputes the k-dominant skyline query result efficiently.

A recent algorithm called CoSMuQ can compute continuous k-dominant skyline [11)]. It divides the whole dataset space into pairs of subspaces and maintains grids for each subspace to compute subspace skyline. The k-dominant skyline is obtained by the union of the subspaces skylines. However, CoSMuQ suffers from maintenance problem. For any insertion or deletion, it needs to update all subspaces to compute k-dominant skyline, i.e., it fails to localize the side effect of updates. Moreover, in high dimensional space, it needs to consider huge number of subspaces. However, our proposed method do not divides the data space into subspaces, it divides the large cardinality dataset into smaller segments. While an update occur in any segment, it may or may not changes k-dominant skyline result of that segment. If the update has effect on the segmented k-dominant skyline result, then our method only updates the result of that segment and do not need to update other segments results to recompute k-dominant skyline.

Recently, more aspects of skyline computation have been explored. Chan, et al. introduce the concept of *skyline frequency* to facilitate skyline retrieval in high-dimensional spaces [12)]. Tao, et al. discuss skyline queries in arbitrary subspaces [13)]. There exist more work addressing *spatial skyline* [14),15)], skylines on partially-ordered attributes [16)], and *reverse skyline* [17)].

3. Preliminaries

This section discusses the k-dominant skyline problems and associated properties.

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 $O_i.D_j$ to denote the j-th dimension value of O_i.

3.1 k-Dominance

An object O_i is said to *dominate* another object O_j, which we denote as $O_i \leq O_j$, 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 $O_i \leq_k O_j$, if $O_i.D_s \leq O_j.D_s$ in k dimensions among n dimensions and $O_i.D_t < O_j.D_t$ 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.

3.2 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 $Sky_k(DB)$. Note that objects that have k-domination power must be k-dominant skyline objects but not vice versa.

3.3 A Priori Property

A k-dominant object has the following a priori property.

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 object O_i is not $(k-1)$-dominated by any other object in DB. It implies that O_i is not k-dominated by any other object. Therefore, we can say O_i is a k-dominant 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 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, O_2, O_{11}, and O_{12} of Table 1 are not in $Sky_6(DB)$ because they are 6-dominated by O_4 and O_8. 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, O_3, O_6, O_7, O_9, and O_{10} are 5-dominated objects. Therefore, we can prune all of those nine objects in 4-dominant query computation. Thus, by decreasing k, more dominated objects can be pruned away.

4. *k*-Dominant Skyline Algorithm

In this section, we present our algorithm for computing k-dominant skyline objects in n-dimensional database DB and how to maintain the result when an update occurs. In order to handle updates efficiently, we adopt a divide and conquer strategy in k-dominant skyline computation. First of all, we horizontally partitioned DB into m segments $S_1, S_2, \cdots, S_m$, i.e., $DB = S_1 \cup S_2 \cup \cdots \cup S_m$. We compute intermediate skyline for each segment. We denote a set of all skyline objects in $S_l (1 \leq l \leq m)$ as $Sky_n(S_l)$. We conquer those m intermediate results, i.e., $Sky_n(S_l)(1 \leq l \leq m)$ to compute $Sky_k(DB)$.

We use a filter based algorithm to compute $Sky_k(DB)$, efficiently, it consist of two parts. One is sorting by domination power, which is in Section 4.1, and another is k-dominant skyline calculation, which is in Section 4.2. In Section 4.3, we discuss the k-dominant skyline maintenance problem when an update occurs in the dataset.

4.1 Domination Power Calculation

Objects whose sum of all their dimension values is small are likely to dominate other objects, while objects whose sum is large are likely to be dominated. Chomicki, et al. [4] proposed an efficient algorithm that sort the whole tuples (objects) in ascending order of the sum of all dimension values. By using the ordered tuples, we can eliminate some of non-skyline objects easily. Chan, et al. used this ordered tuples in their OSA algorithm for k-dominant query [6]. However, this ordered tuples is not effective for k-dominant query computation especially when values of each attribute is not normalized. For example, assume $O_i = (3, 2, 3)$ and $O_j = (7, 1, 2)$ are two objects in 3D space. Although O_i has smaller sum than O_j, O_i does not 2-dominant of O_j. Instead, O_i is 2-dominated by O_j.

In order to prune unnecessary objects efficiently in the k-dominant skyline computation, we compute *domination power* of each object. An object is said to have δ-*domination power* if there are δ minimal values in which it is better or equal to all other objects of DB. We sort objects in descending order by their values of domination power (δ). If more than one objects have same domination power then sort those objects in ascending order of the sum value. This order reflects how likely to k-dominate other objects. Higher objects in the sorted sequence are likely to dominate other objects. Thus this preprocessing helps to reduce the computational cost of k-dominant skyline. However, sometimes, lower object can k-dominate higher object.

Table 2 Segmented dataset1.

Object	D_1	D_2	D_3	D_4	D_5	D_6	Sum
$\underline{O_4}$	3	3	4	5	1	2	18
$\underline{O_3}$	1	2	3	4	5	5	20
O_2	7	6	5	6	3	2	29
O_1	7	7	4	5	4	2	29

Table 3 Segmented dataset2.

