Extract Interesting Skyline Points in High Dimension

Gabriel Pui Cheong Fung[1], Wei Lu[2,3], Jing Yang[3], Xiaoyong Du[2,3], and Xiaofang Zhou[1]

[1] School of ITEE, The University of Queensland, Australia
{g.fung,zxf}@uq.edu.au
[2] Key Labs of Data Engineering and Knowledge Engineering, Ministry of Education, China
[3] School of Information, Renmin University of China, China
{uqwlu,jyang,duyong}@ruc.edu.cn*

Abstract. When the dimensionality of dataset increases slightly, the number of skyline points increases dramatically as it is usually unlikely for a point to perform equally good in all dimensions. When the dimensionality is very high, almost all points are skyline points. Extract interesting skyline points in high dimensional space automatically is therefore necessary. From our experiences, in order to decide whether a point is an interesting one or not, we seldom base our decision on only comparing two points pairwisely (as in the situation of skyline identification) but further study how good a point can perform in each dimension. For example, in scholarship assignment problem, the students who are selected for scholarships should never be those who simply perform better than the weakest subjects of some other students (as in the situation of skyline). We should select students whose performance on some subjects are better than a reasonable number of students. In the extreme case, even though a student performs outstanding in just one subject, we may still give her scholarship if she can demonstrate she is extraordinary in that area. In this paper, we formalize this idea and propose a novel concept called k-dominate p-core skyline (C_p^k). C_p^k is a subset of skyline. In order to identify C_p^k efficiently, we propose an effective tree structure called Linked Multiple B'-tree (LMB). With LMB, we can identify C_p^k within a few seconds from a dataset containing 100,000 points and 15 dimensions.

1 Introduction

Given a set of points X in an N dimensional space, a point $x_i \in X$ dominates another point $x_j \in X$ if x_i performs better than x_j in at least one dimension and performs not worse than x_j in all other dimensions [1]. A point which cannot be dominated by any other point is called a skyline point. A collection of skyline points formulates a skyline. By definition, we cannot compare skyline points with each other without an appropriate preference function [2,3] because each skyline point must have at least one dimension performs better than any other skyline point. If the dimensionality is very high, almost all points are skyline points [4], extracting interesting skyline points in high dimensional space is therefore necessary [5,6].

* Wei Lu's and Xiaoyong Du's research is partially supported by National Natural Science Foundation of China under Grant 60873017.

Table 1. Academic results of some students

Names	**Subjects**				
	Language	Music	Math	Sport	Art
Amy	A	A	A	A	C
Billy	A-	A-	A-	C	C+
Cathy	A	A	B	C	A
Dorothy	B	B	B	B	C
George	F	F	F	C+	C+

For the sake of discussion, Table 1 shows a simply toy example about the academic results of five students. The first three students perform good but the last student performs very poor. Unfortunately, under the definition of skyline, the skyline points are: Amy, Billy, Cathy and George (Dorothy is dominated by Amy). This means *these four students are indistinguishable in terms of their academic achievements*. This does not make too much sense – we will never consider George's performance is good. George is a skyline point because *he performs slightly better than the weakest subject of any given student*, despite of *he performs significantly poorer than all students in most subjects*.[1]

1.1 Preliminaries

[5] extract interesting skyline points as follows: a point x_i is said to k-dominate another point x_j if x_i performs better than x_j in at least one dimension and not worse than x_j in k dimensions. A point that cannot be k-dominated by any other point is a k-dominant skyline point. Quite obvious, k-dominant skyline is a subset of skyline but its size is much smaller especially when k is small. In Table 1, Amy and Cathy both k-dominate George if $k \leq 3$. Hence, George, a non-interesting point, will not be included in the k-dominant skyline. k-dominant skyline is proven to be very effective in extracting interesting skyline points. Yet, the k-dominance relationship is not transitive. Furthermore, they do not consider vertical relationship of each dimension (as we will discuss it later). [7] proposes a concept called skyline frequency to extract interesting skyline points. It ranks all points according to the number of subspaces that they will be regarded as skyline points and returns the points at the top ranking. This approach is smart, but have two major issues: (1) Some points that are intuitively more superior may sometimes have the same ranking (or lower ranking) as the inferior points. (2) There are $2^N - 1$ subspaces in an N dimensional space, which is too expensive to compute in practice. For the first issue, let us consider Table 1 again. Billy obviously outperforms George. Unfortunately, they have the same ranking (they are skyline point in 8 out of 31 subspaces). For the second issue, [7] proposes an approximation algorithm to solve it but it cannot answer dynamic queries [8]. [9] proposes an idea called δ-subspace to identify a set of interesting skyline points. δ-subspace is a subspace of the whole dataspace such that its skyline contains less than δ points. The union of the skyline points in all the δ-subspace are

[1] *Note: One may argue that the problem in Table 1 is just a matter of scaling. However, we would like to stress that Table 1 is a simply example to demonstrate the potential problem of skyline in high dimensional space. Readers can refer to other examples from some existing work [5,6] if they are confused about the potential problem of skyline in high dimensional space.*

called strong skyline points. This algorithm depends on δ heavily. If δ too large, most of the skyline points will be removed, and vice versa. Setting δ is difficult for a normal user or without analyzing the dataset throughly. In addition, the physical meaning of this approach is not intuitive, which further discourage it to be used in practice. [10] proposes a skyline operator called top-K representative skyline query. In this operator, K skyline points will be selected such that the number of points which can be dominated by them will be maximized. Our problem is different from them. Firstly, they do not have vertical comparison among dimension. Secondly, superior points may frequently dominate less number of points than those inferior points. Thirdly, their aim is to rank the skyline points. Another related concept to our problem is K-skyband [8,11,12,13], which is a set of points that are dominated by at most K points. Yet, K-skyband does not aim at minimizing the number of skyline points which is different from our motivation. We do not attempt the elaborate them further due to the limited space.

