World Wide Web (2012) 15:33–60
DOI 10.1007/s11280-011-0122-8

Finding superior skyline points for multidimensional recommendation applications

Jing Yang · Gabriel Pui Cheong Fung · Wei Lu · Xiaofang Zhou · Hong Chen · Xiaoyong Du

Received: 6 November 2010 / Revised: 30 January 2011 /
Accepted: 4 February 2011 / Published online: 4 March 2011

Abstract In a typical Web recommendation system, objects are often described by many attributes. It also needs to serve many users with a diversified range of preferences. In other words, it must be capable to efficiently support high dimensional preference queries that allow the user to explore the data space effectively without imposing specific preference weightings for each dimension. The skyline query, which can produce a set of objects guaranteed to contain all top ranked objects for any linear attribute preference combination, has been proposed to support this type of recommendation applications. However, it suffers from the problem known as 'dimensionality curse' as the size of skyline query result set can grow exponentially

J. Yang (✉) · W. Lu · X. Zhou · H. Chen · X. Du
School of Information, Renmin University of China, Beijing, China
e-mail: jingyang@ruc.edu.cn, sohpiejingyang@gmail.com

W. Lu
e-mail: uqwlu@ruc.edu.cn

H. Chen
e-mail: chong@ruc.edu.cn

X. Du
e-mail: duyong@ruc.edu.cn

J. Yang · W. Lu · X. Zhou · H. Chen · X. Du
Key Labs of Data Engineering and Knowledge Engineering,
Ministry of Education, Beijing, China

G. P. C. Fung
School of Computing Informatics, Arizona State University,
Phoenix, AZ, USA
e-mail: gabriel.fung@asu.edu

X. Zhou
School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, QLD, Australia
e-mail: zxf@uq.edu.au

with the number of dimensions. Therefore, when the dimensionality is high, a large percentage of objects can become skyline points. This problem makes such a recommendation system less usable for users. In this paper, we propose a stronger type of skyline query, called *core skyline query*, that adopts a new quality measure called *vertical dominance* to return only an *interesting* subset of the traditional skyline points. An efficient query processing method is proposed to find core skyline points using a novel indexing structure called *Linked Multiple B'-trees* (LMB). Our approach can find such superior skyline points progressively without the need of computing the entire set of skyline points first.

Keywords preference query · recommendation systems · high dimensional data · vertical dominance · core skyline points · linked multiple B'-tree

1 Introduction

The World Wide Web has brought an unprecedented amount of information and choices to our modern society. To make better use of such information, it demands a Web Information System, being an online shop, a service marketplace or a Web-based social networking platform, to make targeted and relevant recommendations using as much as possible information for a wide range of users. An effective and efficient recommendation system is critical to improving sales, growth and customer satisfaction. This goal, however, is very challenging to achieve. Among many issues, one technical problem yet to be satisfactorily addressed is preference query processing for multidimensional data when there is no single preference function to combine all dimensions of an object into one score that can be used to rank objects. The cruces of the this problem include: (1) the lack of a single or a small set of pre-defined preference functions to combine values form different attributes, as a large user community naturally leads to different and often changing preferences for the importance of attributes; and (2) the intrinsic difficulty of ranking objects in a high dimensional space (in the case of Web applications, high dimensionality comes from the fact that data objects are often described by a rich set of attributes).

Consider the following simplified example for a web site that compares new laptops. There are six models and each model is described by eight attributes here. Although it is possible to prioritize the laptops using a single attribute, such as price, a user may be interested in comparing products based on multiple attributes. How to rank the laptops using multiple attributes is a very difficult problem. As one can see from this example, if all eight attributes are considered there is no single model which is an overall winner. So the question is: which laptops should be recommended to the user?

Generally speaking, an object to be ranked by a recommendation system can be considered as a point in a multidimensional space, where a dimension is an attribute of the object. Let $X = \{x_i | i = 1..n\}$ be a set of points in an N dimensional space $D = D_1 \times \cdots \times D_N$. That is, x_i is represented by N attributes, $x_i(d_1, \cdots, d_N)$, where $d_i \in D_i$. The difficulty to rank these objects with all attributes comes from the lack of a total order in a multidimensional space. In order to make a recommendation of a reasonably-sized subset of objects to a user, a common way is to ask the user to specify a relative preference weighting for each attribute. Without loss of generality, we can assume that the value in every attribute is the larger the better (for attributes *weight* or *price* in the above example, their value v can be replaced by $m - v$ for a common celling value m of that attribute when computing the ranks). Let the user-specified preference weighting on attributes $d_1, \cdots d_N$ be $w_1, \cdots w_N$ respectively ($\sum_{i=1}^{N} w_i = 1$), a simple scoring function $f = \sum_{i=1}^{N} d_i \times w_i$ maps these multidimensional objects into a linear space where all objects can be ranked. Other problems such as values in different attributes maybe of different scale thus cannot be directly combined can be addressed easily using some scaling functions, and of course other scoring functions such as L_p-norm distance function, their weighted versions, or other more sophisticated functions can also be used. This is the basis for what known as *top-k* queries in the literature [7].