Object	D_1	D_2	D_3	D_4	D_5	D_6	Sum
$\underline{O_8}$	1	1	6	3	2	1	14
$\underline{O_6}$	2	5	2	3	3	4	19
$\underline{O_5}$	4	4	1	2	6	3	20
$\underline{O_7}$	3	2	3	2	6	5	21

4.2 *k*-Dominant Skyline Calculation

For each segment $S_l (1 \leq l \leq m)$, we first compute $Sky_n(S_l)$, which is conventional skyline objects in S_l, by SFS method proposed in Ref. 4). After computing all $Sky_n(S_l)$, results are send to the coordinator. The coordinator computes $Sky_k(DB)$ from m intermediate results.

Assume there is a database as in Table 1 and we divide the database into three segments, say S_1, S_2, and S_3. The segments sorted by sum are shown in **Tables 2**, **3**, and **4**. In the example, the underlined objects in Tables 2, 3, and 4 are $Sky_6(S_1) = \{O_4, O_3\}$, $Sky_6(S_2) = \{O_8, O_6, O_5, O_7\}$, and $Sky_6(S_3) = \{O_{10}, O_9, O_{11}\}$, respectively. Remember that we can avoid segmentation and $Sky_6(S_l)$ computation procedure if no update is necessary.

Next, we compute k-dominant skyline of DB, i.e., $Sky_k(DB)$. We first make an union set of $Sky_n(S_l)$ $(1 \leq l \leq m)$ and sort the union set by domination power

Table 4 Segmented dataset3.

Object	D_1	D_2	D_3	D_4	D_5	D_6	Sum
O_{10}	2	6	4	5	1	3	21
O_9	5	6	4	3	3	1	22
O_{11}	1	2	6	5	4	4	22
O_{12}	2	3	6	5	8	4	28

Table 5 Union of intermediate skyline, DB'.

Object	D_1	D_2	D_3	D_4	D_5	D_6	DP	Sum
O_8	1	1	6	3	2	1	3	14
O_5	4	4	1	2	6	3	2	20
O_4	3	3	4	5	1	2	1	18
O_3	1	2	3	4	5	5	1	20
O_7	3	2	3	2	6	5	1	21
O_{10}	2	6	4	5	1	3	1	21
O_9	5	6	4	3	3	1	1	22
O_{11}	1	2	6	5	4	4	1	22
O_6	2	5	2	3	3	4	0	19

and the sum value. Let DB' be the sorted object sequence of the union set. Note that an object in $Sky_k(DB)$ must be in this union set. Thus, $Sky_k(DB')$ and $Sky_k(DB)$ represent the same k-dominant objects set. Moreover, the sorted sequence roughly reflects the importance of objects and our method can progressively output the k-dominant objects based on the sequence.

Table 5 is the example of sorted database DB'. In the sorted database, object O_8 has the highest domination power 3. Note that object O_8 dominates all objects lie below it in three attributes D_1, D_2, and D_6.

Algorithm 1 shows our Sort-Filtering algorithm with domination power approach. In the first scan of DB' (steps 3 to 14), a set of candidate k-dominant skyline objects, $Sky_k(DB')$ is computed progressively by comparing each object $O \in DB'$ against the computed objects in $Sky_k(DB')$. If an object is k-dominated then it is removed from $Sky_k(DB')$. During this scan we also removed some non-skyline objects to non-$Sky_n(DB')$, if they are all dominated by $Sky_k(DB')$ objects. To eliminate the false positives produced by the first scan, a second scan of DB' (step 15 to 18) is necessary. During the second scan we can exclude all $Sky_k(DB')$ as well as non-skyline objects for further k-dominant checking.

Algorithm 1 Sort-Filter (DB', k)

```
1. Sort DB' by domination power and sum
2. Initialize Sky_k(DB') = ∅ and non-Sky_n(DB') = ∅
3.   for each object O ∈ DB' do
4.     initialize isDominant = true
5.     for each object O' ∈ Sky_k(DB') do
6.       if (O' n-dominates O) then
7.       add O in non-Sky_n(DB')
8.       if (O' k-dominates O) then
9.         isDominant = false
10.        break
11.      if (O k-dominates O') then
12.        remove O' from Sky_k(DB')
13.    if (isDominant) then
14.      add O in Sky_k(DB')
15.  for each object O ∈ DB' do
16.    for each object O' ∈ Sky_k(DB'), do
17.      if ((O k-dominates O') and ((O' ≠ O) or (O' ∉ non-Sky_n(DB')))) then
18.        remove O' from Sky_k(DB')
19.  return Sky_k(DB')
```

From Table 5, at the end of first scan we can see that $Sky_5(DB') = \{O_8, O_5, O_4, O_7\}$ and non-$Sky_6(DB') = \{O_{11}\}$. After the completion of the second scan, we obtain $Sky_5(DB') = \{O_8, O_5, O_4\}$ as a 5-dominant result. Because objects O_7 is 5-dominated by non-k-dominant object O_3. Next, using $Sky_5(DB')$ objects, we get $Sky_4(DB') = \{O_8\}$. Notice that without applying δ-domination power sort 5-dominant skyline computation for DB' takes 19 comparisons. However, if we apply δ-domination power sort then it takes 15 comparisons.

