On High Dimensional Skylines

Chee-Yong Chan, H.V. Jagadish, Kian-Lee Tan, Anthony K.H. Tung, and Zhenjie Zhang

National University of Singapore & University of Michigan
{chancy,tankl,atung,zhangzh2}@comp.nus.edu.sg, jag@eecs.umich.edu

Abstract. In many decision-making applications, the skyline query is frequently used to find a set of dominating data points (called skyline points) in a multi-dimensional dataset. In a high-dimensional space skyline points no longer offer any interesting insights as there are too many of them. In this paper, we introduce a novel metric, called *skyline frequency* that compares and ranks the interestingness of data points based on how often they are returned in the skyline when different number of dimensions (i.e., subspaces) are considered. Intuitively, a point with a high skyline frequency is more interesting as it can be dominated on fewer combinations of the dimensions. Thus, the problem becomes one of finding top-k frequent skyline points. But the algorithms thus far proposed for skyline computation typically do not scale well with dimensionality. Moreover, frequent skyline computation requires that skylines be computed for each of an exponential number of subsets of the dimensions. We present efficient approximate algorithms to address these twin difficulties. Our extensive performance study shows that our approximate algorithm can run fast and compute the correct result on large data sets in high-dimensional spaces.

1 Introduction

Consider a tourist who is looking for hotels, in some city, that are cheap and close to the beach. For this skyline query, a hotel H is in the answer set (i.e., the skyline) if there does not exist any hotel in that city that dominates H; i.e., that is both cheaper as well as closer to the beach than H. Our tourist can then tradeoff price with distance from the beach from among the points in this answer set (called *skyline points*). Skyline queries are useful as they define an interesting subset of data points with respect to the dimensions considered, and the problem of efficiently computing skylines has attracted a lot of recent interest (e.g., [2, 3, 13, 9, 18]).

A major drawback of skylines is that, in data sets with many dimensions, the number of skyline points becomes large and no longer offer any interesting insights. The reason is that as the number of dimensions increases, for any point p, it is more likely there exists another point q where p and q are better than each other over different subsets of dimensions. If our tourist, from the example in the preceding paragraph, cared not just about price and distance to beach, but also about the size of room, the star rating, the friendliness of staff, the availability of restaurants etc., then most hotels in the city may have to be included in the skyline answer set since for each hotel there may be no one hotel that beats it on all criteria, even if it beats it on many. Correlations between

Top-10 Frequent Skyline Point, p		Dominating Frequency
Player Name	Season	d(p)
Wilt Chamberlain	1961	1791
Michael Jordan	1986	2266
Michael Jordan	1987	3162
George Mcginnis	1974	4468
Michael Jordan	1988	5854
Bob Mcadoo	1974	6472
Julius Erving	1975	6781
Charles Barkley	1987	8578
Kobe Bryant	2002	9271
Kareem Abdul-Jabbar	1975	9400

Table 1. Top-10 frequent skyline points in NBA data set

dimensions ameliorates this problem somewhat, but does not eliminate it. For example, for the NBA statistics data set [1], which is fairly correlated, a skyline query with respect to all 17 dimensions returns over 1000 points.

To deal with this dimensionality curse, one possibility is to reduce the number of dimensions considered. However, which dimensions to retain is not easy to determine, and at the very least requires intimate knowledge of the application domain. In fact, dimensionality reduction of this sort is a desirable goal in many data management and data mining scenarios, and there has been a great deal of effort expended on trying to do this well, with only limited success. Moreover, choosing different subsets of attributes will result in different points being found in the skyline.

In this paper, we introduce a novel metric, called *skyline frequency*, to compare and rank the interestingness of data points based on how often they are returned in the skyline when different subsets of dimensions are considered. Given a set of n-dimensional data points, the skyline frequency of a data point is determined by the $2^n - 1$ distinct skyline queries, one for each possible non-empty subset of the attributes. Intuitively, a point with a high skyline frequency is more interesting since it can be dominated on fewer combinations of the dimensions. Thus, the problem becomes one of finding top-k frequent skyline points.

Referring once more to the 17-dimension NBA statistics data set that records the performance of all players who have played in the NBA from 1946 to 2003. Each dimension represents a certain “skill”, e.g., number of 3-pointers, number of rebounds, number of blocks, number of fouls, and so on. There are over 17000 tuples, each reflecting a player’s “performance” for a certain year. Note that every player has a tuple for every year he played, so it is possible to have several tuples for one player with different year numbers, like “Michael Jordan in 1986” and “Michael Jordan in 1999”. Table 1 lists the top-10 frequent skyline points (represented by a player and season). The skyline frequency of each point p is given by $2^{17} - d(p) - 1$; where $d(p)$, which is the dominating frequency, represents the number of subspaces for which p is dominated by some other point. Readers who follow basketball will agree that this is a very reasonable set of top basketball players of all time. Clearly, our top-k frequent skyline

Bottom-10 Frequent Skyline Point, p		Dominating Frequency
Player Name	Season	d(p)
Terrell Brandon	2000	130559
John Starks	1991	130304
Allen Leavell	1982	130303
Rich Kelley	1981	130047
Rodney Mccray	1984	129823
Reggie Theus	1990	129727
Jamaal Wilkes	1979	129535
John Williams	1988	129151
Purvis Short	1983	129151
Rasheed Wallace	1999	128863

Table 2. Bottom-10 frequent skyline points in NBA data set

query has the notion of picking “the best of the best”, and is superior to the simpler skyline points (which in this example will mark as equally interesting all 1051 skyline points!).

To further examine this notion of skyline frequency, we selected the least skyline frequency entries among the 1051 entries in the full skyline. The results are shown in Table 2. These players, particularly in the years specified, can hardly be considered all-time greats. Of course, each is a talented player, as one would expect given that these were all in the top 1051 chosen from among all NBA players by the ordinary skyline algorithm.

Unfortunately, skyline computations are not cheap. Given a data set with n dimensions, skyline frequency computation requires $2^n - 1$ skyline queries to be executed. To address this problem, we propose an efficient approximate algorithm that is based on counting the number of dominating subspaces (i.e., the number of subspaces in which a point is not a skyline point). Our scheme is tunable in that we can tradeoff the accuracy of the top-k answers for speed. We have implemented our scheme, and our extensive performance study shows that our method with approximate counting can run fast in very high dimensional data set without sacrificing much on the accuracy.