1.2 Motivation and Objective

Let us take a simple example to illustrate our motivation. In scholarship assignment problem, scholarships are usually given to students based on two steps: (1) Identify some potential candidates from all students; (2) Assign scholarships to some of these potential candidates after some discussions. Ideally, potential candidates can be regarded as skyline because no student should perform better than any of them. In practice, if we identify the potential candidates by traditional skyline, the number of potential candidates will be too many when the number of subjects is many (i.e. dimensionality is high). We need to have a mechanism to reduce the size of skyline. For example, George in Table 1 should not be considered as a potential candidate. Logically, a potential candidate should not be a student who simply performs better than the weakest subject of any other student (like George). A potential candidate should has some subjects that she can perform better than a reasonable number of students in those subjects. In the extreme case, even though a student performs outstanding in one subject, she may still award scholarship if she can demonstrate she is extraordinary in that area. Based on this idea, we claim that when we extract interesting points from a large dataset, we should not only make *horizontal pairwise comparisons* like the concept of skyline. We should further study how many points that a given point can dominate according to a given dimension (i.e. *vertical pairwise comparison*). For example, suppose there are n students perform better than a student called Smith in a subject. If Smith wants to claim he is one of the potential candidates for awarding scholarship, then it is reasonable to ask him to identify another subject that he can perform better than *all* these n students. In Table 1, George cannot fulfill this criterion.

In this paper, we formalize this motivation and propose a novel concept called *core skyline*. A core skyline contains a set of interesting points extracted from a skyline. This work is different from the existing works as we further consider the vertical relationships among points. Up to our knowledge, we are the first mover in this area.

Given a point x_i and a dimension d_m let $M(x_i, d_m)$ be a set of points performs better than x_i in d_m. x_i is said to *dominate-back* $M(x_i, d_m)$ if x_i performs better than all points in $M(x_i, d_m)$ in some other dimensions. x_i is a core skyline point if it can dominate-back $M(x_i, d_m)$ for all different d_m in a dataset. Based on this concept, we further propose

Table 2. Some sample datasets

ID	d_1	d_2	d_3	d_4
x_1	1	7	3	5
x_2	3	2	7	7
x_3	5	4	2	3
x_4	4	5	6	6
x_5	7	1	1	1
x_6	6	3	4	2
x_7	2	6	5	4

A. Dataset 1 (Order by ID)

Rank	d_1	d_2	d_3	d_4
1	x_1	x_5	x_5	x_5
2	x_7	x_2	x_3	x_6
3	x_2	x_6	x_1	x_3
4	x_4	x_3	x_6	x_7
5	x_3	x_4	x_7	x_1
6	x_6	x_7	x_4	x_4
7	x_5	x_1	x_2	x_2

B. Dataset 1 (Order by ranking)

Insertion order	**Points**	**Dimensions**		
		d_1	d_2	d_3
1	x_1	7	10	8
2	x_2	7	10	11
3	x_3	16	10	17
4	x_4	18	15	1
5	x_5	12	10	20
6	x_6	7	10	8

C. Dataset 2

a more robust concept called *k-dominant p-core skyline* (C_p^k). k denotes the number of dimensions that x_i has to dominate-back and p denotes the fraction of points in a dimension that x_i has to dominate-back. Details of this formulation will be described in Section 2. We propose an indexing algorithm in Section 3 called Linked Multiple B'-tree (LMB)[2] to help us identify C_p^k dynamically and progressively.

2 Problem Definition

Let $D = \{d_1, d_2, \dots, d_N\}$ be an N-dimensional space and $X = \{x_1, x_2, \dots\}$ be a set of points in D. We use $x_i.d_m$ to denote the value of x_i in d_m. There is a total order relationship in each dimension such that a smaller value is more preferable. Let us review two existing definitions [14]:

Definition 1. (Dominate, $\prec$) *x_i is said to dominate x_j ($x_i \prec x_j$) iff these conditions hold: (1) $\forall d_m \in D, x_i.d_m \leq x_j.d_m$; and (2) $\exists d_m \in D, x_i.d_m < x_j.d_m$.*

Definition 2. (Skyline, S) *Let $S \subseteq X$. x_i is a skyline point ($x_i \in S$) if and only if $\forall x_j \in X, x_j$ cannot dominate x_i.*

We now formally define core skyline and use Table 2A and 2B as a running example. Table 2A contains a dataset with seven points and four dimensions. Table 2B ranks these points according to their values in each dimension. For example, x_1 is ranked highest in d_1 because it has the smallest value in d_1. Note that all points in Table 2A and 2B are skyline points. Before we can define core skyline, we need to define the following:

Definition 3. (Dominant set, $M(x_i, d_m)$) *Let $M(x_i, d_m) \subset X$. $M(x_i, d_m)$ contains points that perform better than or equal to x_i in d_m, i.e. $x_j \in M(x_i, d_m)$ iff $x_j.d_m \leq x_i.d_m$.*

Definition 4. (Dominate-back, $\prec_b$) *Let $M \subset X$. A point x_i is said to dominate-back M ($x_i \prec_b M$) iff $M = \emptyset$ or $\exists d_n$ such that $\forall x_j \in M, x_i.d_n \leq x_j.d_n$.*

In Table 2A, $M(x_1, d_1) = \emptyset$ because no point has a value less than $x_1.d_1$. Similarly, $M(x_1, d_2) = \{x_2, x_3, x_4, x_5, x_6, x_7\}$, $M(x_1, d_3) = \{x_3, x_5\}$ and $M(x_1, d_4) = \{x_3, x_5, x_6, x_7\}$. Given x_i, if $x_i \in S$ and $x_i \prec_b M(x_i, d_m), \forall d_m \in D$, then x_i is a core skyline point:

[2] "B'-tree" is pronounced as "B-pie-tree".