4.3 *k*-Dominant Skyline Maintenance

The proposed k-dominant skyline algorithm described above computes the $Sky_k(DB)$ from scratch. In this subsection, we discuss the maintenance problem of $Sky_k(DB)$ after an update (insertion/deletion) is occurred in a segment S_l for $l \in [1, m]$. Maintenance of $Sky_n(S_l)$ consists of two operations: (i) checking whether an updated object is dominated by the current $Sky_n(S_l)$ and (ii) retrieving the objects that are dominated by the recently deleted skyline object, since only these objects are candidate for entering into $Sky_n(S_l)$. We can perform the second task efficiently by utilizing the following observation.

Algorithm 2 $Sky_n(S_l)$ **Maintenance framework**

```
1. Input: S_l, skyline Sky_n(S_l) ⊆ S_l, inserting object O_I, deleting object O_D
2.    if O_I not dominated by current Sky_n(S_l) then
3.      Add O_I in Sky_n(S_l) and remove any dominated objects from Sky_n(S_l)
4.    end if
5.    Insert O_I in S_l
6.    if O_D is in current Sky_n(S_l) then
7.      Find objects in Sky_n(S_l) dominated by O_D and use them to mend Sky_n(S_l)
8.    end if
9.    Delete O_D from S_l
```

Lemma 1: Assume $O_1.D_s \leq O_2.D_s$ for all dimensions D_s $(s = 1, \cdots, n)$ for $O_1, O_2 \in S_l (1 \leq l \leq m)$. If O_1 is not deleted from S_l, O_2 will never be in the skyline of S_l.

Proof: Since O_2 is n-dominated by O_1, it is dominated by O_1 until deletion of O_1. Therefore, while O_2 is in the S_l, there will be at least one object, namely O_1, that dominates it and consequently cannot become a $Sky_n(S_l)$. $\diamondsuit$

The lemma implies that a significant number of objects in each S_l is irrelevant for the skyline maintenance task, since they can never become part of $Sky_n(S_l)$. For example in Table 2, the skyline objects for this segment, S_1 is $\{O_4, O_3\}$. Objects O_2 and O_1 are not in skyline, because both are n-dominated by O_4. Thus, O_2 and O_1 are irrelevant objects for maintaining skyline of S_1 while O_4 is in S_1. Algorithm 2 summarizes the high level strategy that is employed in order to keep $Sky_n(S_l)$ up to date. Notice that $Sky_n(S_l)$ is a subset of S_l.

Thanks to the divide and conquer strategy, we can localize the side effect of an update and can compute $Sky_k(DB)$ efficiently.

Lemma 2: Assume $O_1.D_s \leq O_2.D_s$ in k dimensions among n dimensions for $O_1, O_2 \in S_l (1 \leq l \leq m)$. If O_1 is not deleted from S_l, 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 $O_2 \in S_l$, 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 $Sky_n(S_l)$ 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 $Sky_n(S_l)$ objects.

Assume that an object O is inserted into S_l. We scan objects in $Sky_n(S_l)$ and check k-dominance with O. If $Sky_n(S_l)$ is not modified by the insertion, we don't have to modify $Sky_k(DB)$. Otherwise, we have to update $Sky_k(DB)$ by using modified $Sky_n(S_l)$ as follows:

(1) If no object in $Sky_n(S_l)$ is discarded, which means O is added into $Sky_n(S_l)$ as well as DB', then

a) At first scan objects in $Sky_k(DB)$ to check k-dominance with O. i) If O is k-dominated and there is no change in $Sky_k(DB)$ then the maintenance is completed. ii) If O is k-dominated and change occurs in $Sky_k(DB)$ then update $Sky_k(DB)$.

b) If O is not k-dominated by any object in $Sky_k(DB)$ then check k-dominance with $\{DB' - Sky_k(DB)\}$. i) If O is k-dominated and no change in $Sky_k(DB)$ then the maintenance is completed. ii) If O is k-dominated but change occurs in $Sky_k(DB)$ then update $Sky_k(DB)$.

c) If O is not k-dominated by any object in DB' and no change in $Sky_k(DB)$ then insert O into $Sky_k(DB)$.

d) If O is not k-dominated by any object in DB' but change occurs in $Sky_k(DB)$ then update $Sky_k(DB)$.

(2) If any of $Sky_n(S_l)$ is discarded after the insertion, we recompute $Sky_k(DB)$ from updated $Sky_n(S_l)$.

