International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 3, Issue 7, July 2014 ISSN 2319 - 4847

Computing Skyline Points for Decision Making Applications

[1]K.Swathi, M.Tech ,
[2]B.Renuka Devi, Assoc.Professor
Vignan's Lara Institute of technology & Science

ABSTRACT

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. 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. To evaluate manually these points it is possible, if the data is small but if it is high dimensional data, it is vulnerable. 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. Here are the algorithms for finding k-dominant skyline and processing algorithms and providing best insights. These algorithms 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. For example, if we want to buy a mobile we need to several phones and several features, 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. We note that in the above case, ranking can be done by providing some preference functions and requesting users to provide some weight assignments for their preferred features. However providing such weight assignments for a large number of dimensions is not always easy without any initial knowledge about the data. The aim of providing the skyline to the user is to help them to determine the weight assignment. Computing skylines in large number of data sets is challenging because of the large number of skyline points. On the mobile ranking website, for example, it is nearly impossible to find a mobile which is ranked lower than another mobile as both have different features with different qualities, 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 pair wise comparisons are per-formed without effecting any pruning.

1.2 K-Dominant Skyline and its Variants

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. As an example, this means that mobile *p* is only considered better than mobile *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 may be reasonable to consider mobile *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 have 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. If the whole set of free skyline points are 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. There is another algorithm, which is motivated by the rank aggregation algorithm which presorts 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, 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. The algorithms above 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.

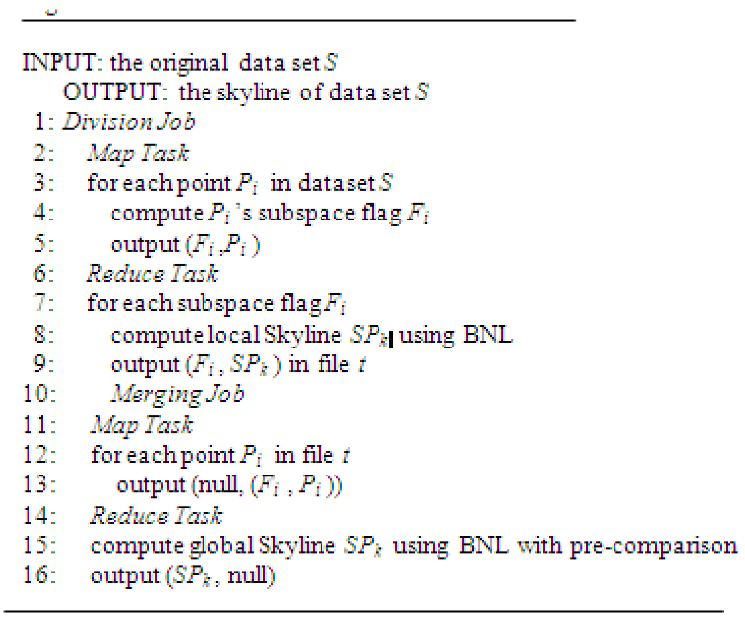
2. BASIC SKYLINE AND PROCESSING ALGORITHMS

The basic idea of skyline queries came from some old research topics like contour problem, maximum vectors and convex hull. First introduced the skyline operator into relational database systems, and proposed the following algorithms:

2.1 Block Nested Loops (BNL)

BNL is an iterative algorithm that repeatedly reads a set of data records for processing. MapReduce based BNL algorithm is a two-stage method. The first stage is to divide the whole data into small disjoint subsets. For each subset, we run a BNL procedure to compute the skyline. In the second stage, the local skylines are merged and filtered, thus the global skyline is obtained. The NN algorithm divides the global space into 2^d subspaces according to the first nearest neighbor. The D & C algorithm recursively partitions the data space according to the median of a certain dimension. In Block Nested Loop, we combine these ideas together such that the input dataset is divided into 2^d subspaces based on a carefully chosen point.

Algorithm1: BNL



```
INPUT: the original data set $S$
    OUTPUT: the skyline of data set $S$
 1: Division Job
 2:   Map Task
 3:   for each point $P_i$ in dataset $S$
 4:     compute $P_i$'s subspace flag $F_i$
 5:     output ($F_i$, $P_i$)
 6:   Reduce Task
 7:   for each subspace flag $F_i$
 8:     compute local Skyline $SP_k$ using BNL
 9:     output ($F_i$, $SP_k$) in file $t$
10:     Merging Job
11:   Map Task
12:   for each point $P_i$ in file $t$
13:     output (null, ($F_i$, $P_i$))
14:   Reduce Task
15:   compute global Skyline $SP_k$ using BNL with pre-comparison
16:   output ($SP_k$, null)
```

The procedure of BNL is outlined in the above Algorithm. In line 14, the key of the map output is null so that all records are gathered in one reducing call. The *pre-comparison* procedure does judgments in advance by using subspace flags. The bottleneck lies in the second stage where the merging is done by one single node. The performance may be significantly degraded in the case that the size of local skyline produced by the first stage is huge.

