Finding k-Dominant Skylines in High Dimensional Space

Chee-Yong Chan[1], H.V. Jagadish[2], Kian-Lee Tan[1], Anthony K.H. Tung[1], Zhenjie Zhang[1]
[1]School of Computing, National University of Singapore
[2]Dept. of Electrical Engineering & Computer Science, University of Michigan
{chancy,tankl,anthony,zhangzh2}@comp.nus.edu.sg jag@eecs.umich.edu

ABSTRACT

Given a d-dimensional data set, a point p dominates another point q if it is better than or equal to q in all dimensions and better than q in at least one dimension. A point is a skyline point if there does not exists any point that can dominate it. Skyline queries, which return skyline points, are useful in many decision making applications.

Unfortunately, as the number of dimensions increases, the chance of one point dominating another point is very low. As such, the number of skyline points become too numerous to offer any interesting insights. To find more important and meaningful skyline points in high dimensional space, we propose a new concept, called *k-dominant skyline* which relaxes the idea of *dominance* to *k-dominance*. A point p is said to *k-dominate* another point q if there are k ($\leq d$) dimensions in which p is better than or equal to q and is better in at least one of these k dimensions. A point that is not k-dominated by any other points is in the k-dominant skyline.

We prove various properties of k-dominant skyline. In particular, because k-dominant skyline points are not transitive, existing skyline algorithms cannot be adapted for k-dominant skyline. We then present several new algorithms for finding k-dominant skyline and its variants. Extensive experiments show that our methods can answer different queries on both synthetic and real data sets efficiently.

1. INTRODUCTION

1.1 Motivation

Given a d-dimensional data set, a point p is said to dominate another point q if it is better than or equal to q in all dimensions and better than q in at least one. A skyline is a subset of points in the data set that are not dominated by any other points. Skyline queries, which return skyline points, are useful in many decision making applications that involve high dimensional data sets. We give two examples here.

Example 1.1 Cell Phone Finder
Consider a person looking for a suitable cell phone at a website[1]. He/she may care about a large number of features including weight, size, talk time, standby time, screen size, screen resolution, data rate and camera quality in order to pick one that suits him/her. There are too many phones at the website for the user to examine them all manually. Computing the skyline over these cell phone features may remove a large number of cell phones whose features are "worse" than those in the skyline, hopefully leaving only a manageable number of candidates for manual evaluation. □

Example 1.2 Movie Rating
Consider a person looking for top-rated movies based on the ratings given by other users[2]. In this case, the rating of each user correspond to a dimension in the data set and given the large number of users, the data set being handled is obviously a high-dimensional one. The skyline of the data set will contain top-rated movies while movies that are consistently ranked worse than others in the data set are pruned away. □

We note that in both the above cases, ranking can be done by providing some preference functions [1, 8] and requesting users to provide some weight assignments for their preferred features or more trusted users in the latter case. However providing such weight assignments for a large number of dimensions is not always easy without any initial knowledge about the data. For example, it is not clear how weight assignments can be provided to aggregate the talk time and camera quality of a phone into one grade. In fact, as stated in the seminal paper on skyline operator [2], the aim of providing the skyline to the user is to help them to determine the weight assignment.

Computing skylines in high dimensional data sets is challenging because of the large number of skyline points [9, 14, 17]. On the movie ranking website, for example, it is nearly impossible to find a movie which is ranked lower than another movie by all the users. Such a blowup in the answer set not only renders the skyline operator worthless (with respect to the desired pruning of candidates), but it also results in high computational complexity for both index and non-index methods as many pairwise comparisons are performed without effecting any pruning.

1.2 k-Dominant Skyline and its Variants

[1]http://www.phonescoop.com/phones/finder.php?m=e
[2]http://movielens.umn.edu/

The primary cause for the skyline set size blow up is the definition of *dominance*, which is rather stringent in that p must be better or equal to q in all dimensions before q can be eliminated from the skyline. Using our movie rating as an example, this means that movie p is only considered better than movie q if and only if p is rated higher or equal to q by all the users. While this is quite possible when there is a small number of users (say 5 users), this is much more unlikely for a larger number of users (say 100) as all it takes is for just one outlier opinion from among the 100 to invalidate the dominance relationship.

It maybe reasonable to consider movie p better than q if say, 98 out of 100 users, consider it to be better. With this intuition, a point p is said to k*-dominate* another point q if there are k ($\leq d$) dimensions in which p is better than or equal to q and is better in at least one of these k dimensions. A point that is not k-dominated by any other points is said to be in the k*-dominant skyline*. Obviously, the conventional skyline is a special case of the k-dominant skyline, where $k = d$. The chance of a point being excluded from a k-dominant skyline is higher than the conventional skyline since more points will be k-dominated than d-dominated. By setting k to an appropriate value, we hope to find a small set of skyline points that are dominant in the sense that they remain in the skyline even with the more relaxed notion of k-dominance. Note that the set of k-dominant skyline points is a subset of the original skyline, which we will henceforth refer to as **free skyline** for clarity.

Unfortunately, algorithms developed for finding the original skyline are not easily adapted for finding k-dominant skyline, except for the obvious case where $k = d$. This is because the transitive property of the original dominance relationship no longer holds, i.e., for any $k < d$ it is possible to have three points p,q,r such that p k-dominates q, q k-dominates r and r k-dominates p, forming a cyclic dominant relationship. Thus, we cannot ignore a point during processing even if it is k-dominated because that particular point might be needed to exclude another point from the skyline through another k-dominant relationship. Existing skyline computation algorithms do not satisfy this requirement.

In view of this, we propose three algorithms in this paper for finding k-dominant skyline. The first is a one-scan algorithm which uses the property that a k-dominant skyline point cannot be worse than any free skyline on more than k dimensions. This algorithm maintains the free skyline points in a buffer during a scan of the data set and uses them to prune away points that are k-dominated. As the whole set of free skyline points can be large, we avoid keeping all of them in a buffer with a two-scan algorithm. In the first scan, a candidate set of dominant skyline points is retrieved by comparing every point with a set of candidates. The second scan verifies whether these points are truly dominant skyline points. This method turns out to be much more efficient than the one-scan method. We provide some theoretical analysis on the reason for its superiority. Finally, we propose an algorithm that is motivated by the rank aggregation algorithm proposed by Fagin et al.[5], which pre-sorts data points separately according to each dimension and then "merges" these ranked lists.

A fundamental question is the choice of value for k. We prove in this paper that it is possible to have an empty k-dominant skyline even for $k = d - 1$ due to the cyclic property. On the other hand, it is still possible to have too many k-dominant skyline points if k is too large. In order to guarantee a small but non-empty set of dominant skyline points, we propose a new type of query, called top-δ dominant skyline query. The aim is to find the smallest k such that there are more than δ k-dominant skyline points. We show that the algorithms proposed for dominant skyline query can be used to answer top-δ query easily.

In some applications, some attributes are more important than other. When the user has an opinion on the relative importance of the different dimensions, we permit the user to express this opinion in the form of relative weights. We extend the k-dominant skyline to the weighted case where d positive weights w_1,...,w_d are assigned to the d dimensions and a point is said to be a weighted w-dominant skyline point if there does not exist any point that can dominate it on a subset of dimensions with their sum-of-weight assignment over w.

1.3 Contributions

Our contributions in this paper are as follows:

1. We introduce a new concept, called k-dominant skyline to alleviate the effect of dimensionality curse on skyline query in high dimensional spaces (Sec. 3).
2. We propose three different algorithms to solve the k-dominant skyline problem (Sec. 4).
3. We modify the k-dominant skyline algorithm to answer the top-δ dominant skyline query (Sec. 5).
4. We extend the concept of k-dominant skyline to weighted space and show how our algorithms work on such spaces (Sec. 6).
5. We show the computational efficiency benefits of these new concepts through a comprehensive experimental study (Sec. 7).

2. RELATED WORK

The basic idea of skyline queries came from some old research topics like contour problem [12], maximum vectors [10] and convex hull [15].

Borzonyi et al. [2] first introduced the skyline operator into relational database systems, and proposed three algorithms: the block nested loops (BNL), divide-and-conquer, and B-tree-based schemes.