Current k-dominant skyline objects of DB are $Sky_5(DB) = \{O_4, O_5, O_8\}$, $Sky_4(DB) = \{O_8\}$, and $\{DB' - Sky_k(DB)\} = \{O_3, O_6, O_7, O_9, O_{10}, O_{11}\}$. If we insert object $O_{13} = (6, 6, 6, 6, 6, 6)$ into S_1. By scanning $Sky_n(S_1)$, we can find that the insertion does not modify $Sky_n(S_1)$. Therefore, we immediately complete the maintenance of $Sky_k(DB)$. If we insert $O_{14} = (6, 6, 6, 6, 6, 1)$ into S_1, it becomes a $Sky_n(S_1)$. In this case, we scan each object in $Sky_5(DB) = \{O_4, O_5, O_8\}$ to check k-dominance with O_{14}. After the domination checking, we can find that O_{14} is 5-dominated by O_8, then we add it into $\{DB' - Sky_k(DB)\} = \{O_3, O_6, O_7, O_9, O_{10}, O_{11}, O_{14}\}$ and the maintenance is completed. If we insert $O_{15} = (2, 2, 3, 4, 3, 2)$ into S_1, it also becomes a $Sky_n(S_1)$. After domination checking with $Sky_5(DB) = \{O_4, O_5, O_8\}$, O_{15} is 5-dominated by O_8 and also becomes the 5-dominant of O_4, then add O_4 and O_{15} into $\{DB' - Sky_k(DB)\} =$

$\{O_3, O_4, O_6, O_7, O_9, O_{10}, O_{11}, O_{14}, O_{15}\}$. Now that k-dominant skyline becomes $Sky_5(DB) = \{O_8, O_5\}$ and $Sky_4(DB) = \{O_8\}$. If we insert $O_{16} = (3, 3, 2, 5, 1, 3)$ into S_1, then it is in $Sky_n(S_1)$. After domination checking with $Sky_5(DB) = \{O_5, O_8\}$, O_{16} is not 5-dominated and can't becomes the 5-dominant of any object in $Sky_5(DB)$, then we check k-dominance of O_{16} with $\{DB' - Sky_k(DB)\}$. After the checking, we find that it is 5-dominated by O_4, then we add it into $\{DB' - Sky_k(DB)\} = \{O_3, O_4, O_6, O_7, O_9, O_{10}, O_{11}, O_{14}, O_{15}, O_{16}\}$ and the maintenance is completed. Next, if we insert $O_{17} = (3, 2, 3, 2, 6, 2)$ into S_1, then it is in $Sky_n(S_1)$. After domination checking with $Sky_5(DB) = \{O_5, O_8\}$, O_{17} is not 5-dominated by $Sky_5(DB)$ but becomes the 5-dominant of object O_5, then we check k-dominance of O_{17} with $\{DB' - Sky_k(DB)\}$. We find that it is 5-dominated by O_{15}, add both objects into $\{DB' - Sky_k(DB)\} = \{O_3, O_4, O_5, O_6, O_7, O_9, O_{10}, O_{11}, O_{14}, O_{15}, O_{16}, O_{17}\}$ and now the $Sky_5(DB)$ becomes $\{O_8\}$. If we insert $O_{18} = (2, 2, 2, 2, 3, 3)$ into S_1, then it is in $Sky_n(S_1)$. After domination checking with $Sky_5(DB) = \{O_8\}$ as well as $\{DB' - Sky_k(DB)\}$, O_{18} is not 5-dominated. Then we add it into $Sky_5(DB)$ and $Sky_5(DB)$ becomes $\{O_8, O_{18}\}$. Next, if we insert $O_{19} = (1, 2, 2, 6, 1, 3)$ into S_1, it becomes a $Sky_n(S_1)$. After domination checking with $Sky_5(DB) = \{O_8, O_{18}\}$ as well as $\{DB' - Sky_k(DB)\}$, O_{19} is not 5-dominated, but becomes 5-dominant of O_{18}. Then we have $Sky_5(DB) = \{O_8, O_{19}\}$ and $\{DB' - Sky_k(DB)\} = \{O_3, O_4, O_5, O_6, O_7, O_9, O_{10}, O_{11}, O_{14}, O_{15}, O_{16}, O_{17}, O_{18}\}$. If we insert $O_{20} = (1, 1, 1, 1, 1, 1)$ into S_1, then discarding all other objects it becomes the member of $Sky_n(S_1)$. We recompute $Sky_k(DB)$ from the union set of $Sky_n(S_1) = \{O_{20}\}$, $Sky_n(S_2) = \{O_8, O_6, O_5, O_7\}$, and $Sky_n(S_3) = \{O_{10}, O_9, O_{11}\}$. We obtain k-dominant skyline as $Sky_5(DB) = \{O_{20}\}$ and $Sky_4(DB) = \{O_{20}\}$.

Next, assume that an object O is deleted from S_l. If the deleted object is not in $Sky_n(S_l)$, we do not have to recompute $Sky_k(DB)$. Otherwise, we have to recompute $Sky_k(DB)$ from the union set of m intermediate results, $Sky_n(S_l)$.

Consider the running example again. Assume $Sky_n(S_1) = \{O_{20}\}$, $Sky_5(S_2) = \{O_6, O_5, O_8, O_7\}$, $Sky_5(S_3) = \{O_{10}, O_9\}$, and k-dominant skyline as $Sky_5(DB) = \{O_{20}\}$, $Sky_4(DB) = \{O_{20}\}$ in the current database. If we delete O_1 or O_2 from S_1, we do not have to recompute $Sky_k(DB)$ and the maintenance is completed. Again, if we delete O_{20} from S_1, $Sky_n(S_1)$ is modified from $\{O_{20}\}$ to $\{O_{18}, O_{19}, O_{15}, O_{16}, O_{17}, O_4, O_3, O_{14}\}$. Now, we have to recompute $Sky_k(DB)$. After the recompilation, $Sky_k(DB)$ becomes $Sky_5(DB) = \{O_8, O_{19}\}$ and $Sky_4(DB) = \{O_8\}$.