We make two key contributions in this paper:

- We introduce skyline frequency as a novel and meaningful measure for comparing and ranking skylines.
- We present efficient approximate algorithms for computing *top-k frequent skylines*, which are the top-k data points whose skyline frequencies are the highest.

The rest of this paper is organized as follows. In Section 2 we formally define the key concepts, including frequent skylines and maximal dominating subspaces. Related work is presented in Section 3. In Section 4, we present our proposed algorithms for computing frequent skylines efficiently. We report on the results of an experimental evaluation in Section 5. Finally, we conclude with a discussion of our findings in Section 6. Due to space limitation, proofs of results are omitted.

2 Preliminaries

Given a space S defined by a set of n dimensions $\{d_1, d_2, \ldots, d_n\}$ and a data set D on S, a point $p \in D$ can be represented as $p = (p_1, p_2, \ldots, p_n)$ where every p_i is a value on dimension d_i. Each non-empty subset of S is referred to as a *subspace*. A point $p \in D$ is said to *dominate* another point $q \in D$ on subspace $S' \subseteq S$ if (1) on every dimension $d_i \in S'$, $p_i \leq q_i$; and (2) on at least one dimension $d_j \in S'$, $p_j < q_j$. The *skyline of a space* $S' \subseteq S$ is a set of points $D' \subseteq D$ which can not be dominated by any other point on space S'. That is, $D' = \{p \in D : \nexists q \in D, q \text{ dominates p on space } S'\}$. The points in D' are called *skyline points* on space S'.

Based on the definition of skyline points on a subspace, we define the *skyline frequency* of a point $p \in D$, denoted by $f(p)$, as the number of subspaces in which p is a skyline point. Given S and D, the *top-k frequent skyline points* are the k points in D that no other point in D can have larger skyline frequency than them. A *top-k frequent skyline query* is a query that computes top-k skyline points for a given data set D and space S. A subspace $S' \subseteq S$ is said to be a *dominating subspace* for a data point p if there exists another data point that dominates p on subspace S'. We define the *dominating frequency* of p, denoted by $d(p)$, as the number of dominating subspaces for p. It is easy to see that the skyline frequency $f(p) = 2^n - d(p) - 1$. So, the top-k skyline frequency query can be computed by finding the k points with the smallest dominating frequencies.

Let $DS(q, p)$ denote the set of all subspaces for which a point q dominates another point p. We call $DS(q, p)$ the set of *dominating subspaces* of q over p. This set can frequently be quite large, and so is unwieldy to enumerate explicitly. Just as a rectangle in cartesian geometry can be represented succinctly by a pair of corner points, we show below in Lemma 1 that the set $DS(q, p)$ can be described succinctly by a pair of subspaces (U, V) where (1) $U \subseteq S$ is the set of dimensions such that $q_i < p_i$ on every dimension $d_i \in U$; and (2) $V \subseteq S$ is the set of dimensions such that $q_i = p_i$ on every dimension $d_i \in V$. It follows that $DS(p, q) = (S - U - V, V)$.

Lemma 1. *Let $DS(q, p) = (U, V)$. Then $S' \in DS(q, p)$ if and only if $\exists U' \subseteq U$, $V' \subseteq V$, such that $S' = U' \cup V'$, and $U' \neq \emptyset$.*

It is easy to verify that $|DS(q, p)| = (2^{|U|} - 1)2^{|V|}$.

Given two collections of subspaces $S_1, S_2 \subseteq S$, we say that S_1 *covers* S_2 if $S_1 \supseteq S_2$. The following result provides a very simple way to determine if $DS(q, p)$ covers $DS(r, p)$ given three points p, q, and r.

Lemma 2. *Let $DS(q, p) = (U_q, V_q)$ and $DS(r, p) = (U_r, V_r)$, where $U_q \neq \emptyset$ and $U_r \neq \emptyset$. Then $DS(q, p)$ covers $DS(r, p)$ if and only if (1) $U_r \cup V_r \subseteq U_q \cup V_q$ and (2) $U_r \subseteq U_q$.*

$DS(q, p)$ is said to be a *maximal dominating subspace set* for a point p if there does not exist another point r such that $DS(r, p)$ covers $DS(q, p)$. Therefore, $d(p) = |\bigcup_{M_i \in \mathcal{M}} M_i|$, where $\mathcal{M} = \{DS(q, p) \mid q \in S,\ DS(q, p) \text{ is a maximal dominating subspace set for } p\}$.

3 Related Work

Computing the skyline of a set of points is also known as the maximum vector problem [10]. Early works on solving the maximum vector problem typically assume that the points fit into the main memory. Algorithms devised include divide-and-conquer paradigm [10], parallel algorithms [17] and those that are specifically designed to target at 2 or very large number of dimensions [12]. Other related problems include top k [4], nearest neighbor search [16], convex hull [16], and multi-objective optimization [14]. These related problems and their relationship to skyline queries have been discussed in [3].

Börzsönyi et al. [3] first introduced the skyline operator into relational database systems by extending the SQL SELECT statement with an optional SKYLINE OF clause. A large number of algorithms have been developed to compute skyline queries. These can be categorized into non-index-based (e.g., *block nested loop* [3], *Sort-Filter-Skyline* [6, 7], *divide and conquer* [3]), and index-based (e.g., *B-tree* [3], *bitmap* [18], *index* [18], *nearest neighbor* [9], *BBS* [13]). As expected, the non-index-based strategies are typically inferior to the index-based strategies. It also turns out the index-based schemes can progressively return answers without having to scan the entire data input. The nearest neighbour scheme, which applies the divide and conquer framework on datasets indexed by R-trees, was shown to be superior over earlier schemes in terms of overall performance [9]. There have also been work on processing skyline queries over distributed sources [2], over streaming data [11], and for data with partially-ordered domains [5]. All these algorithms are developed for computing skylines for a specific subspace.