Chomicki et al. [4] proposed an algorithm named *Sort-Filter-Skyline* (SFS) as a variant of BNL. SFS requires the dataset to be pre-sorted according to some monotone scoring function. Since the order of the points can guarantee that no point can dominate points before it in the order, the comparisons of tuples are simplified. In [6], Godfrey et al. further improved the efficiency of this method by combining the final pass of the external pre-sort with the first skyline-filter pass.

In [16], Tan et al. proposed two progressive processing algorithms: *Bitmap* and *Index*. In the Bitmap approach, every dimension value of a point *pt* 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 approach partitions the whole data set into some lists, every of which contains points with smallest value on the same dimension among all the dimension values. Every list is further divided into batches according the index value of the points. Within each batch, local skyline is computed by comparing every points with global skyline and is merged into the global skyline after computation.

Kossmann et al. [9] proposed a *Nearest Neighbor* (NN) method to process skyline queries progressively. By indexing the dataset by an R*-tree, the method uses the result of nearest neighbor query to partition the space recursively. The nearest neighbor to the origin in the partitioned region must be part of the skyline. Papadias et al. [13, 14] proposed a new progressive algorithm named *Branch-and-Bound Skyline* (BBS) based on the best-first nearest neighbor (BF-NN) algorithm [7]. Instead of searching for nearest neighbor again and again, it directly prunes using the R*-tree structure.

Yuan et al. [17] proposed two methods that efficiently find the skylines of all subspaces, in bottom-up and top-down manner respectively. Their studies aim to find skyline in a subset of dimensions specified by the users. This is in contrast with our work which directly determines a set of interesting skyline points from the full set of dimensions. In [11], the concept of dominance is generalized to define three types of queries called *dominant relationship queries (DRQs)* to support microeconomic data mining. A data cube is proposed to answer DRQs efficiently. We believe that our work here will eventually be useful for this purpose as well.

To find the top objects under some monotone aggregation function, Fagin proposed three methods, FA, TA and NRA in [5] which are optimal in some cases. The FA algorithm accesses in parallel the sorted lists on every dimension. We can find the top-k points when there is a set of at least k points such that each of them has been seen in each list. The TA algorithm improves the FA algorithm by setting a threshold by the function from all the smallest value that have seen from all the lists. The algorithm stops when the current top-k points all have aggregation value larger than the threshold. The NRA algorithm works with only sorted access. The smallest values seen in all dimension lists are used to calculate the upper bound on the points not seen. We can get top-k result without exact aggregation value, when the lower bounds on the current top-k points are larger than the upper bound on the (k+1)th point.

More recently, there has been work on identifying interesting skylines to address the problem of having too many skyline points particularly when the data is high dimensional. The concept of the *skyline frequency* of a data point was proposed in [3], which measures the number of subspaces that a point is a skyline point. Our proposal of k-dominance offers a different notion of interestingness from skyline frequency. For example, consider two data points p and q on a 3-dimensional data space $S = \{s_1, s_2, s_3\}$, where p is a skyline point in the four subspaces $\{s_1\}$, $\{s_1, s_2\}$, $\{s_1, s_3\}$, and $\{s_1, s_2, s_3\}$; while q is a skyline point in the four subspaces $\{s_1, s_2\}$, $\{s_1, s_3\}$, $\{s_2, s_3\}$, and $\{s_1, s_2, s_3\}$. Note that although both p and q have the same skyline frequency of 4, q is "stronger" in terms of k-dominance since q is a 2-dominant skyline but p is only a 3-dominant skyline. On the other hand, it is also possible for two points to be equally "strong" in terms of k-dominance but differ in their skyline frequencies.

Point	s_1	s_2	s_3	s_4	s_5	s_6
p_1	4	4	4	2	2	2
p_2	2	2	2	4	4	4
p_3	3	3	3	1	3	3
p_4	1	1	1	5	1	1
p_5	2	2	2	3	3	3

Figure 1: Example Data Set, D

3. DEFINITION AND ANALYSIS

3.1 Problem Definition

Given a d-dimensional space $S = \{s_1, s_2, \cdots, s_d\}$, a set of points $D = \{p_1, p_2, \ldots, p_n\}$ is said to be a data set on S if every $p_i \in D$ is a d-dimensional data point on S. We use $p_i.s_j$ to denote the j^{th} dimension value of point p_i. For each dimension s_i, we assume that there exists a total order relationship, denoted by $\succ_i$, on its domain values. Here, $\succ_i$ can be '<' or '>' relationship according to the user's preference. For simplicity and without loss of generality, we assume each $\succ_i$ represents '>' in the rest of this paper.

Definition 3.1. **dominate**
A point p_i is said to dominate another point p_j on S if and only if $\forall\ s_k \in S,\ p_i.s_k \geq p_j.s_k$ and $\exists\ s_t \in S,\ p_i.s_t > p_j.s_t$.

Definition 3.2. **skyline,** $SP(D, S)$
A point p_i is a skyline point on S if and only if there does not exist a point $p_j \neq p_i$ dominating p_i. We use $SP(D, S)$ to denote the set of skyline points in data set D on space S.

Definition 3.3. **k-dominate**
A point p_i is said to k-dominate p_j if and only if $\exists\ S' \subseteq S, |S'| = k,\ \forall\ s_r \in S',\ p_i.s_r \geq p_j.s_r$ and $\exists\ s_t \in S', p_i.s_t > p_j.s_t$.

Definition 3.4. **k-dominant skyline,** $DSP(k, D, S)$
A point p_i is a k-dominant skyline point, if and only if there does not exist any point $p_j \neq p_i$ in the data set that p_j k-dominates p_i. We use $DSP(k, D, S)$ to denote the set of all k-dominant skyline points in D on space S.

When $k = d$, the k-dominant dominance relationship is equivalent to the original dominance relationship defined in definition 3.1. The two main problems that we want to solve are as follows:

Problem 1. *Given a specified k, data set D and dimension space S, find $DSP(k, D, S)$.*

Problem 2. *Given a threshold δ, data set D and dimension space S, let k' be the smallest k which satisfies $|DSP(k, D, S)| \geq \delta$ if $SP(D, S) > \delta$, otherwise $k' = d$. We will use the term* **top-δ dominant skyline** *to refer to $DSP(k', D, S)$. Find $DSP(k', D, S)$.*

Example 3.1 Consider the 6-dimensional data set $D = \{p_1, \cdots, p_5\}$ in Fig. 1. There are four free skyline points in D: p_1, p_2, p_3, and p_4. Among these, only p_1, p_2, and p_3 are also 5-dominant skyline points; p_4 is 5-dominated by p_2. The top-2 dominant skyline points are p_1 and p_2 since they are both 4-dominant skyline points while p_3 is not. We will use this data set as a running example in the rest of this paper. □

point	s_1	s_2	s_3	s_4
p_1	4	4	4	4
p_2	8	3	3	3
p_3	7	8	2	2
p_4	6	7	8	1

Figure 2: Cyclic dominance relationship example

3.2 Analysis

In this section, we illustrate several important properties of dominant skyline points. We first formally show that lowering k will always result in smaller or equal number of k-dominant skyline points.

LEMMA 3.5. *If p_i $k+1$-dominates p_j, then p_i must k-dominate p_j.*

THEOREM 3.6. *A point $p_i \in DSP(k, D, S)$, then $p_i \in DSP(k+1, D, S)$.*

COROLLARY 3.7. $|DSP(k, D, S)| \leq |DSP(k+1, D, S)|$

From the last corollary, we can see that the set of dominant skyline points decreases monotonically with the decrease of the parameter k and thus we are sure that a sufficiently small k can reduce the number of k-dominant skyline to a manageable size for presentation to users. However, one worries that too small a k value might find points which do not make sense semantically to users. For example, if $k < d/2$, this will mean that a point p can k-dominate another point q even if it is only better in fewer than half the dimensions. We alleviate this worry by showing that k-dominant skyline points for $k < (d+2)/2$ are in fact rather easy to interpret.

THEOREM 3.8. *If $k < (d+2)/2$, any pair of k-dominant skyline points must have the same values on at least $d-2k+2$ dimensions.*

The last theorem tells us that when k is too small, either 1) there is only one k-dominant skyline point which has very strong dominant power in that it dominates all the points on many dimensions or 2) there are multiple k-dominant skyline points which are equal on a number of dimensions and have different relative ordering on the remaining dimensions.