5. Performance Evaluation

We conduct a series of experiments to evaluate the effectiveness and efficiency of the proposed method. In lack of techniques dealing directly with the problem of maintaining k-dominant skyline in this paper, we compare our method against TSA, which was the most efficient k-dominant skyline search algorithm proposed in Ref. 6). To handle updates, we adapt a variant of the TSA called ATSA (Adaptive Two-Scan Algorithm). Let n be the total number of objects in DB. ATSA takes $O(n^2)$ 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 $O(n)$. If an object is deleted from the DB, ATSA has to recomputed 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 $O(n^2)$ time.

Though the time complexity of our proposed method is substantially the same, we can drastically reduce comparisons for k-dominant skyline computation by the segmentation and localization. Let m be the total number of segment. The proposed method takes $O(n/m)$ to compute each local skyline objects. Let n' be the total number of objects in DB', where DB' is the union set of all local skyline objects. We compute k-dominant skyline objects from n' objects, which takes $O(n'^2)$. We can expect that n' is much smaller than n. For each insertion, proposed method takes $O(n/m)$ time to perform k-domination check, if the inserted object $\notin$ the local skyline. Otherwise, it takes $O(n')$ time to perform k-domination check, if the inserted object is in the local skyline. For each deletion, if the deleted object is not in the local skyline then the proposed method takes $O(1)$. Otherwise, it takes $O(n'^2)$. Results of all experiments support our claim that 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 2 GHz CPU and 3 GB main

memory. All experiments were coded in Java J2SE V6.0. Each experiment is repeated three times and the average result is considered for performance evaluation.

5.1 Performance on Synthetic Datasets

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

Anti-Correlated: an anti-correlated database 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 database is typically quite large.

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

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

The generation of the synthetic datasets is controlled by three parameters, n, "Size", and "Dist", where "Size" is the total number of objects in the dataset and "Dist" can be the any of the three distributions. In addition, we have generated smaller synthetic datasets for all insertion experiments. For example to conduct insertion experiment on 100 k synthetic dataset, we have also generated additional 10 k dataset. As for deletion experiments, we choose the deleted objects randomly from the experimental dataset. The number of insertions and deletions are equal for any update experiments. Unless otherwise stated, we use the following setting in our study: data cardinality to 100 k, m to 10, and the data distribution to anti-correlated.

Some of updates change the result of k-dominant skyline query and some of updates do not. In order to grasp the probability, we conduct an experiment to count how many updates change the result of k-dominant skyline query. **Table 6** shows the result for all type datasets. The second row of Table 6, represents that among 1,000 insertions only 270 insertions have the effect on the k-dominant skyline result for anti-correlated dataset. For this experiment, we set data cardinality to 100 k, m to 10, n to 7, and k to 6.

Table 6 Number of changes in processing insertion & deletion.

#Update	Anti-Correlated		Correlated		Independent	
	#Changes by Ins.	#Changes by Del.	#Changes by Ins.	#Changes by Del.	#Changes by Ins.	#Changes by Del.
1 k	113	132	42	56	78	83
2 k	270	241	57	63	150	178
3 k	371	415	69	87	230	281
4 k	483	574	94	118	287	370
5 k	608	669	141	170	363	397

Table 7 Comparisons reduction.

Data Size(k)	Anti-Correlated			Correlated			Independent		
	#Comp. by ATSA(k)	#Comp. by DCSA(k)	RR (%)	#Comp. by ATSA(k)	#Comp. by DCSA(k)	RR (%)	#Comp. by ATSA(k)	#Comp. by DCSA(k)	RR (%)
100	15,276	10,540	31	734	431	41	14,255	9,891	31
200	28,920	21,275	26	1,597	884	45	26,015	18,601	29
300	48,394	33,977	30	2,391	1,445	40	41,383	29,816	28
400	64,078	49,245	23	3,178	1,962	38	61,518	43,283	30
500	66,480	50,310	24	3,936	2,507	36	69,606	50,991	27

In the following sections, we will refer our proposed Divide and Conquer based Sort-Filtering method, as $DCSF$. To study the potential advantages of δ-domination power on sort by sum, we evaluate comparisons of $DCSF$ against ATSA and compute the reduction rate. The reduction rate is defined as

$$ReductionRate, (RR) = \left(1 - \frac{Comp.byDCSF}{Comp.byATSA}\right) \times 100; \quad (1)$$

where Comp. by DCSF and Comp. by ATSA is the summation of all pairwise comparisons to compute k-dominant skyline by DCSF and ATSA respectively. To be a fair comparison, we do not apply segmentation process to $DCSF$. We set n to 7, k to 6, and vary data cardinality from 100 k to 500 k. From **Table 7** we notice that the number of comparison of DCSF is smaller than that of ATSA and the reduction rate varies from 23% to 45%.