The recent papers on skyline computation in subspaces [19, 15] is more closely related to our work. Yuan et al. [19] proposed two methods to compute skylines in all the subspaces by traversing the lattice of subspaces either in a top-down or bottom-up manner. In the bottom-up approach, the skylines in a subspace are partly derived by merging the skylines from its “child” subspaces at the lower level. In the top-down approach, the sharing-partition-and-merge and sharing-parent property of the DC algorithm [3] is exploited to recursively enumerate the subspaces and compute their skylines from the top to bottom level, which turns out to be much more efficient than the bottom-up approach. Since we can get the skyline frequencies if the skylines in every subspace is available, we compare their top-down approach with our top-k method in the performance study. Another study on computing skylines in subspaces is by Pei et al. [15]. They introduced a new concept called skyline group, every entry of which contains the skyline points sharing the same values in a corresponding subspace collection. They also proposed an algorithm *skyey*, which visits all the subspaces along an enumeration tree, finds the skylines by sorting and creates a new skyline group if some new skyline points are inserted into an old group. The skyline groups found are maintained in a quotient cube structure for queries on subspace skyline. Their study tries to answer where and why a point is part of skyline without any accompanying coincident points. However, their scheme can not help to solve the skyline frequency problem since a point can be in exponential number of skyline groups in high dimensional space.

The approximate counting technique used in our work is related to the problem of counting the number of assignments that satisfies a given disjunctive normal form(DNF).

In [8], Karp et. al. proposed a monte-carlo algorithm which takes $2n \ln \frac{2}{\delta}/\epsilon^2$ samples to give an approximate count of the assignments, whose error rate is smaller than ϵ with probability $1 - \delta$. Since the sample size is irrelevant to the size of the sets, this method is much more efficient than the conventional iteration method, especially when the size of valid assignment set is much larger than sample number.

4 Top-k Frequent Skyline Computation

The most straightforward approach to compute top-k frequent skylines is the following two-phase approach. First, compute the skyline points for each subspace by using an existing algorithm (e.g., skycube algorithm [19]). Next, compute the skyline frequency of each point p by summing up the number of subspaces for which p is a skyline. We called this technique a *subspace-based* approach since it essentially enumerates each subspace to compute skylines. A number of recent approaches have been proposed for computing precise skylines for the complete collection of subspaces [19, 15].

However, computing skylines over all subspaces can be costly. In this paper, we propose a novel approach to compute top-k frequent skylines based on computing *maximal dominating subspace sets*. This approach comprises of two key steps. The first step computes the maximal dominating subspace sets for each data point. Based on these, the second step then computes each point's dominating frequency either precisely or approximately. Thus, our approach actually computes the top-k skyline frequencies by computing the bottom-k dominating frequencies.

In the rest of this section, we first give an overview of our approach in Section 4.1, and then present the details of the two phases, maximal dominating subspace set computation and dominating subspace counting, in Sections 4.2 and 4.3, respectively. Finally, we present two optimization techniques to improve the efficiency of our approach in Section 4.4.

4.1 Overview

The intuition for our approach is based on the result in Section 2 that each dominating subspace of a point p is covered by some maximal dominating subspace set of p. Since the dominating frequency of a point is the dual of its skyline frequency, we can compute the skyline frequency of a point p by computing its dominating frequency in two stages. First, find all the maximal dominating subspace sets of p, and then count the number of subspaces covered by them. The top-k frequent skyline points is then obtained by taking the bottom-k points with the lowest dominating frequencies.

The main procedure of our approach is shown in Algorithm 1 which takes a set of data points D, a set of dimensions S, and an integer value k as inputs and computes the top-k frequent skylines in D w.r.t. S. To avoid the complexity of explicitly sorting the points by their dominating frequencies, we maintain a frequency threshold (denoted by θ) that keeps track of the k^{th} smallest dominating frequency among all the processed points. This frequency threshold is initialized in step 1 to $2^{|S|}-1$, which is the maximum possible dominating frequency value. The top-k frequent skylines are maintained in a

Algorithm 1 Top-k Frequent Skyline Algorithm (D, S, k)

```
1:  initialize frequency threshold $\theta = 2^{|S|} - 1$
2:  initialize $R$, the set of top-$k$ frequent skylines, to be empty
3:  for every point $p \in D$ do
4:      $\mathcal{M}$ = ComputeMaxSubspaceSets ($D$, $S$, $p$, $k$, $\theta$, $|R|$)
5:      $d(p)$ = CountDominatingSubspaces ($\mathcal{M}$)
6:      if $(|R| < k)$ or $(d(p) < \theta)$ then
7:          remove the point with the highest dominating frequency in $R$ if $|R| = k$
8:          insert $p$ into $R$
9:          update $\theta$ to be the highest dominating frequency in $R$
10:     end if
11: end for
12: return $R$
```

set R which is initialized to empty in step 2. For each data point $p \in D$ (steps 3-11), the procedure `ComputeMaxSubspaceSets` is first invoked to compute the set $\mathcal{M}$ of all the maximal dominating subspace sets of point p by comparing every other point with point p on all the dimensions. Next, the procedure `CountDominatingSubspaces` is called to compute the dominating frequency $d(p)$ of p, which is the total number of subspaces in S that are covered by the maximal dominating subspace sets in $\mathcal{M}$. If R has fewer than k skylines or if the dominating frequency of p (i.e., $d(p)$) is smaller than the frequency threshold θ, then p is inserted into R and the value of θ updated. Note that if R already has k skylines before a new point is to be inserted, than a point q in R with the largest dominating frequency (i.e., $d(q) = \theta$) is removed from R.