Definition 5. (Core skyline, C) *Let $C \subseteq S$. A point x_i is a core skyline point ($x_i \in C$) iff $x_i \in S$ and $\forall d_m \in D, x_i \prec_b M(x_i, d_m)$.*

In Table 2A, $C = \{x_1, x_5\}$. All other points do not belong to core skyline. For example, let us consider x_4. In order for x_4 to be a core skyline point, x_4 must be able to dominate-back $M(x_4, d_1)$, $M(x_4, d_2)$, $M(x_4, d_3)$ and $M(x_4, d_4)$. Now, let us consider $M(x_4, d_1) = \{x_1, x_2, x_7\}$. In order for $x_4 \prec_b M(x_4, d_1)$, there must exists a d_n such that $x_4.d_n \leq x_1.d_n$, $x_4.d_n \leq x_2.d_n$ and $x_4.d_n \leq x_7.d_n$. Unfortunately, $x_4.d_2 > x_2.d_2$ (i.e. d_2 fails), $x_4.d_3 > x_1.d_3$ (i.e. d_3 fails) and $x_4.d_4 > x_1.d_4$ (i.e. d_4 fails). Since $x_4 \not\prec_b M(x_4, d_1)$, $x_4 \notin C$. When the dimensionality, N, is very high, it will be very difficult for a point to become a core skyline point because it is difficult for a point to dominate-back all N dominant-sets. Hence, k-dominant core skyline is proposed. k denotes the number of dominant-sets that a point has to dominate-back. Formally:

Definition 6. (k-dominant core skyline, C^k) *Let $C^k \subseteq S$. A point x_i is a k-dominant core skyline point ($x_i \in C^k$) iff $x_i \in S$ and $\exists D' \subseteq D$, $|D'| = k$, $\forall d_m \in D'$, $x_i \prec_b M(x_i, d_m)$.*

Based on Definition 6, $C^4 = \{x_1, x_5\}$, $C^3 = \{x_1, x_3, x_5, x_7\}$, $C^2 = \{x_1, x_2, x_3, x_5, x_6, x_7\}$ and $C^1 = \{x_1, x_2, x_3, x_5, x_6, x_7\}$. Note that x_4 is a skyline point but is not a k-dominant core skyline point. Also, $C^k \subseteq C^{k'}$ for $k < k'$. When there are many points in a dataset, it will be very difficult for a point x_i to dominate-back *all* points in $M(x_i, d_m)$. As such, one may consider x_i is important if it can dominate-back a reasonable number of points in $M(x_i, d_m)$. As a result, a relation called p-dominate-back is defined:

Definition 7. (p-dominate-back, $\prec_{pb}$) *Let $M \subset X$ and $0 \leq p \leq 1$. A point x_i is said to p-dominate-back M ($x_i \prec_{pb} M$) iff $\exists M' \subseteq M, |M'| \geq p \times |M|$ such that $M' = \emptyset$ or $\exists d_n, \forall x_j \in M', x_i.d_n \leq x_j.d_n$.*

Yet, we cannot have the concept of p-dominate-back in k-dominant core skyline by simply replacing $\prec_b$ to $\prec_{pb}$ because it is possible that some non-interesting points might be able to p-dominate-back a large number of its dominant-sets while some interesting points cannot (we obmit this proof due to limited space). So we have a new definition:

Definition 8. (k-dominant p-core skyline, C^k_p) *Let $C^k_p \subseteq S$. A point x_i is a k-dominant p-core skyline point ($x_i \in C^k_p$) if and only if $x_i \in S$ and $\exists D' \subseteq D, |D'| = k, \forall d_m \in D'$ both these two conditions hold: (1) $x_i \prec_{pb} M(x_i, d_m)$; (2) $\forall x_j \in M(x_i, d_m), x_i \prec_{pb} M(x_j, d_m)$.*

In Definition 8, Condition (1) is trivial but will lead to the aforementioned pitfall. So Condition (2) is imposed. x_i can be a k-dominant p-core skyline point only if it can p-dominate-back $M(x_j, d_m)$ for all $x_j \in M(x_i, d_m)$. Note that $C^k_0 = C^0_p = C^0_0 = S$. Since core skyline ($C$) and k-dominant core skyline (C^k) are special cases of C^k_p ($C = C^N_1$ and $C^k = C^k_1$), we will focus on studying C^k_p. The problems that we want to solve are:

1. Given k and p, extract C^k_p dynamically and progressively.
2. Given δ, identify the smallest k' such that $|C^{k'}_p| \leq \delta$ and $0 \leq k' \leq N$. If no k' satisfies this condition, then $k' = N$.

Problem 1 is trivial. For Problem 2, from a user's point of view, it is convenience because a user only needs to specify the maximum number of points that she wants to obtain but does not need to understand the distribution of data.

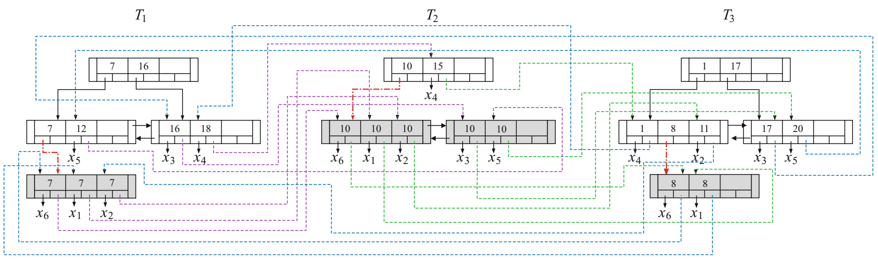


Fig. 1. General structure of the Linked Multiple B'-tree (LMB)

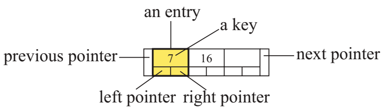


Fig. 2. A node in a B'-tree

3 Proposed Work: LMB

In this section, we describe how to extract C_p^k efficiently by using a novel tree structure called LMB. Note that we do not need to identify skyline before we extract C_p^k. This is important as most skyline points may not belong to C_p^k when p and k are of reasonable values. We can then minimize the computational cost.