Next, we evaluate performances of $DCSF$ against ATSA and examine the effect of cardinality, dimensionality, and segmentation. In each experiment, we evaluate total time to compute k-dominant skyline. Then, we evaluate the performance

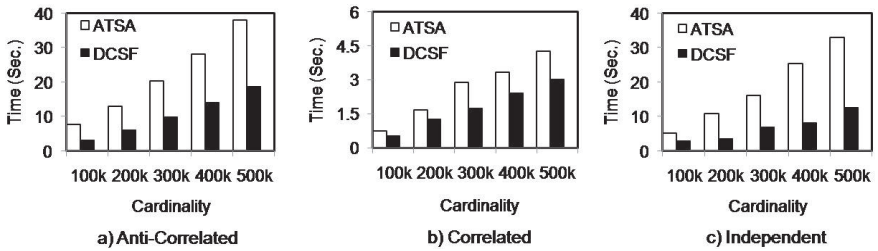


Fig. 2 k-dom. Skyline computation for different datasize.

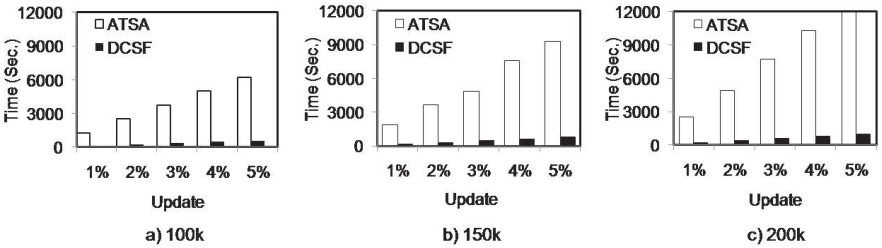


Fig. 3 Update performance for different datasize.

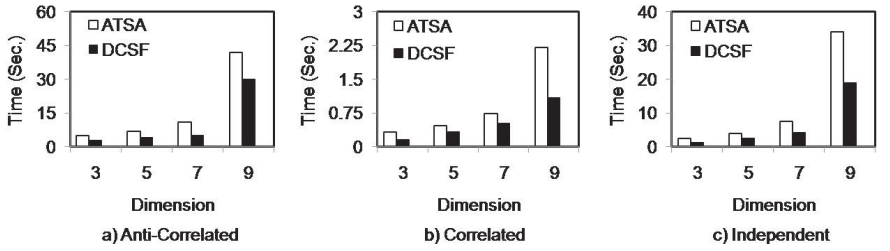


Fig. 4 k-dom. Skyline computation for different dimension.

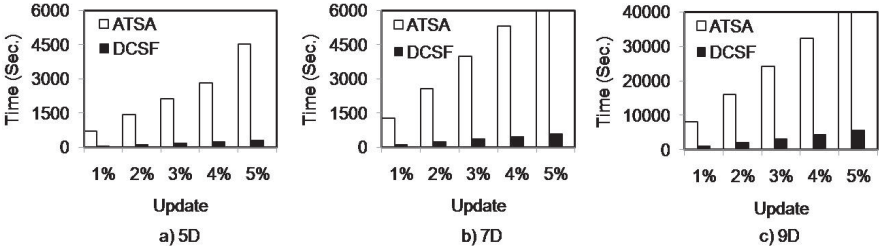


Fig. 5 Update performance for different dimension.

for handling updates. Similar to most of the related work in the literature, we employ the *elapsed time* as the performance metric.

5.1.1 Effect of Cardinality

For this experiment, we vary dataset cardinality ranges from 100 k to 500 k and set the value of n to 7 and k to 6.

Figure 2 (a), (b), and (c) shows the time to compute k-dominant skyline. For all distributions, the time of proposed method is better than ATSA and it is increases if the data cardinality increases.

Figure 3 (a), (b), and (c) shows the time to maintain k-dominant skyline for update ranges from 1% to 5%. In the update experiments, 1% update for 100 k database implies that there are 500 insertions and 500 deletions (total 1 k updates) occurred in the database. The time is the sum of the maintenance time for each update. The result shows that if the update ratio and data cardinality increases the maintenance time also increases. Even though, the time is much smaller than that of ATSA.

5.1.2 Effect of Dimensionality

For this experiment, we vary dataset dimensionality n ranges from 3 to 9 and k from 2 to 8.

Figure 4 (a), (b), and (c) represents the effect of dimensionality. For all distributions, the response time of the proposed method is better than ATSA approach and it is increases if the data dimensionality n increases. This is because by increasing the number of dimensions, the probability that an object dominates another one is reduced significantly.

Figure 5 (a), (b), and (c) shows the update performance. The result shows that if the dimensionality and the update ratio increase the time grows steadily, which is much less than that of ATSA.

5.1.3 Effect of Segmentation

In this experiment, we examine the effect of the number of segments, m. We fix the dimension n to 9, k to 8, and vary m from 5 to 25.