Example 1. Consider the computation of the top-2 frequent skylines for a set of 4-dimensional data points $D = \{a, b, c, e\}$ shown below:

Point	d_1	d_2	d_3	d_4
a	2	3	4	5
b	1	5	2	6
c	3	4	4	4
e	4	3	4	3

To compute the set of maximal dominating subspace sets of point a, we need to determine $DS(q, a)$ for each $q \in D - \{a\}$. We have $DS(b, a) = (\{d_1, d_3\}, \emptyset)$, $DS(c, a) = (\{d_4\}, \{d_3\})$, and $DS(e, a) = (\{d_4\}, \{d_2, d_3\})$. By Lemma 2, $DS(e, a)$ covers $DS(c, a)$, and so $DS(c, a)$ is not a maximal dominating subspace set of a. However, since neither $DS(b, a)$ nor $DS(e, a)$ covers each other, they are both maximal dominating subspace sets of a. The number of dominating subspaces covered by each of them is given by: $|DS(b, a)| = 3$ and $|DS(e, a)| = 4$. Since there are no common dominating subspaces that are covered by both $DS(b, a)$ and $DS(e, a)$, the number of dominating subspaces of a is $d(a) = |DS(b, a)| + |DS(e, a)| = 7$. Similarly, we have $d(b) = 3$, $d(c) = 11$, and $d(e) = 5$. Thus, the top-2 frequent skylines are b and e. □

Algorithm 2 ComputeMaxSubspaceSets (D, S, p, k, θ, r)

```
 1: initialize $\mathcal{M}$, the set of maximal dominating subspace sets of $p$, to be empty
 2: for every point $q$ in $D - \{p\}$ do
 3:     let $U \subseteq S$ such that on every dimension $d_i \in U$, $q_i < p_i$
 4:     let $V \subseteq S$ such that on every dimension $d_i \in V$, $q_i = p_i$
 5:     if ($r$ = $k$) and $((2^{|U|} - 1)2^{|V|} \geq \theta)$ then
 6:         return $\{(U, V)\}$
 7:     end if
 8:     initialize isMaximal = true
 9:     for every maximal dominating subspace set $(P, Q) \in \mathcal{M}$ do
10:         if ($U \cup V \subseteq P \cup Q$) and ($U \subseteq P$) then
11:             isMaximal = false
12:             break out of for loop
13:         else if ($P \cup Q \subseteq U \cup V$) and ($P \subseteq U$) then
14:             remove $(P, Q)$ from $\mathcal{M}$
15:         end if
16:     end for
17:     if isMaximal then
18:         insert $(U, V)$ into $\mathcal{M}$
19:     end if
20: end for
21: return $\mathcal{M}$
```

4.2 Maximal Dominating Subspace Computation

Algorithm 2 shows the `ComputeMaxSubspaceSets` procedure to compute the collection of maximal dominating subspace sets of an input point $p \in D$ (w.r.t. a set of dimensions S). The remaining three input parameters (k, θ, and r), where θ is the highest dominating frequency among all the r frequent skylines processed so far, are used to optimize the computation when p is determined to be not among the top-k frequent skylines. The output collection of maximal dominating subspace sets is maintained in a set M which is initialized to be empty in step 1. Each maximal dominating subspace set in M is represented in the form of a subspace pair; i.e., $M = \{(U_1, V_2), (U_2, V_2), \cdots, (U_n, V_n)\}$, where each (U_i, V_i) corresponds to $DS(q_i, p)$ for some point $q_i \in D$.

To compute the maximal dominating subspace sets of p, the algorithm compares p against each other point q in D (steps 2-20). First, $DS(q, p) = (U, V)$ is determined in steps 3-4. Steps 5-7 is an optimization (to be explained at the end of the discussion) that can be ignored for now. Steps 8-19 compare (U, V) against each of the maximal dominating subspace sets computed so far in $\mathcal{M}$ to determine if (U, V) is also a maximal dominating subspace set and update $\mathcal{M}$ accordingly. Specifically, if there is some subspace set $(P, Q) \in \mathcal{M}$ that covers (U, V), then by Lemma 2, we can conclude that (U, V) is not a maximal dominating subspace set (steps 11-12). On the other hand, if subspace set $(P, Q) \in \mathcal{M}$ is covered by (U, V), then (P, Q) is not a maximal dominating subspace set and is removed from $\mathcal{M}$ (step 14). Finally, if (U, V) is not covered by any of the maximal dominating subspace sets in $\mathcal{M}$, then (U, V) is a maximal dominating subspace set and it is added to $\mathcal{M}$ (step 18).

Algorithm 3 CountDominatingSubspaces ($\mathcal{M}$)

```
let $\mathcal{M} = \{M_1, M_2, \cdots, M_n\}$
initialize counter $C = 0$
for $i$ = 1 to $n$ do
    for every dominating subspace $(P, Q)$ that is covered by $M_i$ do
        if $(P, Q)$ is not covered by any $M_j$, $j \in [1, i)$ then
            $C = C + 1$
        end if
    end for
end for
return $C$
```

We now explain the optimization performed in steps 5-7 that makes use of the additional input parameters k, θ, and r. The main idea is to avoid computing the precise collection of maximal dominating subspace sets of p if p is determined to be not among the top-k frequent skylines. Specifically, if there are already k intermediate frequent skylines (i.e., $r = k$) and $|DS(q,p)|$, which is given by $(2^{|U|} - 1)2^{|V|}$, already exceeds θ, then p clearly can not be among the top-k frequent skylines. In this case, it is not necessary to know the precise maximal dominating subspace sets of p; instead, the algorithm simply returns the single subspace set (U, V) (in step 6) since this is sufficient for the main algorithm to conclude that p is not a top-k frequent skyline. With this optimization, `ComputeMaxSubspaceSets` computes the precise collection of maximal dominating subspace sets of p only when p could potentially be a top-k frequent skyline.

Our implementation of `ComputeMaxSubspaceSets` uses a bitmap representation for subspaces to enable efficient manipulations. If S has n dimensions, then a subspace of S is represented by a n-bit bitmap with the i^{th} bit corresponding to dimension d_i such that a bit is set to 1 iff its corresponding dimension is in the subspace. As an example, in an 8-dimensional space S, the subspace $\{d_1, d_3, d_5, d_6\}$ is represented by the bitmap "10101100". Given two bitmaps B_1 and B_2 (corresponding to subspaces S_1 and S_2, respectively), S_1 covers S_2 if and only if the logical-AND of B_1 and B_2 is equal to B_1. Furthermore, by exploiting arithmetic bit-operation, $|DS(U, V)|$ for a given subspace set (U, V) can be efficiently computed with a left shift operation in $O(1)$ time.

4.3 Dominating Subspace Counting

In this section, we discuss how to derive the number of dominating subspaces for a point p based on the collection $\mathcal{M}$ of maximal dominating subspace sets for p returned by `ComputeMaxSubspaceSets` for p. Since there is usually more than one maximal dominating subspace set in $\mathcal{M}$ and the subspaces covered by them generally overlap, the challenge is to efficiently compute the number of dominating subspace sets taking into account of the overlapping covered subspaces.