However, it is not practical for a recommendation system to give such a set of pre-defined preference weightings, as, after all, this is a user choice and different users should be given the freedom to give higher weights to some attributes and lower or even zero weights to others. At the same time, the user may not have the knowledge, time or even the desire to give weights to all attributes for an online application (note that the number of attributes can be quite large for some applications). This leads to the second type of queries known as *skyline* queries, where all points in X are returned as long as they are not *dominated* by any other point in X. That is, $x_i \in X$ is a skyline point if and only if there does not exist $x_j \in X$ such that x_j performs better than x_i in all N dimensions [6]. All skyline points cannot be compared with each other (without an appropriate preference function) [1, 21], because each skyline point must have at least one dimension performs better than any other skyline points when they are pair-wisely compared. A very useful property of the skyline query is that it will return a superset of preferred objects in X for any valid linear preference combination. Skyline query is a powerful tool to make recommendation to a user without forcing them to give specific weights to each and every attribute—this is often an impossible ask in practice. In the above example, it is easy to see that laptop models 1, 2 and 5 are skyline points because they have the longest battery time, lowest price and fastest CPU respectively in this data set. Models 3 (and 6) are also skyline point because there are no models in the dataset of the same or better warranty but cheaper (and the same or larger display size but lighter). Model 4, however, is not a skyline point as it is ‘dominated’ by model 5.

While the skyline query may sound like an ideal solution for making recommendations without a preference function, it does not really work for many Web applications as the number of skyline points returned by a typical Web application can simply be too large for a user to comprehend. In the above example, five of the six models will be returned by the skyline query. This problem, unfortunately, is not uncommon for large data sets when there are many attributes. It is known that when the dimensionality is very high, almost all points are skyline points [45]. Therefore, it is necessary to further differentiate these skyline points.

Let us look at the above example again. Among the five models that cannot be fully dominated by any other model, it does not make much sense for model 3 to be returned to the user, as there are obvious better options in the dataset. The only reason for it be to included in the skyline is that it has 13 months warranty while all other models except a slightly more expensive model 6 have 12 months or less warranty. While it is possible to give many kinds of domain knowledge to give some significance indication to attributes or attribute values (e.g., length of warranty), the question we will answer in this paper is, without assuming any knowledge about how much importance a user puts on attributes or attribute values, how can we remove those points that meet the skyline definition but are unlikely to be of user's interest?

A number of approaches for extracting 'interesting' skyline points in high dimensional space have been proposed in the literature (e.g., [10, 22]). A reason that there are too many skyline points is that the chance for a point to dominate another point in a high dimensional space is low, as a dominating point needs to be "not worse in *all* dimensions" while performs better in at least one dimension. One way is to relax the condition of "not worse in *all* dimensions" to "not worse in k dimensions" without changing the requirement for performing better in at least one dimension. This notion of k-dominance is introduced in [10]. Obviously, k-dominant skyline returns a subset of the skyline, and its size can be much smaller especially when k is small (when $k = N$, this becomes the traditional skyline). In Table 1, as long as *warranty* is not included in the consideration, model 3, a not-so-interesting object, is k-dominated by model 1 (and several other objects). The notion of k-dominant skyline is proven to be effective in extracting interesting skyline points. However, the k-dominance relationship is not transitive, and sometimes less intuitive to the user. Furthermore, they do not consider vertical relationship of each dimension (as we will discuss it later). A concept called skyline frequency is proposed to extract 'interesting' skyline points considering subspaces [11]. 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. It generalizes the idea of k-dominance and makes it more systematic. On the other side, it does have two major issues: (1) Some points that are intuitively superior may sometimes have the same ranking (or lower ranking) as the inferior points, this makes the intuition behind less obvious to the user. (2) There are $2^N - 1$ subspaces in an N dimensional space, which is too expensive to compute in practice. For the second issue, an approximation algorithm to solve it is proposed in [11], but it cannot answer dynamic queries which are quite common for recommendation systems [34]. Also considering skyline in subspaces, Zhang et al. [47] proposes an idea called δ-subspace to identify a set of interesting skyline points. A 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-subspaces are called strong skyline points. This algorithm depends on δ heavily. If δ is 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 discourages it from being used in practice.

Table 1 New laptops.

Model ID	Display (inch)	Weight (kg)	Memory (GB)	Harddisk (GB)	CPU (GHz)	Battery (hour)	Warranty (month)	Price ($)
1	10	1.19	2	160	1.60	10.5	12	729
2	7	0.86	8	0	0.35	3.4	6	268
3	15	3.10	2	120	1.4	2.0	13	1,899
4	15	2.80	4	500	2.26	4.0	12	1,082
5	15	2.64	4	500	2.3	4.5	12	699
6	13.3	1.36	2	120	1.86	4.2	13	1,999

There are a number of other approaches that attempt to rank skyline points based on some kind of importance among the skyline points. Lin et al. [27] 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 dimensions. 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 [5, 15, 31, 34], 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.