Figure 6 (a), (b), and (c) shows the time to compute k-dominant skyline. The

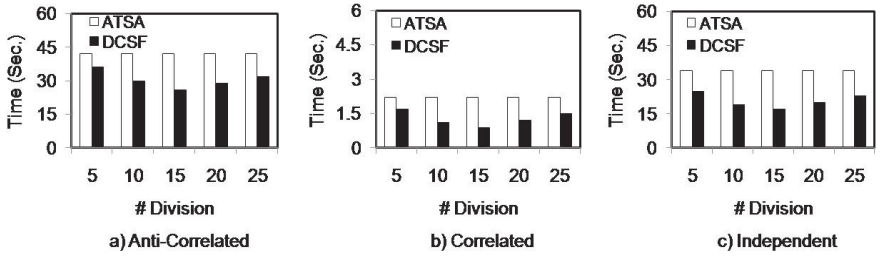


Fig. 6 k-dom. Skyline computation for different segmentation.

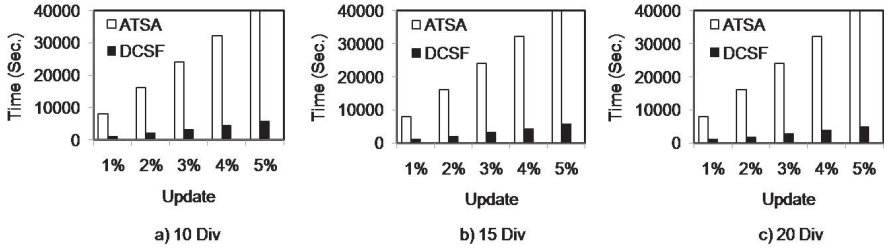


Fig. 7 Update performance for different segmentation.

proposed method is better than ATSA for m in all distributions. However, if the number of segmentation is too small or too large then the performance can become slower. This is because for small m sometimes $DCSF$ fails to take the advantage of update effect localization. Again for large m it can localize the update effect but the k-dominant skyline changes frequency increases as a result the performance becomes slower.

Figure 7 (a), (b), and (c) shows the update performance. The result shows that if the update ratio and data segmentation increases the response time of the proposed method becomes better. This is because for large m the size of each segmented k-dominant skyline decreases, as a result, $DCSF$ can localize side effect of an update into a smaller segment.

5.2 Performance on Real Datasets

To evaluate the performance for real dataset, we used NBA and FUEL datasets. Those are extracted from `www.nba.com` and `www.fueleconomy.gov`, respectively. NBA contains 17 k 13-dimensional data objects, each of which corresponds to the

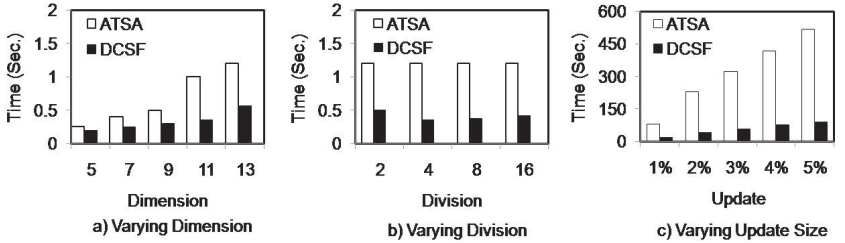


Fig. 8 Experiments on NBA dataset.

statistics of an NBA player's performance in 13 aspects (such as points scored, rebounds, assists, etc). FUEL is 24 k 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.).

5.2.1 Experiments on NBA dataset

We perform four experiments on NBA dataset and fix the default value of m to 8. In the first experiment, we study the effect of dimensionality when n varies from 5 to 13 and k from 4 to 12. **Figure 8** (a) shows the result. NBA dataset exhibits similar result to synthetic dataset, if the number of dimension increases the performance of both algorithms becomes slower. Figure 8 (a) represents that proposed method is faster than ATSA. In the second experiment, we study the effect of segmentation. We vary m from 2 to 16. This experiment consider all aspect of NBA player's and set k to 12. Figure 8 (b) shows that the number of segmentation increases the performance of DCSF increases, but no change in the performance of ATSA because it does not consider any segmentation. In the next experiment, we study the effect of update. For this experiment, we divide the NBA dataset into two set. One set (16 k) is used for k-dominant skyline computation and the other set (1 k) is used for update purpose. The update ratio varies from 1% to 5%. Figure 8 (c) shows the results. There exist significant difference on the performance of both methods and DCSF is 10 times faster than ATSA.

5.2.2 Experiments on FUEL dataset

For FUEL dataset, we perform similar experiments like NBA dataset. We fix the default value for m to 8. For the first experiment, n varies from 3 to 6 and

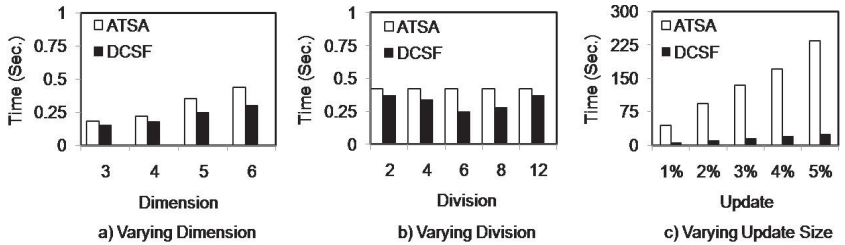


Fig. 9 Experiments on FUEL dataset.

k varies from 2 to 5. Result is shown in **Fig. 9** (a). In the second experiment, m varies from 2 to 12 and we set k to 5. Figure 9 (b) shows the result. In the final experiment, we study the update effect. For this experiment, we divide the NBA dataset into two set. One set (20 k) is used for k-dominant skyline computation and the other set (4 k) is used for update purpose. Figure 9 (c) shows the result. For all of these experiments with FUEL dataset, we obtain similar result like NBA dataset that represents the superiority of DCSF method against ATSA.