As an example, consider $\mathcal{M} = \{M_1, M_2\}$, where $M_1 = (\{d_1, d_2\}, \{d_3\})$ and $M_2 = (\{d_1, d_3\}, \{d_4\})$. Note that there are a total of eight dominating subspaces covered by $\mathcal{M}$: $\{d_2\}$, $\{d_2, d_3\}$, $\{d_1, d_2\}$, $\{d_1, d_2, d_3\}$, $\{d_1\}$, $\{d_1, d_3\}$, $\{d_1, d_4\}$ and $\{d_1, d_3, d_4\}$. Among these, the first six are covered by M_1 while the last four are covered by M_2; hence, there are two dominating subspaces (i.e., $\{d_1\}$ and $\{d_1, d_3\}$) that are covered by both M_1 and M_2.

One direct approach to derive the number of dominating subspaces is to apply the *Inclusion-Exclusion* principle to obtain the union of all the subspaces covered by the maximal dominating subspaces. However, this method is non-trivial as it requires enumerating all the subspaces covered by each maximal dominating subspace and checking if the enumerated subspace has already been previously generated. In the following, we propose two alternative methods based on precise counting and approximate counting, respectively, for counting the number of dominating subspaces covered by $\mathcal{M}$.

Precise Counting Our improved approach for computing the exact number of dominating subspaces is shown in Algorithm 3. For each maximal dominating subspace set $M_i \in \mathcal{M}$, let S_i denote the collection of dominating subspaces that are covered by M_i. We define for each S_i, a new subspace collection (denoted by S'_i) as follows: $S'_i = S_i - \bigcup_{j \in [1,i)} S_j$. It is easy to verify that (1) $\bigcup_{M_i \in \mathcal{M}} S_i = \bigcup_{M_i \in \mathcal{M}} S'_i$; and (2) $S'_i \cap S'_j = \emptyset$ for any distinct pair S'_i and S'_j. In this way, we transform the problem of counting the union of a collection of sets to a subset counting problem without any intersection among the subsets. For every maximal dominating subspace set $M_i \in \mathcal{M}$, we enumerate over each of the subspaces covered by M_i and check whether it is also covered by an earlier maximal dominating subspace set M_j, $j \in [1, i)$. Referring to the preceding example with $\mathcal{M} = \{M_1, M_2\}$, we have $S'_1 = \{\{d_1\}, \{d_1, d_3\}, \{d_2\}, \{d_2, d_3\}, \{d_1, d_2\}, \{d_1, d_2, d_3\}\}$, and $S'_2 = \{\{d_1, d_4\}, \{d_1, d_3, d_4\}\}$.

However, the simple precise counting method can not scale efficiently to handle high-dimensional spaces because we still need to enumerate all the $(2^{|U|} - 1)2^{|V|}$ subspaces for a maximal dominating subspace (U, V). For example, with $|U| = 20$, over one million of subspaces need to be compared against with every previous maximal dominating subspace.

Approximate Counting To avoid the high complexity of the precise counting approach, we present an effective approximate counting method that is based on extending a Monte-Carlo counting algorithm [8] originally proposed for counting the number of assignments that satisfy a specified DNF formula, which is a #P-complete problem.

Our approach is shown in Algorithm 4 which takes three input parameters ($\mathcal{M}$, ϵ, and δ) and returns an approximate count of the number of dominating subspaces covered by a collection $\mathcal{M}$ of maximal dominating subspace sets for some point. The approximate answer is within an error of ϵ with a confidence level of at least $1 - \delta$. Steps 1-6 first compute the number of subspaces (denoted by N_i) covered by each maximal dominating subspace set $M_i \in \mathcal{M}$, and the total number of these (possibly overlapping) subspaces denoted by N. To obtain the desired error bound, a random repeatable sample of $T = 2n \ln(2/\delta)/\epsilon^2$ number of maximal dominating subspace sets is selected from $\mathcal{M}$, where the probability of sampling M_i is proportional to the number of subspaces covered by M_i. For each generated maximal dominating subspace set M_i, a dominating subspace set (U, V) that is covered by M_i is randomly selected and checked if it is also

Algorithm 4 ApproxCountDominatingSubspaces ($\mathcal{M}$, ϵ, δ)

```
let $\mathcal{M} = \{M_1, M_2, \cdots, M_n\}$
for $i$ = 1 to $n$ do
    let $M_i = (U_i, V_i)$
    $N_i = (2^{|U_i|} - 1)2^{|V_i|}$
end for
$N = \sum_{M_i \in \mathcal{M}} N_i$
$T = 2n\ln(2/\delta)/\epsilon^2$
initialize $C = 0$
for $i$ = 1 to $T$ do
    choose a maximal dominating subspace set $M_i$ with probability $N_i/N$
    choose a subspace set $(U, V)$ that is covered by $M_i$ with equal probability
    if $(U, V)$ is not covered by any $M_j$, $j \in [1, i)$ then
        $C = C + 1$
    end if
end for
return $N \cdot C/T$
```

covered by any maximal dominating subspace sets $M_j, j \in [1, i)$. A counter, denoted by C, is used to keep track of the number of distinct dominating subspaces determined from this sampling process. The approximate count output by the algorithm is given $(N \times C)/T$; the proof of the error bound follows from [8].

Complexity Analysis Let $\mathcal{M} = \{(U_1, V_1), (U_2, V_2), \ldots, (U_n, V_n))\}$. We use U_m, V_m, U_a and V_a to denote $\max_{1 \le i \le n}\{|U_i|\}$, $\max_{1 \le i \le n}\{|V_i|\}$, $\sum_{i=1}^{n} |U_i|/n$, and $\sum_{i=1}^{n} |V_i|/n$, respectively.

In the exact counting algorithm, since each covered subspace for a maximal dominating subspace set (U_i, V_i) must be compared with the previous maximal dominating subspace sets, the computation complexity for (U_i, V_i) is $(i-1)(2^{|U_i|-1})2^{|V_i|}$. Therefore, the total time complexity of the exact counting algorithm is $\sum(i-1)(2^{|U_i|-1})2^{|V_i|} = O(n^2 2^{U_m+V_m})$. Note that by Jensen's Inequality, $\sum(i-1)(2^{|U_i|-1})2^{|V_i|} = \Omega(n2^{U_a+V_a-1})$.