We will next motivate the need to look for a top-δ dominant skyline points. The following theorem shows that there may not be any k-dominant skyline point in a data set for any $k < d$.

THEOREM 3.9. *For any $k < d$ ($d \geq 2$) and a d-dimensional space S, there exists a data set D with size $|D| \geq d$ such that $DSP(k, D, S) = \emptyset$.*

Example 3.2 Fig. 2 shows an example data set that exhibits the cyclic dominance relationship when $k = 3$. Specifically, we have p_i 3-dominates p_{i+1}, $\forall\ i \in [1, 3]$, and p_4 in turn 3-dominates p_1. $\square$

4. K-DOMINANT SKYLINE ALGORITHMS

Due to the possibility of cyclic dominance relationships, the existing algorithms for computing free skylines can not be used directly for computing k-dominant skyline points. In this section, we present three novel algorithms, namely, *One-Scan algorithm*, *Two-Scan algorithm*, and *Sorted Retrieval algorithm*, to compute k-dominant skyline points. Each algorithm takes as input a d-dimensional data set D (over a set of dimensions S) and a parameter k, and outputs the set of k-dominant skyline points in D.

4.1 One-Scan Algorithm

Our first approach to compute k-dominant skyline points from an input data set D (over a set of dimensions S) is similar in spirit to the nested-loop approach [2] in that it makes one sequential scan of the data set. The algorithm (shown in Algorithm 1) is based on the following two key properties.

LEMMA 4.1. *Consider a data point $p \in D$ that is not a k-dominant skyline point. Then*

P1. There must exist a free skyline point in D that k-dominates p.

P2. It is possible for p not to be k-dominated by any k-dominant skyline point.

To determine if a point p is k-dominant, property P2 implies that it is not sufficient to use only k-dominant skyline points to compare against p since it is possible for p to be not k-dominant even though it is not k-dominated by any of the k-dominant points. On the other hand, property P1 implies that it is sufficient to compare p against the set of free skyline points (instead of all the points in D) to detect if a point p is k-dominant. Thus, based on Lemma 4.1, our algorithm computes k-dominant skyline points by actually computing the free skyline points in D and using them to eliminate non-k-dominate skyline points. Specifically, two sets of intermediate points are maintained as D is being processed: (1) R stores the set of intermediate k-dominant skyline points in D, and (2) T stores the set of intermediate skyline points in D that are not k-dominant (i.e., not in R). Together, $R \cup T$ gives the set of skyline points in D. Since T is used only for pruning purpose, we can minimize the size of T by storing only the *unique* skyline points; i.e., a new (non-k-dominant) skyline p is added to T only if $p \neq p'$ $\forall\ p' \in T$.

The details of One-Scan algorithm are as follows. For each point p in D, p is first compared against points in T (steps 5 to 11). If a point $p' \in T$ is dominated by p, then p' (which is not a skyline) is removed from T; otherwise, if p' dominates p (i.e., p is not a skyline) or $p = p'$ (i.e., p is not unique), then p can be safely ignored. However, if p is a unique skyline, then p is further compared against the points in R (steps 13 to 22) to check if it is k-dominant. For each $p' \in R$, if p k-dominates p', then p' is moved from R to T; and if p' k-dominates p, then p is not k-dominant. At the end of comparing p against the points in R, p is inserted into R if p is k-dominant; otherwise, p is inserted into T since p is a unique skyline. Once all the points in D have been processed, R contains the set of all k-dominant skyline points in D.

As an additional preprocessing optimization, the points in D can be first sorted in non-ascending order of the sum of all their dimension values (step 1). The purpose of this heuristic, which was first proposed in [4], is to try to maximize the number of skyline points that occur before the non-skyline

Algorithm 1 One-Scan Algorithm (D, S, k)

```
1: sort $D$ in non-ascending order of sum of point's dimension values
2: initialize set of $k$-dominant skyline points $R = \emptyset$
3: initialize set of unique non-k-dominant skyline points $T = \emptyset$
4: for every point $p \in D$ do
5:   initialize isUniqueSkyline = true
6:   for every point $p' \in T$ do
7:     if ($p$ dominates $p'$) then
8:       remove $p'$ from $T$
9:     else if ($p'$ dominates $p$) or ($p = p'$) then
10:      isUniqueSkyline = false
11:      break out of inner for-loop
12:  if (isUniqueSkyline) then
13:    initialize isDominant = true
14:    for every point $p' \in R$ do
15:      if ($p'$ $k$-dominates $p$) then
16:        isDominant = false
17:      if ($p$ $k$-dominates $p'$) then
18:        move $p'$ from $R$ to $T$
19:    if (isDominant) then
20:      insert $p$ into $R$
21:    else
22:      insert $p$ into $T$
23: return $R$
```

points in D so that the non-skyline points are eliminated as early as possible thereby reducing the overhead of maintaining them before they are pruned.

Example 4.1 Applying the One-Scan algorithm (with $k = 5$) on the data set D in Fig. 1, points p_1, p_2, and p_3 will each be inserted into R. However, p_4 will be inserted into T since it is 5-dominated by p_1. Thus, the algorithm returns $\{p_1, p_2, p_3\}$ as the set of 5-dominant skyline points. □

4.2 Two-Scan Algorithm

In the One-Scan algorithm, free skyline points (i.e., T) need to be maintained to compute the k-dominant skyline points. Since the set of free skyline points could be much larger than the set of k-dominant skyline points, maintaining T can incur large space and computation overhead. To overcome this limitation, we present our second approach (shown in Algorithm 2) which avoids the need to maintain T by scanning D twice.

In the first scan of D (steps 1 to 10), a set of candidate k-dominant skyline points, R, is computed progressively by comparing each point $p \in D$ against the computed points in R. If a point $p' \in R$ is k-dominated by p, then p' is removed from R. At the end of the comparison against R, p is added into R if it is not k-dominated by any point in R. After the first scan of D, R contains the candidate k-dominant skyline points. Recall that false positives can exist in R due to property P2 in Lemma 4.1.

To eliminate the false positives produced by the first scan, a second scan of D (steps 11 to 14) is necessary. To determine whether a point $p' \in R$ is indeed k-dominant, it is sufficient to compare p' against each point $p \in D - R$ that occurs earlier than p' in D, since those points that occur later than p' in D have been already compared against p' in the first scan. Note that this optimization can be implemented by associating each point with its position in D, and using this information to avoid unnecessary comparisons.

The efficiency of the Two-Scan approach is dependent on how effective are the k-dominant points in pruning non-

Algorithm 2 Two-Scan Algorithm (D, S, k)

```
1: initialize set of $k$-dominant skyline points $R = \emptyset$
2: for every point $p \in D$ do
3:   initialize isDominant = true
4:   for every point $p' \in R$ do
5:     if ($p'$ $k$-dominates $p$) then
6:       isDominant = false
7:     if ($p$ $k$-dominates $p'$) then
8:       remove $p'$ from $R$
9:   if (isDominant) then
10:    insert $p$ into $R$
11: for every point $p \in D$ do
12:   for every point $p' \in R$, $p' \neq p$ do
13:     if ($p$ $k$-dominates $p'$) then
14:       remove $p'$ from $R$
15: return $R$
```

dominant skyline points during the first scan. If the number of false positives produced by the first scan is small, then the performance of the second scan and hence the overall approach will be good. The following result gives an indication of the pruning ability of dominant skyline points.

THEOREM 4.2. *Consider a d-dimensional data set D with more than 2^d points such that the dimensions are pairwise independent, and no two points in D have the same value for the same dimension. If a data point $p \in D$ can not k-dominate any point in D, then p is a k-dominant skyline point with probability less than $e^{-2^{d-k}}$.*

The above theorem shows that when k is small enough, it is very unlikely that a dominant skyline point does not k-dominate some other point. Thus, this indicates that each k-dominant skyline is likely to prune off many false positives during the first scan of D. For example, when $k \leq 3d/4$, the above probability is smaller than $e^{-2^{d/4}}$. For $d = 20$, this probability works out to be 1.27×10^{-14}, which is a very small number.