2.2 Sort Filter Skyline (SFS)

One shortcoming of BNL is that a record, once inserted into the window, might be discarded later. Therefore, this non-skyline record occupies window space, which may increase the iteration times. Sort-Filter-Skyline (SFS) was devised to overcome the drawback. The input data is sorted at first in a topological order compatible with the skyline criterion.

Thus, once a record R is inserted into the window, no record following it will dominate R and R is a skyline point. Therefore, at the end of each iteration, we can output all records inside. SFS algorithm is modified over BNL: presorting. However, MapReduce can only sort by the keys of intermediate key/value pairs. In order to achieve secondary sort for the values set in the reduce call, we exploit the grouping and comparing features. First, we assign key/value pairs of the same key to one reducer. Second, we put together the values of the same key. Third, user defined comparators are used to determine the order of records in the reduce phase.

2.3 Bitmap

In Bitmap, every point is mapped into a bit vector, and the whole dataset forms the bitmap structure. The bitmap structure reflects the order of data records in all dimensions, ignoring their magnitudes. Every comparison in Bitmap is a lightweight bitwise operation. However, in order to examine a record, we need to compare it with all the other records. Bitmap supports progressive skyline computation since a point can be returned to the user immediately once it is identified as a member of the skyline. But the spatial cost of Bitmap is quite large. The basic idea of MR-Bitmap keeps in line with that of Bitmap. In the first job, we build the bitmap structure. In the second job, we examine each point based on the bitmap structure.

2.4 Processing Skyline Queries:

Nearest Neighbor (NN) method is used 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. A new progressive algorithm named *Branch-and-Bound Skyline* (BBS) based on the best-first nearest neighbor (BF-NN) algorithm. Instead of searching for nearest neighbor again and again, it directly prunes using the R*-tree structure. Two more 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, which directly determines a set of interesting skyline points from the full set of dimensions. 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, there were three methods, FA, TA and NRA 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 has 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, 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. 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 sub-spaces $\{s_1, s_2\}$, $\{s_1, s_3\}$, $\{s_2, s_3\}$, 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. Also there is a possibility of cyclic dominance.

3. COMPUTING K-DOMINANT SKYLINES

Due to the possibility of cyclic dominance relationships, the existing algorithms for computing free skylines cannot be used directly for computing *k*-dominant skyline points. Here there are 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*.

3.1 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 in that it makes one sequential scan of the data set. The algorithm based on the following two key properties.

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, 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.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 is to try to maximize the number of skyline points that occur before the non-skyline 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.

Algorithm1: 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 $R$
```

3.2 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, maintain-ing T can incur large space and computation overhead. To overcome this limitation, we present our second approach 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 ∈ D against the computed points in R. If a point p_ ∈ 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. 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_ ∈ R is indeed k-dominant, it is sufficient to compare p_ against each point p ∈ 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 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.

Algorithm2: 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 $p- \in R$, $p- = p$ do
13:         if ($p$ $k$-dominates $p-$) then
14:             remove $p-$ from $R$
15: return $R$
```

3.3 Sorted Retrieval Algorithm

In this approach data points in *D* are first sorted (in non-ascending order) for each dimension s_i € *S*, and each sorted set of points is stored in an array $D_i[1 \cdots |D|]$ (steps 1 to 3). Thus, $D_i[j].s_i \geq D_i[j + 1].s_i$, $\forall i$ € [1, |*S*|], $\forall j$ € [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*. 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 FindNextDimension 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* € *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-dominant. Specifically, if *p* € *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* € *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* € *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 the following Algorithm, 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.

Algorithm3: 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 ($T = \emptyset$) 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 $D_i[c_i + m].s_i = D_i[c_i].s_i$, 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. Computing Top- Δ Dominant Skylines

The goal of computing k-dominant skyline points is to reduce the number of interesting points returned by the skyline 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., $|DSP(k, D, S)| \geq \delta$). Here we describe how to extend the dominant skyline algorithms in the previous 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^-.

Algorithm4: Ext-One-ScanAlgorithm(D, S, δ)

```
1: initialize set of top-$\delta$ dominant skyline points $R$
2: for every point $p$ € $D$ do
3:   initialize $maxkdom(p) = 0$
4:   for every point $p-$ € $R$ 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
```

4.1 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 € 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 € R, p is a k-dominant skyline point $\forall k$ € $[maxkdom(p) + 1, |S|]$. Thus, the top-δ dominant skyline points are the δ points in R with the smallest value of $maxkdom(.)$.

4.2 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. 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 to evaluate top-δ dominant queries, we need to maintain two variables for each point *p* € *T* : (1) *maxkdom*(*p*); 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*|, re-spectively, are updated as points in the sorted arrays are being processed. This information enables efficient check-ing of whether or not a point *p*€*T* is a top-δ dominant skyline point and is based on the following two properties: (P1) A point *p* € *T* cannot be in the answer (i.e., can be removed from *T*) if *maxkdom*(*p*) is larger than the δ^{th} small-est *maxkdomBound*(.) values in *R* ∪ *T* ; and (P2) A point *p* € *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*∪*T* .