In the approximate counting algorithm, the sampling process is independent of $|U_i|$ and $|V_i|$. Since there are a total of $2n\ln(2/\delta)/\epsilon^2$ subspace sets sampled, the upper and lower bounds on the computation complexity of the approximate counting approach are $O(2n^2\ln(2/\delta)/\epsilon^2)$ and $\Omega(2n\ln(2/\delta)/\epsilon^2)$, respectively.

With the above analysis, it is not difficult to verify that the exact counting method can not be slower than approximate counting method in constant factor when $U_m + V_m \le \ln\ln(2/\delta) + 2\ln(1/\epsilon) - \ln n + 1$, while the approximate counting method can not be slower than exact counting method when $U_a + V_a \ge \ln\ln(2/\delta) + 2\ln(1/\epsilon) + \ln n + 2$.

4.4 Optimizations

In this section, we present two optimizations to further improve the performance of the `ComputeMaxSubspaceSets` algorithm presented in Section 4.2. In the current `ComputeMaxSubspaceSets` algorithm, the main optimization relies on using the frequency threshold θ (steps 5-7) as a quick filtering test to check whether a point is guaranteed to be not among the top-k frequent skylines. Clearly, it is desirable to prune out points that are not top-k frequent skylines as early as possible using this efficient checking to reduce the unnecessary elaborate enumeration and comparison performed in steps 8-19.

Pre-Sorting Our first optimization is based on the observation that the effectiveness of the pruning test is dependent on the order in which the data points are processed. For example, no early pruning would be possible if the points are processed in non-descending order of their skyline frequencies. One idea to maximize the pruning effectiveness is to first sort the data points based on some simple criterion such that points that have higher potential to be top-k frequent skylines appear earlier. Our optimization simply sorts the points in non-descending order of the sum of their dimension values. The intuition behind this heuristic is that a point with a smaller sum is likely to have smaller values on more dimensions and is therefore likely to have a higher skyline frequency. A similar idea was previously used in [6, 15, 19].

Checkpoint Our second optimization aims to generalize the pruning test to improve its effectiveness. Currently, the pruning test for a point p is applied in the context of a single maximal dominating subspace set (i.e., $DS(q,p)$ for some $q \in D$). However, when the number of maximal dominating subspace sets is large, it is possible that each maximal dominating subspace set in $\mathcal{M}$ on its own does not cover too many dominating subspaces (to cause p to be pruned) even though the collection of dominating subspaces covered by $\mathcal{M}$ as a whole is large.

To overcome this limitation, we extend the pruning test to be done at several "checkpoints" by invoking `CountDominatingSubspaces` to count the number of dominating subspaces at intermediate stages and performing the pruning tests using intermediate collections of $\mathcal{M}$ each of which generally consists of more than one maximal dominating subspace set. Thus, by counting the coverage for multiple maximal dominating subspace sets rather than a single maximal dominating subspace set, the opportunity for pruning is increased.

In the implementation of this optimization, we set checkpoints at exponential sizes; i.e., when the number of maximal dominating subspace sets reaches 2^t (for some $t > 0$), the counting process is invoked to check whether the current number of subspaces covered has already exceeded the threshold. This exponential checkpoint setup turns out to perform better than any "linear" checkpoint setup since the number of subspaces covered is usually proportional to the number of maximal dominating subspace sets.

5 Performance Study

In this section, we present an experimental evaluation of our proposed algorithms for computing top-k frequent skylines using both synthetic as well as real data sets.

5.1 Experimental Setup

We generated synthetic data sets by varying the number of dimensions, the size of the data set and the distributions of the data set; in particular, we considered the three commonly used types of data distributions: independent, correlated, and anti-correlated. In addition, we also conducted experiments on the NBA real data set [1] that is mentioned throughout this paper. The characteristics of this real data set most closely resembles a correlated data distribution.

We compare the performance of the following four algorithm variants:

1. **Exact Count (EC)**: This scheme adopts exact counting, and employs the Pre-Sorting and Checkpoint optimizations.
2. **Approx Count without Sorting (ACWS)**: This scheme uses approximate counting and only the Checkpoint optimization.
3. **Approx Count without Checkpoint (ACWC)**: This scheme employs approximate counting with only the Pre-Sorting optimization.
4. **Approx Count(AC)**: This scheme adopts approximate counting together with both Pre-Sorting and Checkpoint optimizations.

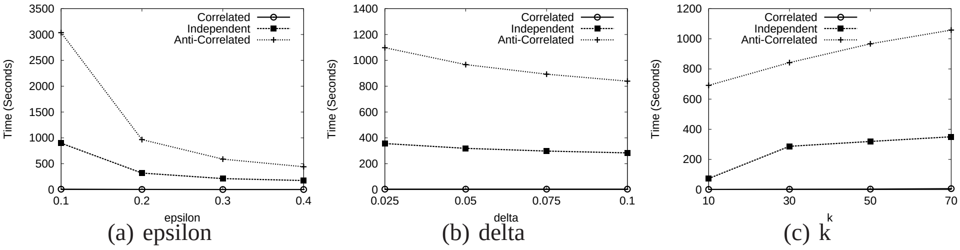
All experiments were carried out on a PC with a 2 GHz AMD Athlon processor and 2 GB of main memory running the Linux operating system. Unless otherwise stated, we use the following default setting in our study: 15-dimensional data set with 100K records, $\epsilon = 0.2$, $k = 10$, and $\delta = 0.05$. The default algorithm for all experiments is AC, which we expect to show is the algorithm of choice.

5.2 Tuning the Approximate Counting Scheme

There are several tunable parameters in the approximate counting scheme: ϵ, δ and k. We study the relationship between the effect of these parameters on efficiency and precision. The efficiency result is shown in Fig. 1, while the precison result is shown in Fig. 2.