Example 4.2 Applying the Two-Scan algorithm (with $k = 4$) on the data set D in Fig. 1, we note that both p_1 and p_2 (which are indeed 4-dominant skyline points) will be inserted into R at the end of the first scan regardless of the order of the points in D. This example demonstrates the effective pruning ability of the dominant skyline points in eliminating non-dominant skyline points. □

4.3 Sorted Retrieval Algorithm

Our third approach is inspired by the ideas in [5]. The data points in D are first sorted (in non-ascending order) for each dimension $s_i \in S$, and each sorted set of points is stored in an array $D_i[1 \cdots |D|]$ (steps 1 to 3)[3]. Thus, $D_i[j].s_i \geq D_i[j+1].s_i$, $\forall\ i \in [1, |S|]$, $\forall\ j \in [1, |D|)$. Each sorted array D_i is associated with a cursor, denoted by c_i, which is initialized to point to the first array entry (step 3). The algorithm maintains two sets: (1) R, the set of k-dominant skyline points (which is initialized to empty); and (2) T, the set of candidate k-dominant skyline points (which is initialized to D). Non-k-dominant skyline points in T are progressively eliminated from T, while k-dominant skyline points in T are progressively moved to R.

[3]For space efficiency, instead of storing data points in D_i, the array entries can simply store pointers to the points in D.

Unlike the first two approaches, which sequentially scans D to compute dominant skyline points, the Sorted Retrieval approach iteratively chooses one of the dimensions s_i (using a function `FindNextDimensions` in step 9), and processes the batch of data points in D_i, denoted by D', that have the same s_i value as $D_i[c_i]$ (steps 10 to 20). The iterative processing terminates once T becomes empty. Since each data point $p \in D$ is stored a total of $|S|$ times in the sorted arrays, each point can potentially be processed in $|S|$ different iterations. We use a counter, denoted by $count(p)$, to keep track of the number of times that a point p has been processed. The counter values are initialized to 0 (steps 4 and 5). In each iteration, the selected batch of points D' is processed in two phases. The first phase (steps 11 to 16) uses D' to eliminate points in T that are determined to be non-dominnant. Specifically, if $p' \in D'$ is being processed for the first time (step 12), then p' is used to eliminate the points in T that are k-dominated by p' (steps 13 to 15). The counter for each $p' \in D'$ is updated at the end of the first phase (step 16).

The second phase (steps 17 to 19) moves the points in T that are determined to be k-dominant skyline points to R. Specifically, a point $p' \in D'$ is determined to be a k-dominant skyline point if it satisfies two conditions: (C1) p' has not yet been k-dominated by any processed point (i.e., p' is still in T); and (C2) p' has been processed $d-k+1$ times (i.e., $counter(p') = d-k+1$). The correctness of these conditions is based on the following observation: if a point p' is k-dominated by some other point p, then p' can be processed in an earlier batch of points than p in at most $d-k$ iterations. This is because the definition of k-dominance implies that p must be processed in an earlier batch than p' or in the same batch as p' in at least k iterations. Therefore, condition (C2) implies that any point p that could possibly k-dominate p' would have been processed at least once, and together with condition (C1), it means that p' is guaranteed to be a k-dominant skyline point. At the end of each iteration, the cursor c_i is updated accordingly to beyond the last processed point in D_i (step 20).

The performance of this approach depends crucially on the choice of the heuristic function `FindNextDimension` which selects the next dimension and hence sorted array to be processed. There are several straightforward options for this function, such as round-robin iteration and one-dimensional iteration. In Algorithm 3, we use the round-robin iteration heuristic which chooses the dimension that has been selected the least often; ties are broken by selecting the dimension with the smallest dimension index number.

Example 4.3 Applying the Sorted Retrieval approach (with $k = 4$) on D in Fig. 1, we first sort D to obtain $D_1, \cdots, D_6$ as shown in Fig. 3, where for clarity, points having the same s_i values are enclosed within braces. The first three iterations all select dimension s_1. At the end of the first iteration, the points p_3 and p_4 (which are 4-dominated by p_1) are removed from T, $count(p_1) = 1$, and $c_1 = 2$. At the end of the second iteration, we have $count(p_1) = 2$, and $c_2 = 2$. At the end of the third iteration, we have $count(p_1) = 3$; since $count(p_1)$ $= d - k + 1 = 3$, the algorithm concludes that p_1 is a 4-dominant point and p_1 is moved from T to R. The point p_5 is eliminated from T (as it is 4-dominated by p_2) at the end of the fifth iteration. Finally, at the end of the tenth iteration, $count(p_2) = 3$ and p_2 is moved from T to R as a 4-dominant point. Since T becomes empty, the processing terminates and the algorithm returns $R = \{p_1, p_2\}$ as the answer. □

Algorithm 3 Sorted Retrieval (D, S, k)

```
 1: for each dimension $s_i \in S$ do
 2:    sort $D$ in non-ascending order of each point's $s_i$ value and
       store the sorted points in array $D_i[1 \cdots |D|]$
 3:    initialize the cursor for $D_i$, $c_i = 1$
 4: for each $p \in D$ do
 5:    initialize $count(p) = 0$
 6: initialize the set of $k$-dominant points $R = \emptyset$
 7: initialize $T = D$
 8: while ($T \neq \emptyset$) do
 9:    $s_i$ = FindNextDimension ($c_1, \cdots, c_{|S|}$)
10:    let $D' = \{D_i[c_i], D_i[c_i+1], \cdots, D_i[c_i+m-1]\}$,
          where $D_i[c_i+m].s_i \neq D_i[c_i].s_i$, and
          $D_i[c_i].s_i = D_i[c_i+1].s_i = \cdots = D_i[c_i+m-1].s_i$
11:    for each $p' \in D'$ do
12:       if ($count(p') = 0$) then
13:          for each $p \in T$ do
14:             if ($p'$ $k$-dominates $p$) then
15:                remove $p$ from $T$
16:       $count(p') = count(p') + 1$
17:    for each $p' \in D'$ do
18:       if ($p' \in T$) and ($count(p') = d - k + 1$) then
19:          move $p'$ from $T$ to $R$
20:    $c_i = c_i + m$
21: return $R$
```

Function FindNextDimension($c_1, \cdots, c_{|S|}$)

```
 1: return the dimension $s_i \in S$ with the smallest $c_i$ and smallest
    $i$
```

4.3.1 *Analysis*

In this section, we use the concept of instance optimality [5] to show that our proposed round-robin iteration method is instance optimal with a constant factor.

Let $\mathcal{A}$ be a class of algorithms, and let $\mathcal{D}$ be a class of legal inputs with at most $O(1)$ points having same value on any dimension. If the time cost of the algorithm $A \in \mathcal{A}$ on data $D \in \mathcal{D}$ is $C(A, D)$, an algorithm $A_i \in A$ is said to be instance optimal if it satisfies the following condition.

Definition 4.3. *[5] An algorithm $A_i \in \mathcal{A}$ is instance optimal in $\mathcal{A}$ on $\mathcal{D}$ if $C(A_i, D) = O(C(A_j, D))$ for any $A_j \in \mathcal{A}$ and $D \in \mathcal{D}$.*

In the following part of the section, we prove that round-robin iteration is such an instance optimal iteration method based on the sorted arrays.

Lemma 4.4. *If a point p can be pruned by any iteration method after t comparisons, it can be pruned by round-robin after at most $O(td)$ comparisons.*

Lemma 4.5. *If any iteration method can assert that point p is a k-dominant skyline point after t comparisons, round-robin method can assert it after at most $O(td)$ comparisons.*

Theorem 4.6. *Round-robin iteration is instance optimal in the iteration method class $\mathcal{A}$ and data sets $\mathcal{D}$.*

5. TOP-δ DOMINANT SKYLINE

The goal of computing k-dominant skyline points is to reduce the number of interesting points returned by the skyline

D_1	D_2	D_3	D_4	D_5	D_6
p_1	p_1	p_1	p_4	p_2	p_2
p_3	p_3	p_3	p_2	$\{p_3, p_5\}$	$\{p_3, p_5\}$
$\{p_2, p_5\}$	$\{p_2, p_5\}$	$\{p_2, p_5\}$	p_5	p_1	p_1
p_4	p_4	p_4	p_1	p_4	p_4
			p_3		

Figure 3: Sorted Data Sets of D from Fig. 1

Algorithm 4 Ext-One-Scan Algorithm (D, S, δ)

```
 1: initialize set of top-$\delta$ dominant skyline points $R = \emptyset$
 2: for every point $p \in D$ do
 3:   initialize $maxkdom(p) = 0$
 4:   for every point $p' \in R$ do
 5:     $maxkdom(p) = \max\{maxkdom(p),\ maxDom(p', p)\}$
 6:     $maxkdom(p') = \max\{maxkdom(p'),\ maxDom(p, p')\}$
 7:     if $(maxkdom(p') = |S|)$ then
 8:       remove $p'$ from $R$
 9:     else if $(maxkdom(p) = |S|)$ then
10:       break out of inner for-loop
11:   if $(maxkdom(p) < |S|)$ then
12:     insert $p$ into $R$
13: return the $\delta$ points in $R$ with smallest $maxkdom(.)$ values
```

operator. However, the number of dominant skyline points can still be large when k is not sufficiently small. On the other hand, if k is too small, no (or too few) k-dominant skyline points may be returned. To avoid the difficulty of choosing the right value of k, we consider instead computing *top-δ* dominant skyline queries, which are queries to find the smallest k such that there are at least δ number of dominant skyline points (i.e., $|DSP(k, D, S)| \geq \delta$).