To summarize, skyline queries can be an effective way to provide insights to the dataset which cannot be ranked by any total order, but suffer from the problem of returning too many results especially when the number of attributes is large. The existing work that attempt to reduce the number of results to be returned (i.e., only to return those most 'interesting' skyline points), are limited by the fact that skyline operator compares tuples *horizontally*. That is, two tuples are compared if one dominates another in isolation, by pair-wise comparison of their attributes (or a subset of their attributes). We believe that such a comparison should also be done *vertically*, as those 'interesting' points are 'interesting' in the context of other tuples in the dataset. Logically, interesting laptop models should not be those which simply perform better than the weakest attribute of any other models (e.g., model 3 wins mainly in warranty length). A potential candidate should have some attributes that it has a value better than a reasonable number of models in those attributes. 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 laptop models perform better than a model x in an attribute. If x wants to claim it is one of the potential candidates to be better than others, then it is reasonable to check if x has another attribute that it can perform better than *all* those n models. In Table 1, model 3 cannot fulfil this criterion (for attributes such as battery life and weight).

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. The main idea in this work is a novel concept called *dominate-back*, that vertically checks if a set of points perform better than a skyline point in one dimension can be dominated back by that point in another dimension; if yes, then such a skyline point in called a core skyline point. Details of this formulation will be described in Section 2. After presenting some theoretical results about skyline query result set size and dimensionality in Section 3 and a complexity analysis for a straightforward solution to the core skyline problem in Section 4, we propose an indexing algorithm in Section 5 called Linked Multiple B'-tree (LMB)[1] to help us identify core skyline

[1] "B'-tree" is pronounced as "B-pie-tree"

results dynamically and progressively. An empirical performance study on our approach is reported in Section 6. An overview about skyline query processing is give in Section 7, before we conclude this paper in Section 8.

2 Problem definition

Let $D = \{d_1, d_2, \ldots, d_N\}$ be an N-dimensional space and $X = \{x_1, x_2, \ldots\}$ 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 [6]:

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 B as a running example. Table 2A contains a dataset with seven points and four dimensions. Table 2B

Table 2 Some sample datasets.

ID	d_1	d_2	d_3	d_4
A. Dataset 1 (Order by ID)				
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
Rank	d_1	d_2	d_3	d_4
B. Dataset 1 (Order by ranking)				
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
Insertion	ID	Dimensions		
order		d_1	d_2	d_3
C. Dataset 2				
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

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 B 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:

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 omit this proof due to limited space). So we have a new definition:

Definition 8 (k-dominant p-core skyline, C_p^k) Let $C_p^k \subseteq S$. A point x_i is a k-dominant p-core skyline point ($x_i \in C_p^k$) 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 $C_0^k = C_p^0 = C_0^0 = S$. Since core skyline ($C$) and k-dominant core skyline (C^k) are special cases of C_p^k ($C = C_1^N$ and $C^k = C_1^k$), we will focus on studying C_p^k. The problems that we want to solve are:

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

Problem 1 is easy to understand. For Problem 2, from a user's point of view, it is convenient 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.

3 Skyline size and dimensionality

It is well-known that when the dimensionality increases slightly, the size of skyline increases dramatically. In this section, we try to quantify the relationship between the probability of a point belonging to a skyline and the dimensionality of the point.

Theorem 1 *Given a point x_i,where $\forall d_m$, $x_i.d_m$ are independent and identically distributed,then the probability that a point x_i can be a skyline point, $P(x_i \in S)$, is:*

$$P(x_i \in S) = \prod_{j=1, j\neq i}^{n} \left(1 - \prod_{m=1}^{N} \left(p^s_{x_j,d_m} + p^e_{x_j,d_m}\right) + \prod_{m=1}^{N} p^e_{x_j,d_m}\right), \tag{1}$$

where $p^e_{x_j,d_m}$ and $p^s_{x_j,d_m}$ are the probabilities that $x_j.d_m = x_i.d_m$ and $x_j.d_m < x_i.d_m$.

Proof x_i is a skyline point if no point can dominate it. Hence:

$$P(x_i \in S) = \prod_{j=1, j\neq i}^{n} P(x_j \nprec x_i), \tag{2}$$

Let $p^g_{x_j,d_m}$, $p^e_{x_j,d_m}$ and $p^s_{x_j,d_m}$ be the probabilities that $x_j.d_m > x_i.d_m$, $x_j.d_m = x_i.d_m$ and $x_j.d_m < x_i.d_m$, respectively. $p^g_{x_j,d_m} + p^e_{x_j,d_m} + p^s_{x_j,d_m} = 1$. $x_j \nprec x_i$ if $\forall d_m \in D, x_i.d_m = x_j.d_m$ or $\exists d_m, x_i.d_m < x_j.d_m$. Here, $P(x_i.d_m = x_j.d_m)$ and $P(\exists d_m, x_i.d_m < x_j.d_m)$ are:

$$P(\forall d_m, x_i.d_m = x_j.d_m) = \prod_{m=1}^{N} p^e_{x_j,d_m}, \tag{3}$$

$$P(\exists d_m, x_i.d_m < x_j.d_m) = 1 - \prod_{m=1}^{N} \left(p^s_{x_j,d_m} + p^e_{x_j,d_m}\right). \tag{4}$$

Hence, the result follows. □

Theorem 1 estimates the probability of $x_i \in S$. For a random point x, we can obtain:

Theorem 2 *For a random point $x \in X$, its probability of being a skyline point, $P(x \in S)$:*

$$P(x \in S) = \left(1 - \prod_{m=1}^{N} \left(p^s_{d_m} + p^e_m\right) + \prod_{m=1}^{N} p^e_{d_m}\right)^{n-1}, \tag{5}$$

where $p^e_{d_m}$ and $p^s_{d_m}$ are respectively the probabilities of x equals to another point and less than another point in d_m.