Figure 1 shows the structure of LMB by using the dataset in Table 2C . There are three B'-tree, T_1, T_2 and T_3, linked together. T_1, T_2 and T_3 are responsible for indexing the data in d_1, d_2 and d_3, respectively. Each node in a B'-tree contains some entries, a previous pointer and a next pointer (Figure 2). The next pointer points to a succeeding sibling node (if any) and the previous pointer points to a preceding sibling node (if any). Each entry in a node contains a key and two pointers. The left pointer either points to a record or points to a node. The right pointer either points to an entry in another B'-tree that refers to the same point or points to null. In real implementation, each node is a page in disk so a node should have more entries rather than three. The nodes that are shaded are called *overflow nodes*. Overflow nodes are *not* regarded as leave nodes. So the heights of T_1, T_2 and T_3 are respectively two, one and two. In a B'-tree, records are stored at leave nodes and overflow nodes. Each overflow node contains points sharing the same key. For example, T_1 contains one overflow node (key 7) with three entries. This indicates there are three points having the value 7 in d_1.

Points in an overflow node are ordered according to their values in *some other dimensions*. For example, given two points x_i and x_j, where $x_i.d_m = x_j.d_m$, there are two possibilities for their values in the other dimensions: (1) $\forall d_n \in D, x_i.d_n = x_j.d_n$ (i.e. x_i and x_j are identical), and (2) $\exists d_n \in D, x_i.d_n \neq x_j.d_n$. For Case (1), x_i and x_j will be ordered in an overflow node according to their reverse order of insertion. E.g. x_1 and x_6 in Table 2C are identical, so their orders in T_1, T_2 and T_3 are all $x_6 \rightarrow x_1$ (x_6 is on the left of x_1) as x_6 is inserted after x_1. For Case (2), B'-tree will continue to compare

Algorithm 1. `insert`(x_i, T_m)

```
input : A point $x_i$ and a B'-tree $T_m$
1  $\nu \leftarrow x_i.d_m$;
2  if $\exists \nu$ in $T_m$ then
3      if overflow node for the key $\nu$ does not exist then
4          node $\leftarrow$ a new overflow node;
5          identify entry where entry.key = $\nu$ and entry is in leave node;
6          insert $x_j$ into node where $x_j$ = entry.left;
7          entry.left $\leftarrow$ node;   entry.right $\leftarrow$ null;
8      else
9          node $\leftarrow$ first overflow node of $\nu$;
10     insert($x_i, d_m, node$);
11 else
12     insert $x_i$ into $T_m$ just like a traditional B+tree;
```

the values of x_i and x_j in the immediate next dimension, until their values are different. E.g. in d_2, $x_1.d_2 = x_2.d_2 = x_3.d_2 = x_5.d_2 = x_6.d_2 = 10$. To order these points in T_2, we compare their values in the immediate next dimension of d_2, which is d_3. In d_3, $x_1.d_3 = 8$, $x_2.d_3 = 11$, $x_3.d_3 = 17$, $x_5.d_3 = 20$, $x_6.d_3 = 8$. So their ordering in T_2 is $(x_1, x_6) \rightarrow x_2 \rightarrow x_3 \rightarrow x_5$. To order x_1 and x_6 in T_2, we compare their values in the immediately next dimension, which is d_1. Since x_1 and x_6 are identical, we order them according to the reverse order of their insertion, which is $x_6 \rightarrow x_1$. Eventually, the ordering is $x_6 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_5$.

3.1 Implementation

Algorithm 1 outlines the insertion process. Suppose we now insert a new point, x_i, into T_m. First, we check whether there is any point has the same value as x_i in d_m (line 2). If not, the insertion process will be the same as a B+tree [15] (line 13). Otherwise, we check whether an overflow node exists (line 3). If not, we will create a new overflow node (line 4), identify the entry at leave node where its key is $x_i.d_m$ (line 5), insert x_j ($x_j.d_m = x_i.d_m$) into the overflow node (line 6) and attach the overflow node to the correct position in T_m (line 7). x_i will be inserted into the overflow node by calling the function `insert` (line 11).

Algorithm 2 outlines the process of inserting x_i into an overflow node. We iterate each point x_j in an overflow node to see which position should we insert x_i. Let $d_{m'}$ be the immediate next dimension of d_m (line 4). If $x_i.d_{m'} < x_j.d_{m'}$ (line 5), then x_i will be inserted before x_j (line 6). We can do this because all existing points in an overflow node must already be ordered properly based on $d_{m'}$ when they are inserted by calling this function previously. If $x_j.d_{m'} = x_i.d_{m'}$ (line 6), then we will compare the values of x_i and x_j in their immediate next dimension continuously until they are different (line $7-11$). If $\forall d_{m'}, x_i.d_{m'} = x_j.d_{m'}$ (i.e. x_i and x_j are identical), then x_i will be inserted before x_j (line 12) because we have to order the identical points in their reverse order of insertion. Line $14-18$ is used to iterate all points in an overflow node. Finally, if

Algorithm 2. `insert`($x_i, d_m, node$)

```
input : A point, $x_i$, a dimension, $d_m$, an overflow node, $node$
1  $i \leftarrow 1$;
2  repeat
3      $x_j \leftarrow$ the point at entry $i$ in $node$;
4      if $m \neq N$ then $m' \leftarrow m+1$; else $m' \leftarrow 1$;
5      if $x_i.d_{m'} < x_j.d_{m'}$ then insert $x_i$ before $x_j$; return;
6      else if $x_j.d_{m'} = x_i.d_{m'}$ then
7          for $i \leftarrow 1$ to $N-1$ do
8              if $m' \neq N$ then $m' \leftarrow m'+1$; else $m' \leftarrow 1$;
9              if $x_i.d_{m'} < x_j.d_{m'}$ then insert $x_i$ before $x_j$; return;
10             else if $x_i.d_{m'} > x_j.d_{m'}$ then insert $x_i$ after $x_j$; return;
11         insert $x_i$ before $x_j$; return;
12     $i \leftarrow i+1$;
13     if entry $i$ is empty then insert $x_i$ in entry $i$; return;
14     else if $i >$ number of entries in node then
15         $i \leftarrow 1$;    $node \leftarrow node.next$;
16 until node is null ;
17 $node \leftarrow$ a new overflow node;
18 insert $x_i$ in the first entry of $node$;
19 attach $node$ to the last overflow node;
```

$\forall x_j, x_i.d_{m'} > x_j.d_{m'}$, then x_i be inserted into the last entry of an overflow node (line 15). If the entries are full, then x_i will be inserted into a new overflow node(line 20–22). For each entry, its right pointer will point to an entry in the immediate next B'-tree that points to the same record. This process is trivial and can be performed in any stage during the insertion process. For deletion, its steps are similar to a B+tree, except that when an overflow node contains only one entry, then this entry will be propagated back to its parent node and the overflow node will be deleted. As this process is trivial, we do not present a detailed algorithm due to the limited space in this paper.