In this section, we describe how to extend the dominant skyline algorithms in the previous section to evaluate top-δ dominant skyline queries. The input parameters to each algorithm are D, S, and δ (instead of k).

Given two points p and p', we use $maxDom(p, p')$ to denote the largest value $k \in [0, |S|]$ such that p k-dominates p'.

5.1 One-Scan Algorithm

The key idea behind the extension of One-Scan approach (shown as Algorithm 4) is to keep track, for each point $p \in D$, the maximum value of k for which p has been k-dominated. We use $maxkdom(p)$ to denote this value for p. A point is maintained in R so long as $maxkdom(p) < |S|$ (i.e., p is at least a free skyline point). At the end of processing all the points in D, for each point $p \in R$, p is a k-dominant skyline point $\forall\ k \in [maxkdom(p) + 1, |S|]$. Thus, the top-$\delta$ dominant skyline points are the δ points in R with the smallest value of $maxkdom(.)$.

5.2 Two-Scan Algorithm

Since the Two-Scan algorithm maintains only candidate k-dominant skyline points but not the free skyline points that are not k-dominant, it is not possible to precisely maintain the $maxkdom(.)$ values as in the extended One-Scan approach. Instead, we apply a binary search technique to determine the smallest k value such that $|DSP(k, D, S)| \geq \delta$. The details are given in Algorithm 5.

Algorithm 5 Ext-Two-Scan Algorithm (D, S, δ)

```
 1: initialize set of top-$\delta$ dominant skyline points $R = \emptyset$
 2: initialize $k_{min} = 1$ and $k_{max} = |S|$
 3: repeat
 4:   $k = (k_{min} + k_{max})/2$
 5:   $T =$ Two-Scan $(D, S, k)$
 6:   if $(|T| = \delta)$ then
 7:     $R = T$
 8:     $k_{min} = k_{max} + 1$
 9:   else if $(|T| > \delta)$ then
10:     $R = T$
11:     $k_{min} = k + 1$
12:   else
13:     $k_{max} = k - 1$
14: until $(k_{min} > k_{max})$
15: return $\delta$ points in $R$
```

Although the Two-Scan algorithm might be invoked $\log(|S|)$ times in the worst case, when the minimum value of k for $|DSP(k, D, S)| \geq \delta$ to hold turns out to be small (which is necessary when δ is small), Theorem 4.2 indicates that the Two-Scan algorithm and hence its extended variant are very efficient due to the pruning effectiveness of the dominant skyline points.

5.3 Sorted Retrieval Algorithm

To extend the Sorted Retrieval approach to evaluate top-δ dominant queries, we need to maintain two variables for each point $p \in T$: (1) $maxkdom(p)$ (as defined in Section 5.1); and (2) $maxkdomBound(p)$, which is the upper bound for $maxkdom(p)$. The values for $maxkdom(p)$ and $maxkdomBound(p)$, which are initialized to 0 and $|S|$, respectively, are updated as points in the sorted arrays are being processed. This information enables efficient checking of whether or not a point $p \in T$ is a top-δ dominant skyline point and is based on the following two properties: (P1) A point $p \in T$ can not be in the answer (i.e., can be removed from T) if $maxkdom(p)$ is larger than the δ^{th} smallest $maxkdomBound(.)$ values in $R \cup T$; and (P2) A point $p \in T$ can be confirmed to be in the answer (i.e., can be moved from T to R) if $maxkdomBound(p)$ is smaller than the δ^{th} smallest $maxkdom(.)$ values in $R \cup T$. The details of the algorithm are shown as Algorithm 6.

Example 5.1 Consider finding the top-2 dominant skyline points in D from Fig. 1. Initially, each point in D has values of 0 and 6 for $maxkdom(p)$ and $maxkdomBound(p)$, respectively. After $p_1 \in D_1$ has been processed, we have $maxkdom(p_2) = 3$, $maxkdom(p_3) = 4$, $maxkdom(p_4) = 5$, and $maxkdom(p_5) = 3$. After $p_1 \in D_3$ has been processed, $maxkdomBound(p_1)$ is reduced to 3. The processing of $p_4 \in D_4$ does not affect any variable values. But after $p_2 \in D_6$ has been processed, the points p_3, p_4 and p_5 can all be eliminated from T because their $maxkdom(.)$ values are larger $maxkdomBound(p_2)$. Thus, p_1 and p_2 are returned as the top-2 dominant skyline points. □

6. WEIGHTED DOMINANT SKYLINE

The dominant skyline problem so far gives every dimension equal importance in the result. This may not be enough in all the cases, since sometimes the user may want to give some bias to some attributes of greater interest. A simple extension from the original problem is to assign some weights to the dimensions and determine the (dominant) skyline points over some subset of dimensions with enough weights. In the basketball players' statistics data, for example, the user may want to find those exceptional players who

Algorithm 6 Ext-Sorted Retrieval (D, S, δ)

```
1: for each dimension $s_i \in S$ do
2:    sort $D$ in non-ascending order of each point's $s_i$ value and
      store the sorted points in array $D_i[1 \cdots |D|]$
3:    initialize the cursor for $D_i$, $c_i = 1$
4: for each $p \in D$ do
5:    initialize $count(p) = 0$
6:    initialize $maxkdom(p) = 0$ and $maxkdomBound(p) = |S|$
7: initialize the set of top-$\delta$ dominant skyline points $R = \emptyset$
8: initialize $T = D$
9: while ($T \neq \emptyset$) do
10:   $s_i$ = FindNextDimension ($c_1, \cdots, c_{|S|}$)
11:   let $D' = \{D_i[c_i], D_i[c_i+1], \cdots, D_i[c_i+m-1]\}$,
         where $D_i[c_i+m].s_i \neq D_i[c_i].s_i$, and
         $D_i[c_i].s_i = D_i[c_i+1].s_i = \cdots = D_i[c_i+m-1].s_i$
12:   for each $p' \in D'$ do
13:      if ($count(p') = 0$) then
14:         for each $p \in T$ do
15:            $maxkdom(p) = \max\{maxkdom(p), maxDom(p', p)\}$
16:            $maxkdom(p') = \max\{maxkdom(p'), maxDom(p, p')\}$
17:            let $\alpha$ be the $\delta^{th}$ smallest $maxkdomBound(q), q \in R \cup T$
18:            if ($maxkdom(p) > \alpha$) then
19:               remove $p$ from $T$
20:         $count(p') = 1$
21:      if ($maxkdom(p') < maxkdomBound(p')$) then
22:         $maxkdomBound(p') = maxkdomBound(p') - 1$
23:      else
24:         $maxkdomBound(p') = maxkdom(p')$
25:   for each $p' \in D'$ do
26:      let $\beta$ be the $\delta^{th}$ smallest $maxkdom(q), q \in R \cup T$
27:      if ($p' \in T$) and ($maxkdomBound(p') < \beta$) then
28:         move $p'$ from $T$ to $R$
29:   $c_i = c_i + m$
30: return $R$
```

are better at defense than attack, by giving more weight to defensive attributes, such as block and steal.