Proof In Theorem 2, $p^e_{d_m}$ and $p^s_{d_m}$ are:

$$p^e_{d_m} = \sum_{j=1}^{n} p^e_{x_j,d_m}, \qquad p^s_{d_m} = \sum_{j=1}^{n} p^s_{x_j,d_m}. \tag{6}$$

where $p^e_{x_j,d_m}$ and $p^s_{x_j,d_m}$ follow the same definition in (1). Using the similar arguments in Theorem 1, the result follows. □

Corollary 1 *If values of all dimensions follow a same distribution, then $P(x \in S)$ is:*

$$P(x \in S) = \left(1 - \left(p^s + p^e\right)^N + \left(p^e\right)^N\right)^{n-1}, \tag{7}$$

where p^e and p^s are the probabilities of x equals to another point and less than another point, respectively.

Proof If all dimensions follow a same distribution, then in (5), $p^e_m = p^e_{m+1}$, for $1 \leq m < (N-1)$. □

In Table 2A, all dimensions contain categorical values and fall within 1 to 7. Thus, $p^e_{d_1} = p^e_{d_2} = p^e_{d_3} = p^e_{d_4} = 1/7$ and $p^s_{d_1} = p^s_{d_2} = p^s_{d_3} = p^s_{d_4} = 3/7$. When $N = 2$, $P(x \in S) = 0.112$. When $N = 4$, $P(x \in S) = 0.510$. $P(x \in S)$ increases 4 times when N doubled. If a dataset contains 100K points and all dimensions are continuous with some reasonable ranges, then $p^e \to 0$ and $p^s \to 0.5$. When $N = 2$,

$P(x \in S) \sim 1.3 \times 10^{-12497}$. When $N = 15$, $P(x \in S) \sim 4.7 \times 10^{-2}$. Now, let us analysis the property of dominate-back:

Theorem 3 *Given a random point x and a dimension d_m, the probability that $x \prec_b M(x, d_m)$ is:*

$$P(x \prec_b M(x, d_m)) = 1 - \prod_{\substack{m'=1 \\ m' \neq m}}^{N} \left[1 - (p^s_{d'_m})^{|X| p^s_{d_m}}\right]. \tag{8}$$

where $|X|$ is the cardinality of X.

Proof The probability that x cannot dominate-back $M(x, d_m)$ by using dimension d_n $(d_m \neq d_n)$ is:

$$P(x \not\prec_b M(x, d_m)|d_n) = 1 - (p^s_{d_n})^{\delta}, \tag{9}$$

where $p^s_{d_n}$ is the probability that x is less than another point in d_n and $\delta = |X| - p^s_{d_m}$ is the expected size of $M(x, d_m)$. □

With Theorem 3, we can compute the probability that a point belongs to C^k. Yet, an analytical form will be very complex if each dimension has its own distribution. If all dimensions has the same distribution, then:

Theorem 4 *If all dimensions in a dataset have the same distribution, then $P(x \in C^k)$ is:*

$$P(x \in C^k) = \sum_{n=k}^{N} \binom{N}{n} (p)^n (1 - p)^{N-n}, \tag{10}$$

$$p = \left\{1 - \left[1 - (p^s)^{np^s}\right]^{(N-1)}\right\} \times P(x \in S), \tag{11}$$

where p^s is the probability of x less than another point.

Proof If all dimensions have the same distribution, then in (8), $P(x, d_m) = P(x, d_n)$, $\forall d_m, \forall d_n$. Our problem then reduced to a binomial distribution problem. In addition, $x \in C^k$ only if $x \in S$, so we need the condition of $P(x \in S)$ in (11). □

In Table 2A, $P(x \in C^4) = 0.0012$, $P(x \in C^3) = 0.0182$, $P(x \in C^2) = 0.107$ and $P(x \in C^1) = 0.317$ and $P(x \in C^0) = P(x \in S) = 0.510$. Finally, it is worth noting the following property of C^k:

Property 1 If $k = N$, then the k-dominant core skyline:

$$C^k \supseteq \bigcup_{x \in X} \{x \,|\, x.d_m = \min d_m\} \tag{12}$$

where $\min d_m$ is the minimum value in dimension d_m.

Proof If $x_i.d_k = \min d_k$, then x_i can dominate-back $M(x_i, d_m)$ for all d_m by using d_k. □

According to Property 1, we can see that C^k is never empty for any $k \leq N$. For the properties of C_p^k, the analysis similar to this section can be made, except that we need to use join probability and conditional probability. This is useful because it can show that C^k is not empty, which can hardly be seen from its definition. Also, some existing methods for identifying useful skyline cannot guarantee this property.