6. Conclusion

In this paper, we consider k-dominant skyline query problem and present a method for computing and maintaining the query result. By applying a divide and conquer strategy, we can localize the update effect and recompute the k-dominant skyline query result efficiently. Using real datasets and synthetic datasets, we demonstrate the efficiency and scalability of our proposed method. Intensive experiments show the superiority of the proposed method against the ATSA method.

Though the proposed methods increases the performance of k-dominant skyline computation in most of the cases, the performance of k-dominant skyline computation is not sufficiently small if the number of segmentation is too small or too large. To find proper guide for choosing the right parameter values, such as the number of segmentation, is an open problem.

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

References

1) Xia, T., Zhang, D. and Tao, Y.: On Skylining with Flexible Dominance Relation, *Proc. ICDE*, pp.1397–1399 (2008).
2) Borzsonyi, S., Kossmann, D. and Stocker. K.: The skyline operator, *Proc. ICDE*, pp.421–430 (2001).
3) Kossmann, D., Ramsak, F. and Rost S.: Shooting stars in the sky: An online algorithm for skyline queries, *Proc. VLDB*, pp.275–286 (2002).
4) Chomicki, J., Godfrey, P., Gryz, J. and Liang, D.: Skyline with Presorting, *Proc. ICDE*, pp.717–719 (2003).
5) Papadias, D., Tao, Y., Fu, G. and Seeger, B.: Progressive skyline computation in database systems, *ACM Trans. Database Syst.*, Vol.30, No.1, pp.41–82 (2005).
6) Chan, C.Y., Jagadish, H.V., Tan, K.-L., Tung, A.-K.H. and Zhang, Z.: Finding k-Dominant Skyline in High Dimensional Space, *Proc. ACM SIGMOD*, pp.503–514 (2006).
7) Tan, K.-L., Eng, P.-K. and Ooi, B.C.: Efficient Progressive Skyline Computation, *Proc. VLDB*, pp.301–310 (2001).
8) Tao, Y. and Papadias, D.: Maintaining Sliding Window Skylines on Data Streams, *IEEE Trans. Knowledge and Data Engineering*, Vol.18, No.3, pp.377–391 (2006).
9) Fagin, R., Lotem, A. and Naor, M.: Optimal aggregation algorithms for middleware, *Proc. ACM PODS*, pp.102–113 (2001).
10) Lee, K.C.K., Zheng, B., Li, H. and Lee, W.C.: Approaching the Skyline in Z Order, *Proc. VLDB*, pp.279–290 (2007).
11) Kontaki, M., Papadopoulos, A.N. and Manolopoulos, Y.: Continuous k-Dominant Skyline Computation on Multidimensional Data Streams, *Proc. ACM SAC*, pp.16–20 (2008).
12) Chan, C.Y., Jagadish, H.V., Tan, K.-L., Tung, A.-K.H. and Zhang, Z.: On High Dimensional Skylines, *Proc. EDBT*, pp.478–495 (2006).
13) Tao, Y., Xiao, X. and Pei, J.: Subsky: Efficient Computation of Skylines in Subspaces, *Proc. ICDE*, pp.65–65 (2006).
14) Deng, K., Zhou, X. and Shen, H.T.: Multi-source Skyline Query Processing in Road Networks, *Proc. ICDE*, pp.796–805 (2007).
15) Sharifzadeh, M. and Shahabi, C.: The Spatial Skyline Query, *Proc. VLDB*, pp.751–762 (2006).
16) Chan, C.-Y., Eng, P.-K. and Tan, K.-L.: Stratified Computation of Skylines with Partially-Ordered Domains, *Proc. ACM SIGMOD*, pp.203–214 (2005).
17) Dellis, E. and Seeger, B.: Efficient Computation of Reverse Skyline Queries, *Proc. VLDB*, pp.291–302 (2007).

(Received December 20, 2009)
(Accepted April 6, 2010)

(Editor in Charge: *Miyuki Nakano*)

Md. Anisuzzaman Siddique received his B.Sc. and M.Sc. degrees in Computer Science and Technology from University of Rajshahi (RU), Bangladesh in 2000 and 2002, respectively. Since 2002, he is a faculty member in RU. He is currently a Ph.D. candidate at Hiroshima University. His research interests include skyline evaluation, data mining, and privacy preserving information retrieval.

Yasuhiko Morimoto is an Associate Professor at Hiroshima University. He received his B.E., M.E., and Ph.D. degrees from Hiroshima University in 1989, 1991, and 2002, respectively. From 1991 to 2002, he had been with IBM Tokyo Research Laboratory where he worked for data mining project and multimedia database project. Since 2002, he has been with Hiroshima University. His current research interests include data mining, machine learning, geographic information system, and privacy preserving information retrieval.