To support this type of weighted query, our previous algorithms need to be extended only slightly. Specifically, when comparing two points p_1 and p_2, we record the weights of the dimensions on which p_1 dominates p_2 and sum them up. If the sum exceeds a threshold w, we say that p_1 dominates p_2 with weight w. A point p is said to be a *w-dominant skyline* if no point can dominate p on dimensions whose weight sum is over w. The following lemma shows that weighted dominant skyline shares similar properties as the original dominant skyline.

LEMMA 6.1. *Given a subset of dimensions $S = \{s_1, s_2, \cdots, s_d\}$ and a corresponding positive weight set $W = \{w_1, w_2, \cdots, w_d\}$, if a point $p \in S$ w-dominates another point q, then p also w'-dominates q for $w' \leq w$.*

From the simple lemma above, we can restate almost all of the earlier theorems in a weighted form. Since the one-scan and two-scan algorithms are based on point comparison, the weight assignments for the dimensions have no impact on its correctness. The structure of these two algorithms remain largely unchanged and only the pair-wise domination comparison procedure must be modified to cater to the weights. For the sorted retrieval method, instead of counting the number of times a point has been processed from the sorted arrays, we need to sum up all the weights of the processed dimensions.

Note that these modification do not affect the computational complexities of all the proposed methods. However, having weight assignments on the dimensions can impact the effectiveness of the pruning and the frequency distribution of w-dominant skyline points with varying w. We will show how these affect the running time of the algorithms in our experimental study.

Table 1: Parameters in Experiments

Parameter	Description
d	Dimension Number
$Size$	Data Size
$Dist$	Distribution
k	Constraint Parameter
δ	Top Dominant Skyline Point Number
w	Constrained Parameter in Weighted Query
R	the ratio of maximum weight to minimum weight

7. EXPERIMENTS

We have implemented all the algorithms proposed in this paper: One-Scan Algorithm (OSA), Two-Scan Algorithm (TSA) and Sorted Retrieval Algorithm (SRA). In this section, we compare their performances, and report our findings.

7.1 Experiment Setting

We use both synthetic data sets and real data sets in the experiments. The generation of the synthetic data sets is controlled by the parameters listed in Table 1.

The dimension number d is the number of attributes of the points, while the data size $Size$ is the number of points in the data set. There are three optional distributions in the synthetic data sets: Correlated, Independent and Anti-Correlated. In the correlated data set, all dimensions are positively correlated to each other. As such, there are very few skyline points, free or k-dominant, in the data set. In the independent data set, dimensions are independent of each other. Under this assumption, points rarely dominate each other when the dimension number grows, so the free skyline set becomes large. In the anti-correlated data set, dimensions are negatively correlated. Almost all points are free skyline points in this type of data sets. In Table 2, we show the number of the dominant skyline points on a 15-dimensional data set with 100K points on different distributions and different constraint parameter k. This table shows that when k is close to dimension number d, the number of dominant skyline point in the anti-correlated data set is much larger than that in the independent and correlated data sets. However, when k is small, the correlated data set can still have some dominant skyline points, while no dominant skyline points can be found on the other two distributions. The constraint parameter k, w and the top dominant skyline parameter δ have the same meaning as that described in the paper. When we assign weights to the dimensions, we ensure that the sum of the weights on all dimensions is equal to the number of dimensions. In the weight generation, we use the ratio of maximum weight to minimum weight, R, to control the degree of bias on the weights. Given the ratio R, the d weights of the dimensions are generated by normalizing d random numbers between 1 and R.

The default parameter setting in our synthetic data set test is: $d = 15$, $Size = 100K$, $Dist$ =Independent, $k = 11$, $\delta = 100$, $w = 11$ and $R = 2$.

Table 2: Dominant Skyline Point Number

k	Correlated	Independent	Anti-Correlated
8	0	0	0
9	0	0	0
10	8	3	0
11	24	61	33
12	57	960	3175
13	134	7881	28305
14	436	33087	67866

We also study two different real data sets. The first is the NBA statistics data set[4]. This data set contains 17000 players' season records on 17 attributes from the first season of NBA in 1945. Every record contains the statistical value of a player's performance in one season, such as game played(GP), points(PT), rebounds(RB) and so on. One player may have several records if he played in NBA for more than one season. The second data set is MovieLens data set[5], which contains 100,000 ratings (1-5) from 943 users on 1682 movies. The data was collected by the MovieLens web site from September 1997 to April 1998. All the users in this data set has rated at least 20 movies. To make the records comparable, we insert 0 to all the empty entries of the movies a user did not rate.

All the experiments are conducted on a PC with Intel Pentium 2.4GHz CPU and 2G main memory, running Linux with kernel 6.2.14.

7.2 On Synthetic Data Sets

7.2.1 On k-Dominant Skyline

We evaluate the computational costs of the algorithms on three different distributions with respect to the constraint parameter k. From the results in Fig. 4, we observe that TSA is more efficient than the other two methods on all distributions when $k < 12$. This is because the dominant skyline points can prune almost all other points in the first scan (as implied in Theorem 4.2). However, as k increases, TSA becomes slower than SRA since there are too many false candidates left after the first scan. It is even worse than OSA on anti-correlated data set when $k = 14$. The performance of OSA is stable in all cases because the computation of free skyline cannot be reduced with small k, and this computation dominates the computation time.

In Fig. 5, we show the influence of dimensionality on the efficiencies of the three algorithms. When k is small, both TSA and SRA are much faster than OSA on all three distributions. When k is close to the dimensionality d, TSA is less efficient than OSA. With increasing dimensionality, this disadvantage grows so great that TSA is several times slower than the other two algorithms on 20-dimensional data set with $k = 19$. As shown in the figure, SRA is more scalable on high dimensional data sets.

We also study the effect of the size of the datasets on the performances of the three algorithms. The results, depicted in Fig. 6, show that when the size of the data set grows from 50K to 200K, the computation time of the three algorithms all increase by about one order of magnitude. Moreover, the relative performance of the schemes remain largely the same as that in earlier experiments: TSA performs best while OSA is the most inferior.

7.2.2 On Top-δ Dominant Skyline

For top-δ dominant skyline query, the most important parameter is δ, the number of dominant skyline points desired. The efficiency of TSA on top-δ query is strongly related to the distribution of the dominant skyline points. On the 15-dimensional correlated data set, for example, there are 57 dominant skyline points when $k = 12$ (shown in Table 2). When δ grows from 50 to 100, TSA has to run on $k = 13$ instead of $k = 12$ before finding all these top dominant skyline points. Since the computation time of TSA on $k = 13$ is much more than that on $k = 12$ (shown in Fig. 4), it spends much more time on top-100 query than top-50 query. Using the same logic, we can explain all the sudden increase in time of TSA on Fig. 7. For SRA, the increase of computation time of SRA is much smaller because fewer points in the candidate set do not improve the pruning threshold of lower bounds greatly in the early stages of SRA. Unlike the other two algorithms, OSA's efficiency is worst but it is the most stable since the final number of points desired does not have impact on its computing process until the final step.

When tested on data sets with different dimensions, we observe from Fig. 8 that TSA is faster on higher dimensional data set on anti-correlated distribution. This phenomenon is still related to the distribution of the dominant skyline points. When dimensionality grows, the top dominant skyline points can be obtained at a lower level of k, which contributes to TSA's efficiency. OSA and SRA cannot take advantage of this since both schemes' efficiency are proportional to the dimension number but not to the level where the top-dominant skyline can be obtained.

In Fig. 9, we present the impact of data size on the efficiency of top-δ dominant skyline query. The trends of the algorithms are similar to the performance to that on k-dominant skyline query shown in Fig. 6: TSA performs the best, followed by SRA, and OSA is the worst.

7.2.3 On Weighted Dominant skyline

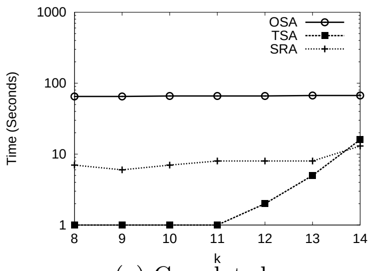
In this set of experiments, we examine the performance of the three algorithms on weighted dominant skyline queries. When the ratio R of the weights increases, most of the weight is assigned to a smaller fraction of the dimensions. If a point is not dominated on those dimensions with heavy weights, they are very likely to be dominant skyline points since any subspace with weight sum over w must contain some of these dimensions. This impact is not large but is still observable in Fig. 10. TSA and SRA turn out to be faster on independent and anti-correlated data sets when the weight ratio is varied from 2 to 5. This improvement in efficiency is not as significant in correlated data set because the dominant skyline does not change with the variation of weights on correlated distribution. Since the dominating set is independent of dimension weights, no matter what the weights are, the OSA algorithm stays at the same level in all three types of data sets.