4 A simple solution

In this section, we outline a simple but intuitive solution to solve our problem and analyze its time complexity and its space complexity so as to motivate why we need to propose a new solution to solve this problem. Intuitively, since C_p^k is a subset of skyline, it is natural to extract skyline from a dataset first. Let T be the time for identifying skyline. According to Definition 8, for each point $x_i \in S$ and each d_m, we need to identify whether x_i can p-dominate-back $M(x_i, d_m)$ (Condition 1) and whether it can p-dominate-back $M(x_j, d_m)$ for all $x_j \in M(x_i, d_m)$ (Condition 2). As a result, the first thing we need to do is to identify $M(x_i, d_m)$ for all $x_i \in X$. Given a dimension d_m and a point x_i, the time requires to identify $M(x_i, d_m)$ is $O(|X|)$. Since we have N number of dimensions and $|S|$ number of points, the total time requires is $O(N|S||X|)$. To check whether x_i can fulfill Condition 1 is straightforward; however, verifying Condition 2 is very time consuming because we need to check whether $x_i \prec_{pb} M(x_j, d_m), \forall x_j \in M(x_i, d_m)$. One of the possible ways to efficiently verifying Condition 2 is to sort $M(x_i, d_m), \forall x_i, \forall d_m$ first. We give a simple example to explain this. Let $x_j \in M(x_i, d_m)$ and $x_{j'} \in M(x_i, d_m)$. Suppose $x_j.d_m < x_{j'}.d_m$. If $x_i \not\prec_{pb} M(x_j, d_m)$, then it must be true that $x_i \not\prec_{pb} M(x_{j'}, d_m)$ because $x_j \in M(x_{j'}, d_m)$ as $x_j.d_m < x_{j'}.d_m$. Hence, if we have sorted $M(x_i, d_m)$, we can check whether $x_i \prec_{pb} M(x_j, d_m), \forall x_j \in M(x_i, d_m)$ efficiently. The time requires is just $O(p|X|)$ for each d_m (without sorting, the time can be $O(p|X|^2)$). Since we have N dimensions and $|S|$ number of x_i, the total time is $O(pN|S||X|)$. The time requires for sorting all points and dimensions is $O(N|S||X| \log |X|)$ by using some efficient sorting algorithms such as quick sort. By combining all of them (note: $0 \leq p \leq 1$), the time requires is $T + O(N|S||X| + N|S||X| \log |X| + pN|S||X|) = T + O(N|S||X| \log |X|)$. When N is large, $|S| \simeq |X|$. The computational time is somehow too high for real applications.

For the space consumption, let n be the size of d_m. It is not difficult to show that the above framework requires $n \times |X||N||S|$ space in memory if we need to identify C_p^k dynamically. As a result, an efficient algorithm for extracting C_p^k is desirable.

5 Linked multiple B'-tree

In this section, we describe how to extract C_p^k efficiently by using a novel tree structure called LMB. Each tree is like a B^+-Tree that handles one dimension, but the mechanism of handling collision (same key) is different. Therefore, this type of tree is called B'-tree in this paper. All B'-trees are linked together based on the values of the objects. Note that we do not need to identify the skyline points 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.

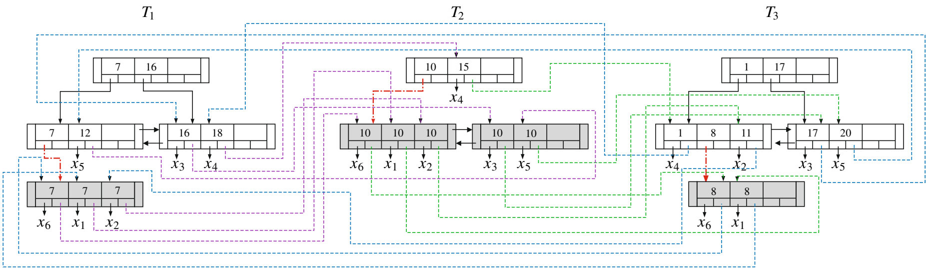
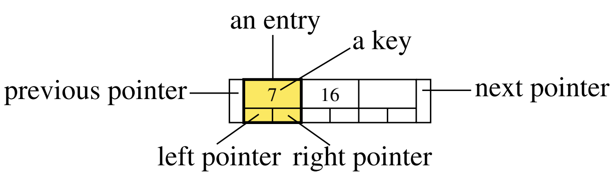


Figure 1 General structure of the Linked Multiple B'-tree (LMB).

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 the 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 $x_6 \to x_1$ (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 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 $(x_1, x_6) \to x_2 \to x_3 \to x_5$. 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

Figure 2 A node in a B'-tree.



them according to the reverse order of their insertion, which is $x_6 \rightarrow x_1$. Eventually, the ordering is $x_6 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_5$.