3.2 Extraction

Once LMB has indexed all points, we can extract C_p^k *dynamically* and *progressively* according to a reference point, r (e.g. r is the origin). Given a point x_i and a dimension d_m, suppose all points in $M(x_i, d_m)$ are ordered according to d_n ($d_n \neq d_m$) in an descending order. Let x^* be the point at position $\lceil p \times |M(x_i, d_m)| \rceil$ in $M(x_i, d_m)$. If $x_i.d_n < x^*.d_n$, then x_i obviously must be able to p-dominate-back $M(x_i, d_m)$ by using d_n. Hence, Condition (1) of Definition 8 can be verified quickly if x^* can be identified efficiently. Now, assume there is a point $x_j \in M(x_i, d_m)$. In order for x_i p-dominate-back $M(x_j, d_m)$ using d_n, $x_i.d_n$ must be less than or equal to $x_j^*.d_n$ where x_j^* is the point at position $\lceil p \times |M(x_j, d_n)| \rceil$ (remember $M(x_j, d_m)$ is sorted based on d_n). Hence, to verify Condition (2) of Definition 8 quickly, what we need to do is to check whether $\exists d_n, x_i.d_n \leq \min_j x_j^*.d_n$ where x_j^* is the point at position $\lceil p \times |M(x_j, d_n)| \rceil$. Once we can verify Condition (1) and Condition (2) of Definition 8 quickly, we can extract C_p^k efficiently. Hence, the major issue

```
Algorithm 3. extract($p,k,r$)
input : two user defined threshold, $p$ and $k$, and a reference point, $r$
output: $k$-dominant $p$-core skyline, $C_p^k$
1  for $m \leftarrow 1$ to $N$ do
2      $x^m \leftarrow$ the point closest to $r$ in $d_m$; // e.g. $r$ is the origin
3      $B_{mn} \leftarrow \emptyset, n = 1,2,\ldots,N$; // $B_{mn}$ is a BTree
4  $C_p^k \leftarrow \emptyset$;   $H \leftarrow \emptyset$;
5  for $i \leftarrow 1$ to $|X|$ do
6      for $m \leftarrow 1$ to $N$ do
7          $added \leftarrow false$;
8          for $n \leftarrow 1$ to $N, n \neq m$ do
9              if $B_{mn} = \emptyset$ or $|x^m.d_n - r_m.d_n| < B_{mn}.last$ then
10                 insert $|x^m.d_n - r_m.d_n|$ into $B_{mn}$;
11                 if $addded = false$ and $\nexists x \in H, x \prec x^m$ then
12                     $C_p^k \leftarrow C_p^k \cup$ update($H,k,x^m$);
13                     $added = true$;
14                 if $\lceil p \times i \rceil > |B_{mn}|$ then
15                     remove last element from $B_{mn}$;
16         $x^m \leftarrow$ the point closest to $x^m$ (besides itself) in $d_m$;
17     $i \leftarrow i + 1$;
18 return $C_p^k$;
```

we need to deal with is how to identify x^* and x_j^* with respect to d_m quickly. We extract C_p^k based on this idea.

Algorithm 3 outlines the steps for extracting C_p^k. Line $1-5$ initialize some parameters. In line 2, if there are more than one point closest to the reference point r in d_m, then x^m will be initialized to the one which is the first occurrence in an overflow node. With LMB, we can identify x^m very quickly. This process is similar to a B+tree. B_{mn} (line 3) is a BTree. It helps us to determine whether a point can p-dominant-back a dominant-set. If x_i can p-dominate-back $M(x_i,d_m)$ by using d_n, then x_i will be stored in B_{mn}. We may not always need to store x_i permanently in B_{mn}. We want to keep B_{mn} as small as possible so as to reduce memory consumption and computational time. We explain this step by step below. Let $\nu = |x^m.d_n - r_m.d_n|$. We can identify $x^m.d_n$ quickly by using LMB without accessing the record directly. Whenever B_{mn} is empty or ν is less than the last value in B_{mn} (i.e. the largest value in B_{mn}), then ν will be inserted into B_{mn} (line 10 and 11). Let i be the i^{th} point closest to the reference point r. Whenever $|B_{mn}| < \lceil p \times i \rceil$, then the last value in B_{mn} will be removed (line $16-18$). By doing so, we can keep the size of B_{mn} always not exceed $\lceil p \times |M(x^m,d_m)| \rceil$. We can do so because we extract x^m one by one according to the ascending distance to r (line 21). Furthermore, note that the last value in B_{mn} is in fact $\min\{x^*.d_n, x_j^*.d_n\}$ with respect to d_m. If $x^m < \min\{x^*.d_n, x_j^*.d_n\}$, it implies x^m satisfies Condition 1 and Condition 2 of Definition 8. So x^m will be added into H (line $12-15$). In line 12, the condition $\nexists x \in H, x \prec x^m$ guarantees x must be a skyline. The rest of Algorithm 3 should be self-explained. Finally, H (line 6) is a hash

Algorithm 4. `update`(H,k,x,v)

```
input  : A hashtable, $H$, a parameter, $k$, a point, $x$, a value, $v \in \{0,1\}$
output: A $k$-dominant $p$-core skyline point, $x$, or $\emptyset$
1 if H does not contain element with key x then
2     put $(x,V)$ into $H$; // $x$ is the key and $V$ (a set) is the element
3 add $v$ into $V$; // $V$ is the element with key $x$
4 $k' \leftarrow$ number of $v$ in $V$ equals 1;
5 if $k' \geq k$ and x is not yet returned then
6     return $x$;
7 return $\emptyset$;
```

table. It stores how many dominant-sets that a point can p-dominant-back. The function `update` (line 24 and 26) is used to update the information stored in H so as to extract C_p^k. It is outlined in Algorithm 4. In Algorithm 4, V (e.g. line 2) is a set that stores the "p-dominate-back result" of x. If x can p-dominate-back a specified dominant-set, then 1 will be added into V; otherwise, 0 will be added (line 4). If the number of 1 in V is greater than or equals to k (k is a user-defined parameter to extract C_p^k), then x will be added into C_p^k (line $5-8$). Note that x is added into C_p^k progressively so we can return results to users immediately and progressively.