Algorithm5: Ext-Two-ScanAlgorithm(*D*, *S*, δ)

```
    initialize set of top-$\delta$ dominant skyline points $R$
1:  initialize $k_{min} = 1$ and $k_{max} = |S|$
2:  repeat
3:      $k = (k_{min} + k_{max})/2$
4:      $T$ = Two-Scan ($D$, $S$, $k$)
5:      if($|T| = \delta$) then
6:          $R = T$
7:          $kmin = kmax + 1$
8:      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 $R$
```

is necessary when δ is small), Two-Scan algorithm and hence its extended variant are very efficient due to the pruning effectiveness of the dominant skyline points.

4.3 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 € T : (1) maxkdom(p); 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 € T is a top-δ dominant skyline point and is based on the following two properties: (P1) A point p € T cannot 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 ∪ T ; and (P2) A point p € 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∪T.

5. 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. The 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.

6. 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 presented different algorithms to solve the *k*-dominant skyline problem also the skyline processing algorithms and the query processing algorithms. We gave the notions of a top-δ dominant skyline query and a weighted (dominant) skyline query, and given the three algorithms for the k-dominant sky-line problem could be extended to address these problems as well. These algorithm results in finding 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. By checking all the dimensions one by one here in the computing algorithms, the skyline points could be accurate. Also the extension algorithm provides the most useful skyline points that reduces the skylines that makes useful to make the right decision. The weighted dominant skyline provides preference to the dimensions according to the user's interest to provide best insights.

REFERENCES

[1.] Nykiel, T., Potamias, M., Mishra, C., et al.: MRShare: Sharing Across Multiple Queries in MapReduce. In: Proceedings of VLDB, vol. 3(1), pp. 494–505 (2010)
[2.] Dittrich, J., Quiane-Ruiz, J.-A., Jindal, A., et al.: Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). In: Proceedings of VLDB, vol. 3(1), pp. 518–529 (2010)
[3.] Skyline computation, by jian wen, in CSG 399:http://www.ccs.neu.edu/home/jarodwen/materials/skyline_pre.pdf
[4.] Discovering strong skylines points in high dimensional space; Zhenjie Zhang, Xinyu Guo, Anthony K.H.Tung, Nan Wang @comp.nus.edu.sg
[5.] On finding skyline points for Range Queries in plane, Anilkishore k, Ananda swarup D, CCCG 2011, Toronto ON.
[6.] MRShare: Sharing across Multiple Queries in MapReduce,cs-eople.bu.edu/mp/images/pap 102.pdf, by T Nykiel ,2010
[7.] Finding k-dominant skylines in high dimensional space, {chancy, tankl, Anthony, zhangzh2}@comp.nus.edu.sg
[8.] Bu, Y., Howe, B., Balazinska, M.: HaLoop: Efficient Iterative Data Processing on Large Clusters. In: Proceedings of VLDB, pp. 285–296 (2010)
[9.] Extract interesting skyline points in High Dimension, Database Systems for Advanced Applications,Lecture Notes in Computer Science; Volume 5982, 2010, pp 94-108
[10.] Finding superior skyline points for multidimensional recommendation applications; Jing Yang, Gabriel Pui Cheong Fung, Wei Lu, Xiaofang Zhou, Hong Chen, Xiaoyong Du; January 2012, Volume 15, Issue 1, pp 33-60. Springer.
[11.] Fung, G.P.C., Lu, W., Du, X.: Dominant and k nearest probabilistic skylines. In: Proceedings of the 14th International Conference on Database Systems for Advanced Applications, DASFAA 2009
[12.] Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: Proc. of the 35th SIGMOD International Conference on Management of Data(SIGMOD 2009), pp. 483–494. ACM, New York (2009)
[13.] Lee, J., You, G.-W., Hwang, S.-W.: Personalized Top-k Skyline Queries in High-Dimensional Space. Inf. Syst. 34(1), 45–61
[14.] SkyDist: Data Mining on Skyline Objects; Christian Böhm, Annahita Oswald, Claudia Plant; Advances in Knowledge Discovery and Data Mining,Lecture Notes in Computer Science Volume 6118, 2010, pp 461-470
[15.] Skyline: Stacking Optimal Solutions in Exact and Uncertain Worlds; Wenjie Zhang, Muhammad Aamir Cheema; Int J Software Informatics, Volume 6, Issue 4 (2012)
[16.] Microeconomic analysis using dominant relationship analysis; Zhu, Ling; Li, Cuiping; Tung, Anthony; Wang, Shan; Knowledge & Information Systems;Jan2012, Vol. 30 Issue 1, p179