Algorithm 1: insert (x_i, T_m)

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

Algorithm 2: insert ($x_i, d_m, node$)

```
input : A point, x_i, a dimension, d_m, an overflow node, node
i ← 1;
repeat
    x_j ← the point at entry i in node;
    if m ≠ N then m' ← m + 1; else m' ← 1;
    if x_i.d_m' < x_j.d_m' then insert x_i before x_j; return;
    else if x_j.d_m' = x_i.d_m' then
        for i ← 1 to N − 1 do
            if m' ≠ N then m' ← m' + 1; else m' ← 1;
            if x_i.d_m' < x_j.d_m' then insert x_i before x_j; return;
            else if x_i.d_m' > x_j.d_m' then insert x_i after x_j; return;
        end
        insert x_i before x_j; return;
    end
    i ← i + 1;
    if entry i is empty then insert x_i in entry i; return;
    else if i > number of entries in node then
        i ← 1;    node ← node.next;
    end
until node is null;
node ← a new overflow node;
insert x_i in the first entry of node;
attach node to the last overflow node;
```

5.1 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 having the same value as x_i in d_m (line 2). If not, the insertion process will be the same as a B^+-tree [37] (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 $\forall x_j, x_i.d_{m'} > x_j.d_{m'}$, 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.

5.2 Extracting C_p^k points

Once an 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 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. Lines 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 Conditions 1 and 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 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^k_p. 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^k_p), then x will be added into C^k_p (line 5–8). Note that x is added into C^k_p progressively so we can return results to users immediately and progressively.

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^k_p

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 **end**
5 $C^k_p \leftarrow \emptyset$; $H \leftarrow \emptyset$;
6 **for** $i \leftarrow 1$ **to** $|X|$ **do**
7   **for** $m \leftarrow 1$ **to** N **do**
8   *added* $\leftarrow$ *false*;
9   **for** $n \leftarrow 1$ **to** $N, n \neq m$ **do**
10   **if** $B_{mn} = \emptyset$ *or* $|x^m.d_n - r_m.d_n| < B_{mn}.last$ **then**
11   insert $|x^m.d_n - r_m.d_n|$ into B_{mn};
12   **if** *added* $=$ *false and* $\nexists x \in H, x \prec x^m$ **then**
13                     $C^k_p \leftarrow C^k_p \cup$ `update` (H,k,x^m);
14   *added* $=$ *true*;
15   **end**
16   **if** $\lceil p \times i \rceil > |B_{mn}|$ **then**
17   remove last element from B_{mn};
18   **end**
19   **end**
20   **end**
21         $x^m \leftarrow$ the point closest to x^m (besides itself) in d_m;
22   **end**
23     $i \leftarrow i + 1$;
24 **end**
25 **return** C^k_p;

5.3 Extensions

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 [33], 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.

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

```
   input : A hashtable, H, a parameter, k, a point, x, a value, v ∈ {0,1}
   output: A k-dominant p-core skyline point, x, or ∅
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  end
4  add v into V; // V is the element with key x
5  k' ← number of v in V equals 1;
6  if k' ≥ k and x is not yet returned then
7  |  return x;
8  end
9  return ∅;
```

For the time complexity of extracting C_p^k in Algorithm 3, identifying x^* (line 12) and X_p^* (line 18) is simple and efficient ($O(1)$ for each case). We can immediately tell whether x_m can p-dominate-back $M(x_m, d_m)$ by using d_n and whether x_m can p-dominate-back $M_p(x_m, d_m)$ by using d_n. Since we have N dimensions, we need to compare $N - 1$ dimensions in the worst case. Let $O(T)$ be the time complexity to determine whether a point can be dominated by the points in H (line 10 of Algorithm 3). Eventually, the time complexity for identifying C_p^k will be $O(|X| \cdot N^2 + |X| \cdot T)$. The time complexity is in the order of N^2, however, this in practice is not very high as N is the dimensionality, which is unlikely to be hundreds or thousands.The operation time is therefore more or less feasible in real application. This is also being confirmed by our experiments. Details will be given in Section 6.

For the space complexity, let δ be the size of a value (e.g. 4 byte for an integer). For C_p^k, each dimension stores the values of the other dimensions that it has been dominated. In the worst case (which is extremely unlikely to happen for a reasonable p), we need to store the whole dataset for each dimension. Thus, the space complexity for the worst case is: $O(\delta \cdot n \cdot N^2)$.

6 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 page size of 4KB for each node of LMB. Following [10, 33], 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.

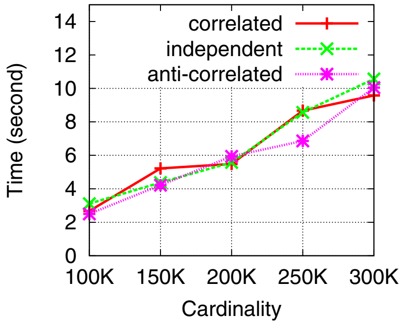
6.1 Effect of cardinality

We set $|X|$ (number of points) = $\{100000, 150000, 200000, 250000, 300000\}$, $N = 15$, $k = 15$ and $p = 1$. Figure 3a 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 s to identify C_p^k from a dataset with 15 dimensions and 100,000 points and spend around 11 s 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 [34] (the most efficient algorithm to identify skyline) takes more than 1,000 s to identify skyline from an independent dataset with 15 dimensions and 100,000 points and takes more than 2,200 s to identify skyline from an anti-correlated dataset with the same setting. Figure 3b 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-correlated 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 3c 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.