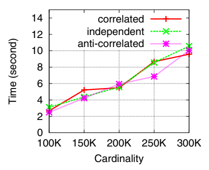
3.3 Further Analysis

Algorithm 3 tries to solve Problem 1 which is raised in Section 2. For Problem 2, we can solve it by: (1) Remove line 4 to line 8 in Algorithm 4; (2) The appropriate k' could be obtained easily by checking the number of $v = 1$ in each V in H after Algorithm 3 is completed. For example, when $N = 3$ (i.e. $|V| = 3$), suppose after completing Algorithm 3, there are five V having three $v = 1$, seven V having two $v = 1$ and ten V having one $v = 1$. Then, $|C_p^3| = 5$, $|C_p^2| = 12$, $|C_p^1| = 22$. If $\delta = 20$ (maximum number of core skyline points returned is 20), then $k' = 2$.

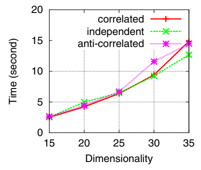
Furthermore, we can extend our algorithm to answer some complex queries easily. We give some examples here to illustrate how this can be done. If we want to return C_p^k for a particular range of k, then we simply change line 6 of Algorithm 4 to the appropriate range. For example, if we want to return C_p^k when $k = 2$ but exclude those points in C_p^k when $k = 4$, then we change line 6 to: "**If** $2 \leq k' < 4$ *and x is not yet returned* **then**". To deal with constraint skyline queries [16], we only need to pay attention to line 2 and line 33 of Algorithm 3. If the x_m returned is not within the constrained region, then we can immediately ignore that dimension d_m and do no need to conduct any further computation.

4 Experiment

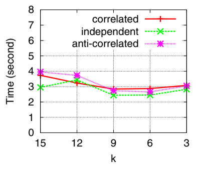
All experiments are conducted using an Intel XEON 2.5GHz CPU in Microsoft Windows Server 2003 R2 Enterprise x64 Edition. All programs are written in Java. We use



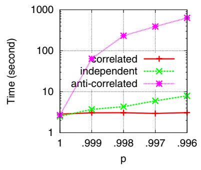
(a) CPU vs. Cardinality



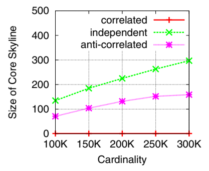
(a) CPU vs. Dim.



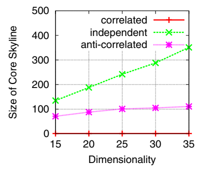
(a) CPU vs. k



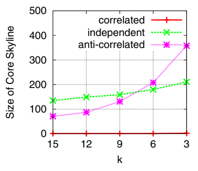
(a) CPU vs. p



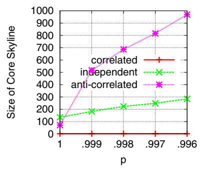
(b) $|C_p^k|$ vs. Cardinality



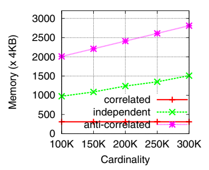
(b) $|C_p^k|$ vs. Dim.



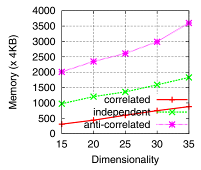
(b) $|C_p^k|$ vs. k



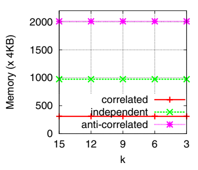
(b) $|C_p^k|$ vs. p



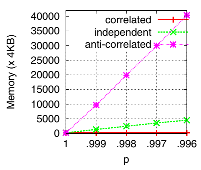
(c) Memory vs. Cardinality



(c) Memory vs. Dim.



(c) Memory vs. k



(c) Memory vs. p

Fig. 3. Cardinality **Fig. 4.** Dim. **Fig. 5.** Effect of k **Fig. 6.** Effect of p

a page size of 4KB for each node of LMB. Following [5,16], we generate several independent, correlated and anti-correlated datasets. In order to evaluate the quality of C_p^k, we use two real life datasets.

4.1 Effect of Cardinality

We set $|X|$ (number of points) $= \{100000, 150000, 200000, 250000, 300000\}$, $N = 15$, $k = 15$ and $p = 1$. Figure 3 (a) shows the CPU time versus cardinality in different datasets. In the figure, there are three lines. They denote the results against independent dataset, correlated dataset and anti-correlated dataset. With LMB, we only need to spend less than 3 seconds to identify C_p^k from a dataset with 15 dimensions and 100,000 points and spend around 11 seconds to identify C_p^k from a dataset with 15 dimensions and 300,000 points. In general, the CPU time increases linearly when the dimensionality increases linearly. For a reference, BBS [8] (the most efficient algorithm to identify skyline) takes more than 1,000 seconds to identify skyline from an independent dataset with 15 dimensions and 100,000 points and takes more than 2,200 seconds to identify skyline from an anti-correlated dataset with the same setting. Figure 3 (b) shows the size of C_p^k versus cardinality. For the independent dataset, $|C_p^k|$ increases linearly. For the correlated dataset, $|C_p^k|$ almost constant (less than 5). For the anti-correlat-ed dataset, $|C_p^k|$ increases slowly when $|X| > 250K$. For the same $|X|$, the size of C_p^k in an independent dataset is always larger than the size of C_p^k in an anti-correlated dataset. Figure 3 (c) shows the memory consumption versus cardinality. The trend of is highly related to the size of C_p^k because we always need to keep a fix amount of information (B_{mn} and

H in Algorithm 3) in the main memory. For each B'-Tree, B_{mn} is more or less constant ($|B_{mn}|$ is usually around $p \times |M(x_i, d_m)|, \forall n$) but $|H|$ is highly related to the number of skyline points in the dataset.