To summarize, from the experiments on synthetic data sets, we can conclude that when k, δ and $Size$ are small, TSA is the most efficient algorithm. In other cases, SRA is faster than the others and has more stable performance on different data sets.

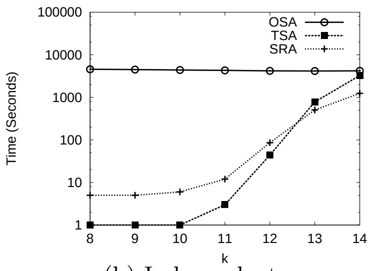
7.3 On Real Data Sets

[4]http://www.basketballreference.com/

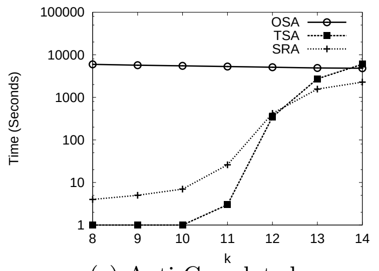
[5]http://movielens.umn.edu/



(a) Correlated

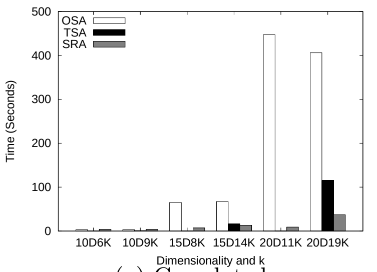


(b) Independent

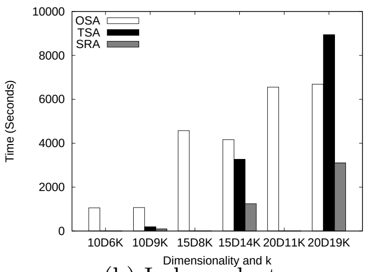


(c) Anti-Correlated

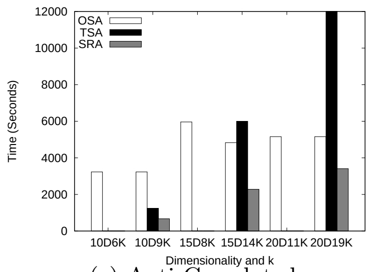
Figure 4: k-Dominant Skyline Test on varying k



(a) Correlated

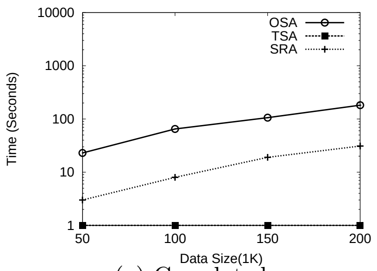


(b) Independent

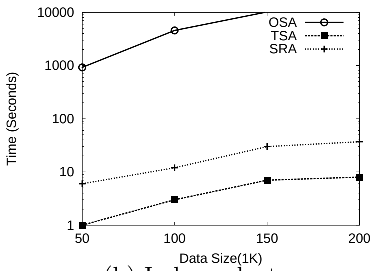


(c) Anti-Correlated

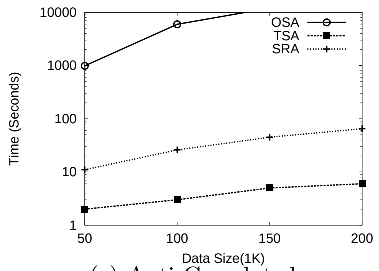
Figure 5: k-Dominant Skyline Test on varying dimension



(a) Correlated

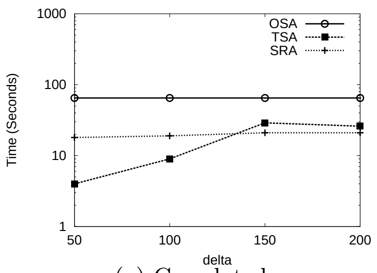


(b) Independent

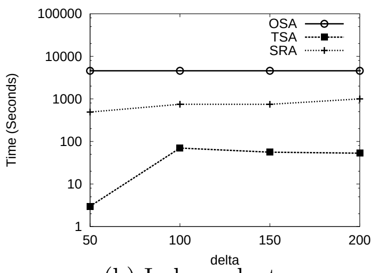


(c) Anti-Correlated

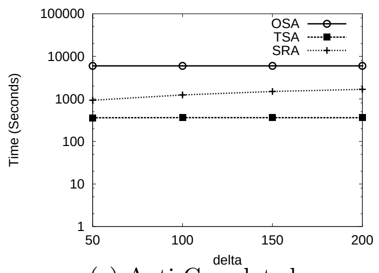
Figure 6: k-Dominant Skyline Test on varying data size



(a) Correlated

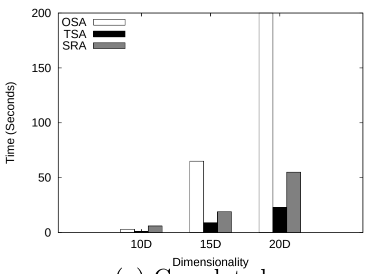


(b) Independent

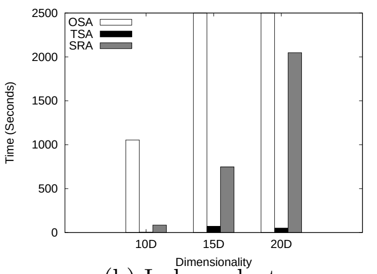


(c) Anti-Correlated

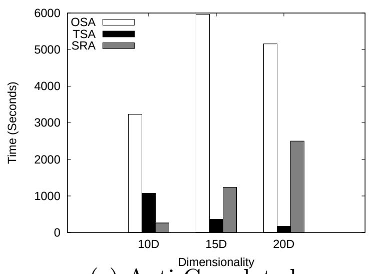
Figure 7: top-δ Dominant Skyline Test on varying δ



(a) Correlated



(b) Independent

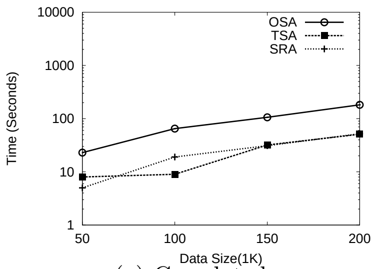


(c) Anti-Correlated

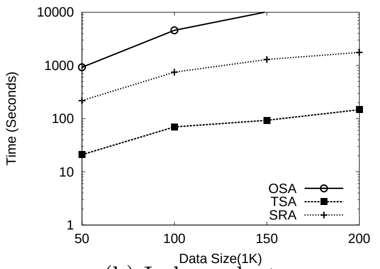
Figure 8: top-δ Dominant Skyline Test on varying dimension

7.3.1 *On NBA Data Set*

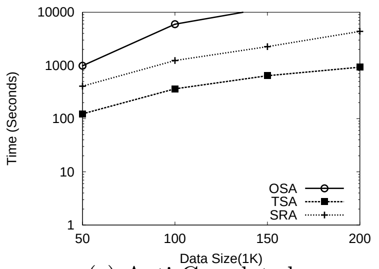
In Figs. 11(a) and 11(b), we show several experimental results on the NBA data set with equal weights on all attributes. When varying the constraint parameter k, TSA is the most efficient algorithm when $k < 14$, but is worst among the three algorithms when $k > 15$. SRA is faster than the other two when k is large. For top-δ dominant skyline query, SRA is much more efficient than TSA because of