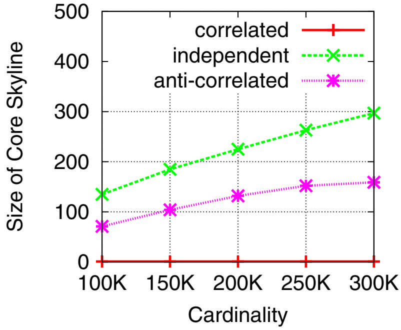
6.2 Effect of dimensionality

We set N (dimensionality) = $\{15, 20, 25, 30, 35\}$, $|X| = 100{,}000$, $p = 1$ and $k = 1$. Figure 4a, b and 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 s 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 Figures 3c and 4c, Figure 3c is more

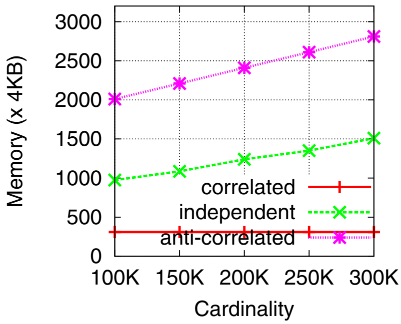
Figure 3 Cardinality.



(a) CPU vs. Cardinality.



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



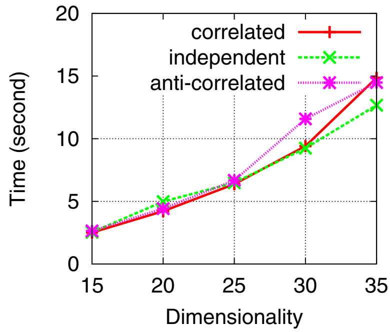
(c) Memory vs. Cardinality.

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.

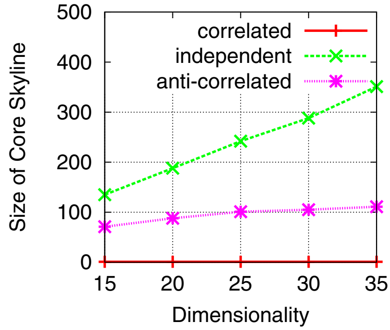
6.3 Effect of k

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 5a, b and 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

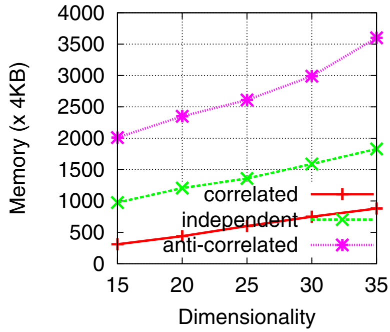
Figure 4 Dim.



(a) CPU vs. Dim.



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



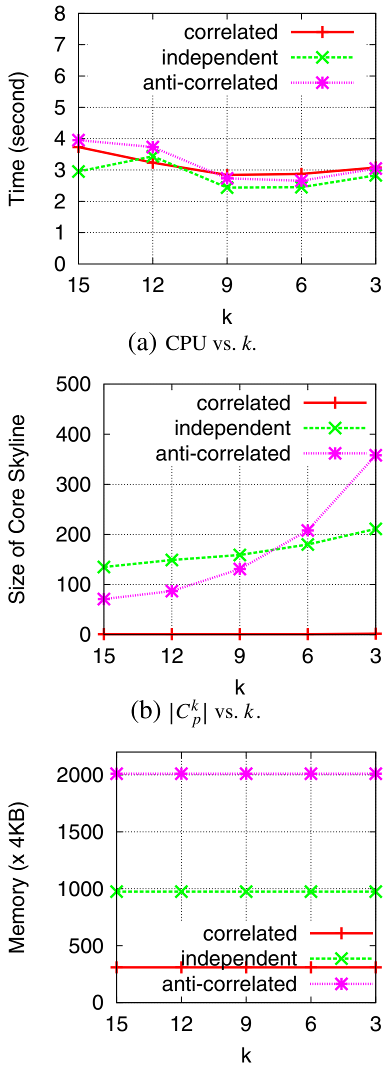
(c) Memory vs. Dim.

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.

6.4 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

Figure 5 Effect of k.



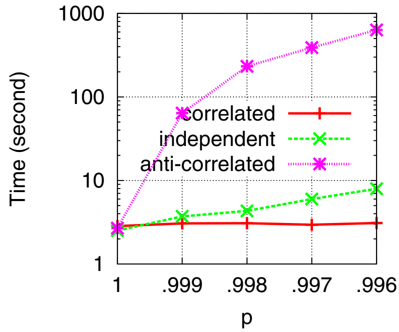
(a) CPU vs. k.

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

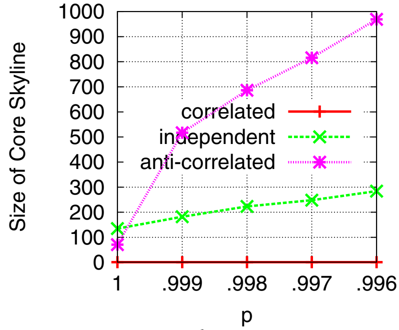
(c) Memory vs. k.

the anti-correlated dataset). In Figure 6a, 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 6b. 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 6c), 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.