4.2 Effect of Dimensionality

We set N (dimensionality) $= \{15, 20, 25, 30, 35\}$, $|X| = 100{,}000, p = 1$ and $k = 1$. Figure 4 (a), 4 (b) and 4 (c) respectively show the CPU time, the size of C_p^k and the max. memory consumption in different datasets. We can identify k-dominant p-core skyline points within 15 seconds for all datasets even when $k = 35$. The computational time roughly increases linearly. When the dimensionality increases, $|C_p^k|$ does not vary much in the anti-correlated and the correlated datasets. Note that more than 95% of points are skyline points when $|X| = 100{,}000$ and $N = 35$ in the anti-correlated dataset. For the independent datasets, $|C_p^k|$ increases linearly. Nevertheless, more than 90% of points are skyline points when $N = 35$, but the number of core skyline points is just less than 400. We can reduce the number of skyline points dramatically. For the memory consumption, when we compare Figure 3 (c) and Figure 4 (c), Figure 3 (c) is more flat because when the dimensionality increases, we need to store more p-dominate-back information (i.e. B_{mn} in Algorithm 3) for each B'-Tree.

4.3 Effect of k

We set k (number of dominant-sets a point has to p-dominate-back) $= \{15, 12, 9, 6, 3\}$, $|X| = 100{,}000$ $N = 15$ and $p = 1$. Figure 5 (a), 5 (b) and 5 (c) respectively show the CPU time, the size of C_p^k and the maximum memory consumption against k in different datasets. For the CPU time, all lines are roughly constant (or having a slightly decreasing trend) regardless of the choice of k. Technically, when k is small, we have less comparisons before we can decide whether a point should be returned. In practice, it seems that the computational time differences between a small k and a large k is negligible. For the size of C_p^k, the rate of increase of the anti-correlated dataset is much faster than the independent dataset. For the memory consumption, it is constant.

4.4 Effect of p

We set $p = \{1, 0.999, 0.998, 0.997, 0.996\}$, $|X| = 100{,}000$, $N = 15$ and $k = 1$. Figure 6 shows the results. It is quite obvious that p has a significant impact on C_p^k (especially the anti-correlated dataset). In Figure 6 (a), the time requires to identify C_p^k from an anti-correlated dataset increases exponentially. Fortunately, having a small p in our problem is unreasonable as our objective is to extract a small set of interesting skyline points. When the value of p decreases, the size of C_p^k increases dramatically. This is shown in Figure 6 (b). When $p = 1$, $|C_p^k|$ is less than 100 in the anti-correlated dataset; when $p = 0.996$, $|C_p^k|$ is around 900. For the memory consumption (Figure 6 (c)), the memory needed for correlated dataset and independent dataset do not vary much. However, the anti-correlated dataset does require a large amount of memory when p decreases.

Table 3. Players extracted by C_p^k

k	p	Players
11	1	Alvin Robertson 1985, Dennis Rodman 1991, Hakeem Olajuwon 1989, John Stockton 1990, Latrell Sprewell 1993, Michael Jordan 1986, Moses Malone 1979, Ray Allen 2005 (**8 records, 8 players**)
5	1	Allen Iverson 2002, Alvin Robertson 1985, Antoine Walker (2000, 2001), Charles Barkley 1988, Dennis Rodman 1991, Gary Payton 1999, George Mccloud 1995, Gilbert Arenas 2005, Hakeem Olajuwon (1988, 1989, 1992), Isiah Thomas 1984, Jason Richardson 2007, John Stockton (1988, 1990, 1991), Karl Malone 1989, Kevin Garnett (2002, 2003), Kevin Willis 1991, Kobe Bryant 2005, Latrell Sprewell 1993, Magic Johnson 1986, Mark Eaton 1984, Michael Jordan (1986, 1987, 1988, 1989), Michealray Richardson 1979, Moses Malone (1979, 1981, 1982), Predrag Stojakovic 2003, Ray Allen 2005, Shaquille O'neal 1999 (**36 records, 25 players**)
11	0.94	Adrian Dantley 1980, Allen Iverson (2002, 2007), Alvin Robertson (1985, 1986), Anthony Mason 1995, Antoine Walker 2001, Dennis Rodman 1991, Dennis Scott 1995, Gary Payton 1999, George Mccloud 1995, Gilbert Arenas 2005, Hakeem Olajuwon (1988, 1989), Jason Richardson 2007, John Stockton (1987, 1988, 1990, 1991), Kevin Garnett (2002, 2003, 2004), Kobe Bryant 2005, Latrell Sprewell 1993, Magic Johnson (1981, 1982), Michael Finley 1999, Michael Jordan (1986, 1987, 1988), Michealray Richardson 1979, Mitch Richmond 1995, Mookie Blaylock (1993, 1995), Moses Malone (1979, 1981,1982), Predrag Stojakovic 2003, Ray Allen 2005, Reggie Miller 1996, Shaquille O'neal 1999, Steve Nash (2006, 2007), Tim Hardaway 1991 (**44 records, 29 players**)