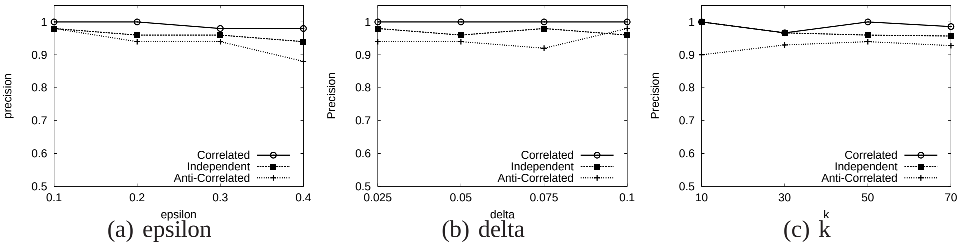
We first discuss the efficiency results which compare the computation time as a function of different parameters. When we vary ϵ from 0.1 to 0.4 in AC, the processing time decreases greatly since the number of samples is quadratic to $1/\epsilon$ in approximate counting. When we vary δ from 0.025 to 0.1 in AC, the processing time is very stable since the number of samples in approximate counting is linear to $\ln(2/\delta)$, which does not change much with δ. From Fig. 1(c), which shows the result when k is varied from 10 to 70, we can see that the increase trend of the processing time is almost linear to the result size k, which indicates that AC is scalable to various values of k.

The effectiveness of the method is measured by precision, which is the ratio between the number of true top-k frequent skylines and the result size k. Looking at Fig. 2, we note that the precision on correlated data set is always close to 1. Even on independent and anti-correlated data sets, the AC algorithm can achieve precision over 90% with a large range of different parameters. The figure also indicates that ϵ is the most important factor affecting the precision of the result. The precision decreases monotonically with the increase of ϵ, while the other two parameters, δ and k, do not have too much impact on the precision.



(a) epsilon (b) delta (c) k

Fig. 1. Efficiency comparison as a function of different parameters



(a) epsilon (b) delta (c) k

Fig. 2. Precision comparison as a function of different parameters

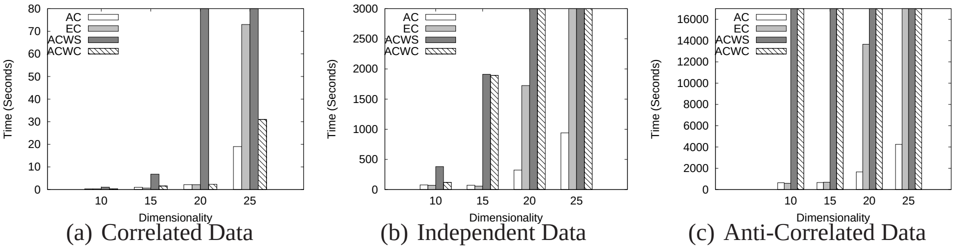
From this experiment, we can conclude that even setting $\epsilon = 0.2$ and $\delta = 0.05$ is enough to provide very good results for top-k frequent skyline queries. As such, we use these as the default setting.

5.3 Effect of number of dimensions

We study the impact of dimensionality on the efficiency of the algorithms. We compare all the four algorithms EC, AC, ACWS and ACWC on data sets ranging from 10-25 dimensions. The results on the three synthetic data sets are shown in Fig. 3.

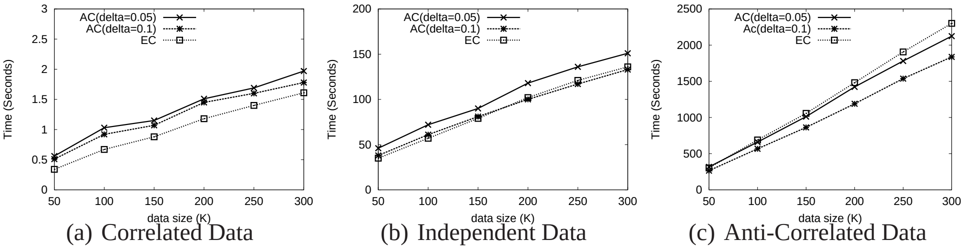
First, we look at the three approximate counting schemes. In the figure, the “bars” that are beyond the maximum time plotted are truncated (in other words, all “bars” with the maximum value have much larger value than the maximum value plotted). From the poor performance of ACWS and ACWC, we can see the effect of the pre-sorting and checkpoint optimizations. It is clear that pre-sorting is an important optimization. Without pre-sorting, the efficiency decreases by at least one order of magnitude. The checkpoint optimization is useful when the dimensions are independent or anti-correlated since it can prune many points. The combined effect of both optimizations contributes to the superior performance of AC.

Now, comparing AC and EC, we observe that EC slightly outperforms AC at low dimensionality (< 15). This is because when the dimensionality is low, the subspaces covered by those maximal subspace sets are fewer than the number of samples needed



(a) Correlated Data (b) Independent Data (c) Anti-Correlated Data

Fig. 3. Efficiency comparison on varying dimensionality



(a) Correlated Data (b) Independent Data (c) Anti-Correlated Data

Fig. 4. Efficiency comparison on varying data size

by AC. However, when the dimensionality is high, AC shows its strength since the number of samples is irrelevant to the dimensionality, while EC must enumerate all the covered subspaces whose number is exponential to the dimensionality.

Third, looking at the figure, we see that the relative performance of the schemes remain largely unchanged under different distributions. As expected, the computation time is higher for all schemes when the data becomes more anti-correlated.

Since ACWS and ACWC perform poorly relative to AC, we shall not discuss them further.

5.4 Effect of data set cardinality

In this experiment, we evaluate the impact of data size on the computation efficiency for 15-dimensional data. The data size is varied from 50K to 300K. We study two variants of AC: $\delta = 0.05$ and $\delta = 0.1$. The results in Fig. 4 show that although the time complexity of Algorithm 1 is theoretically quadratic to the data size in the worst case, the actual efficiency of these methods is almost linear in the data size. On the correlated data set, EC always outperforms the algorithms with approximate counting. This is because the dominating frequencies of the top-k points are all very small when the data is correlated. For the data set with independent dimensions, EC outperforms AC ($\delta = 0.05$) but only outperforms AC ($\delta = 0.1$) when the data size is smaller than 200K. For the anti-correlated data set, both AC ($\delta = 0.05$) and AC ($\delta = 0.1$) are faster than EC since the dominating frequency is large enough.