(a) Correlated

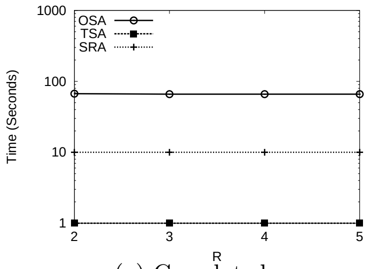


(b) Independent

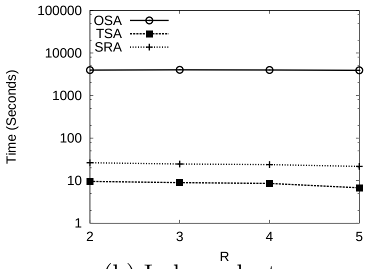


(c) Anti-Correlated

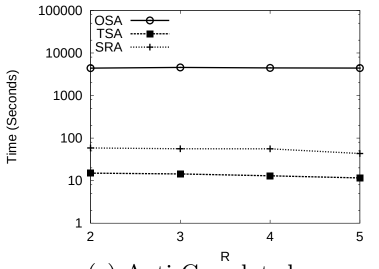
Figure 9: top-δ Dominant Skyline Test on varying data size



(a) Correlated



(b) Independent



(c) Anti-Correlated

Figure 10: k-Dominant Skyline Test with different Weight Assignments

Table 3: Top-5 Dominant Skyline Results on NBA data set

Equal Weight	Defender Biased	Shooter Biased
Wilt Chamberlain 1961	Wilt Chamberlain 1961	Wilt Chamberlain 1961
Bob Mcadoo 1974	Artis Gilmore 1974	Rick Barry 1971
George Mcginnis 1974	George Mcginnis 1974	Michael Jordan 1986
Kareem Abdul-jabbar 1975	Kareem Abdul-jabbar 1975	Michael Jordan 1987
Julius Erving 1975	Julius Erving 1975	Michael Jordan 1988
Artis Gilmore 1975	Michael Jordan 1987	Michael Jordan 1989
Michael Jordan 1986		Gary Payton 1999
Michael Jordan 1987		Kobe Bryant 2002
Michael Jordan 1988		

its great advantage over TSA on k-dominant skyline query with large k.

In Table 3, we show the top-5 dominant skyline results on NBA data set with three different weights assignments. The first assignment is equal-weight assignment, i.e., all dimensions are assigned the same weight. With the same weight on all dimensions, we successfully find the superstars in NBA's history, such as Chamberlain and Jordan, listed in the first column. In the second weight assignment, six attributes related to defense, such as block, steal and rebound, are given weights two times more than all other dimensions. From the result of the defender biased weights in the second column of the table, we can find famous defenders in NBA. The third weight assignment concentrates on shooters, which gives shooters' strengths, such as points scored and three pointers, two times more weight than other attributes. Legendary shooters appear in the third column as expected.

Since the NBA data set is fairly correlated, especially on those attacking attributes, TSA is faster on shooter biased weights than the other weights assignments in Fig. 11(c). This shows that TSA is more sensitive to weight assignment than the other two algorithms.

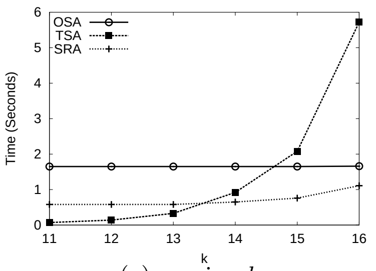
7.3.2 On Movie-Lens Data Set

We also evaluate the impact of parameters on the movie-lens data set with equal weights in Figs. 12(a) and 12(b). Although TSA and SRA are much faster than OSA on k-dominant skyline query, they cannot beat OSA on top-δ dominant skyline query because of the large dimensionality of the data set. For TSA, the binary search on large k consumes too much time to locate the best level of k. For SRA, too many different top rated movies by different users reduces its pruning efficiency.

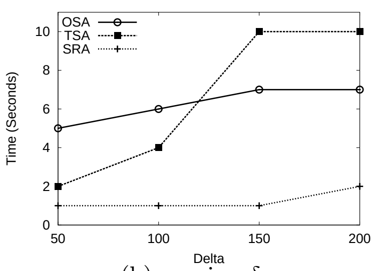
Table 4: Top-5 Dominant Skyline Result on Movie-Lens data set

Equal Weight	Rate Number Based
Star Wars (1977)	Star Wars (1977)
Fargo (1996)	Pulp Fiction (1994)
Contact (1997)	Silence of the Lambs (1991)
English Patient (1996)	Fargo (1996)
Scream (1996)	Godfather (1972)

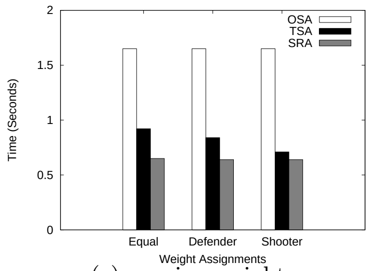
On the movie-lens data set, we also try two different weight assignments. With the equal weight assignment to every user, five top movies are found and listed in the first column of Table 4. When the weights for every user is set proportional to the number of movies he or she rated, we see some different results in the second column. The difference comes about because, in the movie-lens data set, most users have rated only a small fraction of the movies. If we give equal weights to all users, a movie A has advantage over another movie B only if more users rated A, no matter how many users like B more than A. Since the data set was collected from 1997 to 1998, 4 out of 5 top dominant skyline points of equal weight are movies from 1996 to 1997, which had been just watched by the users. From the result in second



(a) varying k

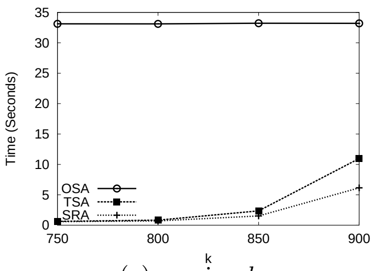


(b) varying δ

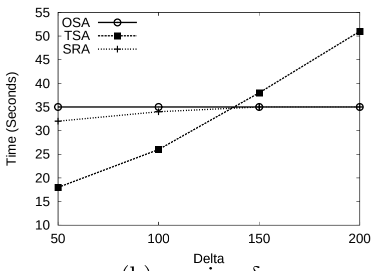


(c) varying weights

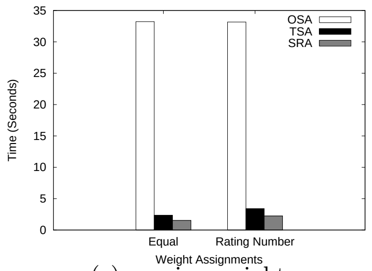
Figure 11: Tests on NBA data set



(a) varying k



(b) varying δ



(c) varying weights

Figure 12: Tests on Movie-Lens data set

column, we believe this problem is alleviated in the biased weight assignment.

With heavier weights on the users with more ratings, the computation times of TSA and SRA increase in Fig. 12(c), since the data set is so sparse that those heavily weighted users can have very few common movies rated, which makes it even harder to find dominant skyline points.

8. CONCLUSION

The skyline operator has been used as an effective mechanism to identify "dominating" points in a multi-dimensional data set. Unfortunately, as the dimensionality of the data set grows, the skyline operator begins to lose its discriminating power and returns a large fraction of the data. In this paper, we proposed a generalization of the skyline concept, called k-dominant skyline, to overcome this difficulty. We presented three different algorithms to solve the k-dominant skyline problem. We defined notions of a top-δ dominant skyline query and a weighted (dominant) skyline query, and showed how the three algorithms for the k-dominant skyline problem could be extended to address these problems as well. Our experimental results showed that our methods can find interesting objects in the data set efficiently. In summary, the notion of k-dominant skylines proposed in this paper gracefully extends traditional skylines, and leads to both more meaningful skyline results as well as more efficient computation.

Acknowledgements Chee-Yong Chan and Kian-Lee Tan are supported in part by NUS Grant R-252-000-207-112. H.V. Jagadish is supported in part by US National Science Foundation Grant IIS-0438909.

9. REFERENCES

[1] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD*, 2000.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[3] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, 2006.

[4] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.

[5] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[6] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, 2005.

[7] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2), 1999.

[8] W. Kießling. Foundations of preferences in database systems. In *VLDB*, 2002.

[9] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB*, 2002.

[10] H. Kung, F. Luccio, and F. Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4), 1975.

[11] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. Dada: A data cube for dominant relationship analysis. In *SIGMOD*, 2006.

[12] D. H. McLain. Drawing contours from arbitrary data points. *The Computer Journal*, 17(4), November 1974.

[13] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, 2003.

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

[15] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[16] K. L. Tan, P. K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.

[17] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of skyline cube. In *VLDB*, 2005.