4.5 Quality of Result

We evaluate the quality of C_p^k using two real life datasets. One is called NBA dataset and the other one is called MovieLens. [3]. The NBA dataset has 11,301 records. It includes all NBA players from 1979 to 2007. Each record denotes the average performance of a player in a year. The MovieLens dataset has 3,952 movies. The schema for NBA dataset is: minutes played, points obtained, offensive rebound obtained, defensive rebound obtained, total rebound obtained, assistant made, steal made, block made, three points made, free throw made and field goal made. The schema for MovieLens dataset is: Female rating, Male rating, Female under, Male under, Female 19 - 25, Male 19 - 25, Female 26 - 35, Male 26 - 35, Female 36 - 45, Male 36 - 45, Female 46 - 50, Male 46 - 50, Female 51 - 55, Male 51 - 55, Female 56 or above, Male 56 or above and Overall rating. Due to the limited space, we only report some of the most interesting findings. Readers can download all results online. We implement the *k*-dominant skyline algorithm [5] for comparison. This algorithm is proved to be very effective in extracting interesting points from a large set of skyline points. *We do not compare this algorithm in the previous experiments because we are neither an extension of it nor targeting to obtain the same result.*

Table 3 shows the players extracted by C_p^k. All players are sorted by their first names. [5] We report C_1^{11}, C_1^5 and $C_{0.95}^{11}$. We choose them because: (1) C_1^{11} is the most tight constraint; (2) C_1^5 implies a player should *p*-dominate-back around half of dimensions in the

[3] `www.databasebasketball.com` and `www.grouplens.org`

Table 4. Players extracted by k-dominant skyline [5]

K	Players
9	Alton Lister 1986, Charles Barkley (1985, 1986, 1987, 1988), Charles Oakley 1986, David Robinson (1990, 1992, 1993, 1994, 1995), Dennis Rodman (1991, 1993), Dikembe Mutombo (1995, 1999), Dirk Nowitzki 2002, Dwight Howard 2007, Elvin Hayes 1979, Gary Payton 1999, Hakeem Olajuwon (1988, 1989, 1992, 1993, 1994), James Donaldson 1986, Julius Erving 1980, Karl Malone (1989, 1990, 1991, 1992, 1993), Kevin Garnett (1999, 2000, 2001, 2002, 2003, 2004, 2006), Larry Bird (1983,1986), Mark West 1989, Michael Jordan (1986, 1987, 1988, 1989), Moses Malone (1979, 1980, 1981, 1982), Patrick Ewing (1989, 1990), Samuel Dalembert 2006, Shaquille O'neal (1992, 1993, 1999), Shawn Marion 2002, Tim Duncan (2001, 2002) (**58 records, 24 players**)

dataset. This make sense in the basketball situation; (3) The number of skyline points returned by $C^5_{0.94}$ in this dataset is similar to the number of skyline points returned by C^5_1. We can compare the results returned by $C^5_{0.94}$ and C^5_1.

One frequently asked question is that "why X is not included? I think he plays equally well with Y!" Yet, this type of question is subjective. Furthermore, one cannot deny the fact that the players in Table 3 are all famous. In addition, there are more than 1,000 skyline points in the dataset. Some skyline points belong to some not-so-famous players. If we randomly extract some skyline points from the dataset, it is very likely to extract the not-so-famous players. When we compare $C^5_{0.94}$ and C^5_1, although most names are similar, some names appear in C^5_1 but not in $C^5_{0.95}$, such as Charles Barkley (another great NBA players). If we change $C^5_{0.95}$ to $C^5_{0.90}$, then Charles Barkley re-appears again. This shows that both k and p are useful in extraction. Finally, when $k = 1$ and $p = 1$, there are 132 records (67 players) extracted. When we apply [5], it extracts one record, Moses Malone, when $6 \leq K \leq 7$ (according to [5], it is meaningless to set $K \leq N/2$, so we do not test these cases), extracts 17 records when $K = 8$, extracts 58 records when $K = 9$, extracts 279 records when $K = 10$. The number of records increases exponentially when K increases linearly. We cannot have much control over the number of records to be extracted. However, in C^k_p we can set different value of p so that we can have more control about the number of records to be extracted. E.g. when $k = 11$ and $p = 0.99$, we can extract 19 records in C^k_p. In addition, Table 4 shows the records extracted by K-dominant skyline when $K = 9$. The records extracted by K-dominant skyline approach are not similar to ours. It extracts fewer players but more records. For example, Allen Iverson and Kobe Bryant, two very great NBA players, do not appear in K-dominant skyline when $K \leq 10$. This is because both players are shooters and do not perform very outstanding in rebound and block. It is very difficult for them to K-dominate other players for whatever K. This is why they cannot be extracted in K-dominant skyline. However, for our proposed work, we consider vertical relationships rather than only horizontal relationships.

5 Summary and Conclusion

We propose a novel concept called k-dominant p-core skyline for extracting interesting points from a skyline. k denotes the number of dimensions that a point has to dominate-back and *p* denotes the fraction of points in a given dimension that a point has to dominate-back. This work is different from the existing work as we further consider the vertical relationships among points. To the best of our knowledge, we are the first mover in this area.

References

1. 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 (2009)
2. Agrawal, R., Wimmers, E.L.: A framework for expressing and combining preferences. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD 2000 (2000)
3. KieBling, W.: Foundations of preferences in database systems. In: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB 2002 (2002)
4. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005 (2002)
5. Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD 2006 (2006)
6. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th Very Large Database Conference, VLDB 2002 (2002)
7. Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K., Zhang, Z.: On high dimensional skylines. In: Proceedings of the 10th International Conference on Extending Database Technology, EDBT 2006 (2006)
8. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transactions on Database Systems (TODS) 30(1), 41–82 (2005)
9. Zhang, Z., Guo, X., Lu, H., Tung, A.K.H., Wang, N.: Discovering strong skyline points in high dimensional spaces. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, CIKM 2003 (2005)
10. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: The k most representative skyline operator. In: Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007 (2007)
11. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of Data, SIGMOD 2006 (2006)
12. Das, G., Gunopulos, D., Koudas, N., Sarkas, N.: Ad-hoc top-k query answering for data streams. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB 2007 (2007)
13. Bohm, C., Ooi, B.C., Plant, C., Yan, Y.: Efficiently processing continuous k-nn queries on data streams. In: Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007 (2007)
14. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering, ICDE 2001 (2001)
15. Ramakrishnan, R., Gehrke, J.: DatabaseManagement Systems, 3rd edn. McGraw-Hill, New York (2003)
16. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD 2003 (2003)