5.5 Results on Real Data Set

We use the NBA player statistics data set as our real data set for this experiment. As noted, there are 17266 tuples over 17 dimensions.

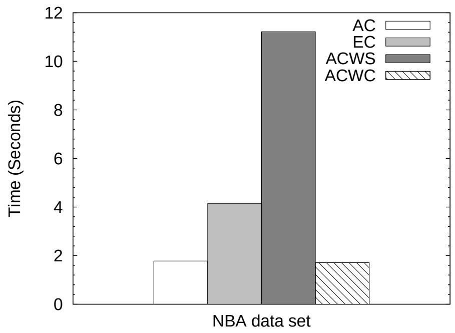


Fig. 5. Efficiency comparison on NBA data set

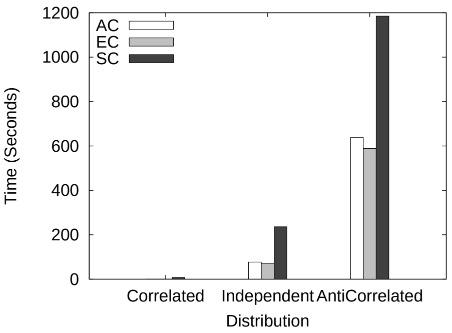


Fig. 6. Efficiency comparison with Skycube algorithm

Fig. 5 compares the efficiency of the four algorithms, AC, EC, ACWS and ACWC. From the figure, we can see that ACWC outperforms all the other methods on the real data set; this is due to the fact that the NBA data set is fairly correlated. Although AC is slower than ACWC by a little (due to the cost of the unnecessary checkpoints), its performance is still much better than that of the exact counting algorithm.

5.6 Comparison of number of maximal dominating subspaces

Table 3 compares the number of maximal dominating subspaces that are maintained by the various algorithms for different number of data dimensions and data distributions (i.e., correlated, independent, and anti-correlated). The last column in the table lists the upper bounds on the number of maximal dominating subspace sets. Note that for a data set with D dimensions, the upper bound is given by $\binom{D}{\lceil D/2 \rceil}$, where each maximal dominating subspace consists of $\lceil D/2 \rceil$ dimensions.

As expected, the points in the anti-correlated data set has the maximum number of maximal dominating subspaces. However, the number of maximal dominating subspace is still much smaller than the theoretical upper bound listed in the last column. This indicates that our method does not suffer from the exponential increase of dimensionality in practice.

5.7 Comparative study

We also compared our EC and AC schemes against the Skycube algorithm [19]. Although the Skycube algorithm (denoted as SC) can find the precise top-k frequent skylines, it does not scale beyond 15 dimensions. Our results show that it takes more than

Dimensionality	Correlated	Independent	Anti-Correlated	Upper Bound
10	17	64	62	252
15	72	483	565	6435
20	188	1897	2477	184756
25	587	5119	7617	5200300

Table 3. Comparison of number of maximal dominating subspaces

10 hours for SC to run on the independent data set with 100K 15-dimensional points. This is because SC focuses on conventional skyline query in any specified subspace, and thus spends most of its computation time on points which cannot be top frequent skyline points. As such, we only present the results for 100K 10-dimensional data sets in Fig. 6.

From the figure, it is clear that both EC and AC are superior to SC in all the three types of data sets. SC is not scalable as it may need to compute all the subspaces which is exponential in the number of dimensions. We note that for small number of dimensions (10 in this case), our AC scheme returns the exact answers, i.e., it has 100% precision. That is, we do not give up any precision loss to obtain performance gain in this case.

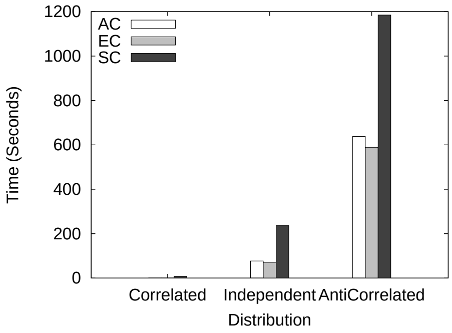


Fig. 7. Efficiency Comparison on 10D

6 Conclusions

Skyline queries have been lauded for their ability to find the most interesting points in a data set. However, in high dimensional data sets, there are too many skyline points for them to be of practical value. In this paper, we introduced skyline frequency as a measure of interestingness for points in the data set. The skyline frequency of a point measures the number of subspaces in which the point is a skyline. We developed an efficient approximation algorithm to compute the top-k frequent skyline query. Our experimental study demonstrated the performance and the effectiveness of the proposed algorithm.

We plan to extend this work in several directions. First, we would like to explore precomputation techniques (e.g., indexes) to further speed up the computation of top-k frequent skyline query. Second, our current work assumes a static data set. We would like to study techniques to facilitate incremental updates. Finally, exploring other interestingness measures of skyline points is also part of our future work.

Acknowledgement: We thank the authors of [19] for sharing their implementation of the Skycube algorithm.

References

1. NBA basketball statistics. http://databasebasketball.com/stats.download.
2. W. Balke, U. Güntzer, and X. Zheng. Efficient distributed skylining for web information systems. In *EDBT'04*.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE'01*.
4. M. Carey and D. Kossmann. On saying "enough already!" in SQL. In *SIGMOD'97*.
5. C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD'05*.
6. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE'03*.
7. P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB'05*.
8. R. M. Kapp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.
9. D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB'02*.
10. H.T. Kung, F. Luccio, and F.P. Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4), 1975.
11. X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: efficient skyline computation over sliding windows. In *ICDE'05*.
12. J. Matousek. Computing dominances in E^n. *Information Processing Letters*, 38(5):277–278, 1991.
13. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD'03*.
14. C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *PODS'01*.
15. J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: a semantic approach based on decisive subspaces. In *VLDB'05*.
16. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
17. I. Stojmenovic and M. Miyakawa. An optimal parallel algorithm for solving the maximal elements problem in the plane. *Parallel Computing*, 7(2), June 1988.
18. K.-L. Tan, P.-K. Eng, and B.C. Ooi. Efficient progressive skyline computation. In *VLDB'01*.
19. Y. Yuan, X. Lin, Q. Liu, W. Wang, J.X. Yu, and Q. Zhang. Efficient computation of skyline cube. In *VLDB'05*.