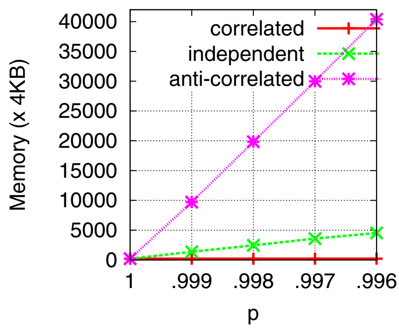
Figure 6 Effect of p.



(a) CPU vs. p.



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



(c) Memory vs. p.

6.5 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. 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

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**)

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 5155, 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.[2] We implement the *k*-dominant skyline algorithm. Chan et al. [10] 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. 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 dataset. This make sense in the basketball situation; (3) The number of skyline points returned by $C_{0.94}^5$ in this dataset is similar to the number of skyline points returned by C_1^5. We can compare the results returned by $C_{0.94}^5$ and C_1^5.

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 $C_{0.94}^5$ and C_1^5, although most names are similar, some names appear in C_1^5 but not

[2] www.databasebasketball.com and www.grouplens.org

Table 4 Players extracted by k-dominant skyline [10].

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**)

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 [10], it extracts one record, Moses Malone, when $6 \leq K \leq 7$ (according to [10], 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.

7 Related work

The skyline computation originates from the maximal vector problem in computational geometry, proposed by Kung et al. [23]. The algorithms developed [4, 28, 33] usually suits for a small dataset with computation done in main memory. One variant of maximal vector problem, which is related to but different from the notion of thick skyline, is the maximal layers problem [32]which aims at identifying different layers of maximal objects.

Borzsonyi et al. [6] first introduce the skyline operator over large databases and propose efficient external memory algorithms for processing skyline queries. The BNL (block-nested-loop) algorithm scans the dataset while employing a bounded buffer for tracking the points that cannot be dominated by other points in the buffer. A point is reported as a result if it cannot be dominated by any other point in the dataset. On the other hand, the DC (divide-and-conquer) algorithm recursively partitions the dataset until each partition is small enough to fit in memory.

After the local skyline in each partition is computed, they are merged to form the global skyline. The method based on [3, 23] partitions the database into memory-fit partitions. The partial skyline objects in each partition is computed by using a main-memory-based algorithm [16, 33], and the final skyline is obtained by merging the partial results. In [40], the authors proposed two progressive skyline computing methods. The first employs a bitmap to map each object and then identifies skyline through bitmap operations. Though the bit-wise operation is fast, the huge length of the bitmap is a major performance concern. The second method introduces a specialized B-tree which is built for each combination list of dimensions that a user might be interested in. Data in each list is divided into batches. The algorithm processes each batch with the ascending index value to find skylines.

The BNL algorithm was later improved to SFS (sort-filter-skyline) [14] and LESS (linear elimination sort for skyline) [19] in order to optimize the average-case running time. The algorithm may require a large number of passes until the complete skyline is computed. The sort filter skyline (SFS) [14] is an improvement of BNL, which first sorts the dataset topologically with the help of a monotone function (e.g., sum of coordinates, assuming they have been normalized). Sorting guarantees that each object cannot be dominated by ones that follow it in the order. As a result, each object that is pushed into the buffer window can immediately be reported as part of the skyline.

The number of passes over the data is then equal to the size of the skyline over the size of the memory buffer. An optimized version of SFS, called linear elimination sort for skyline (LESS), is proposed in [19]. LESS uses a small buffer, called elimination-filter window in the initial pass of the external sort routine of SFS, which keeps a small set of objects used to prune others dominated by them early. Further, LESS combines the last pass of the external sort in SFS with the first filter-scan of SFS (i.e., first pass of the BNL component of SFS).

In SFS and LESS, all objects should be scanned at least once after sorting. Sort and limit skyline algorithm (SaLSa) [2] strives to avoid scanning the complete set of sorted objects. First, the authors suggest an optimal sorting function, which orders the points according to their minimum coordinate value among all dimensions. Second, during the filter-scan process, this method checks whether all points in the remaining dataset are dominated by a so-called stop object that can be determined in O(1) time from the data accessed so far. However, the performance of this method is drastically affected by the data distribution and increasing dimensionality; in high-dimensional problems, the pruning power of the stop object is limited. All sort-based techniques (SFS, LESS, SaLSa) suffer from the large number of computations required during the filter-scan step, as every read point should be compared with the skyline points in the buffer.

The above algorithms are generic and applicable for non-indexed data. On the other hand, Kossmann et al. [22] and Papadias et al. [34] exploited data indexes to accelerate skyline computation. Kossmann et al. [22] present an online algorithm, NN, based on the nearest neighbor search. It gives a big picture of the skyline very quickly in all situations. However, it has raw performance when large amount of skyline needs to be computed. The current most efficient method is BBS (branch and bound skyline), proposed by Papadias et al. [33, 34], which is a progressive algorithm to find skyline with optimal times of node accesses. The state-of-the-art algorithm is shown to be I/O optimal for computing skylines on datasets indexed by R-trees. The

ZSearch algorithm [24] uses a new variant of B^+-tree, called ZBtree, for maintaining the set of candidate skyline tuples in Z-order, which is compatible with the domain correlation. Therefore, index-based approaches have certain limitations that make them useful only for special cases.

As discussed before, Pei et al. [36] and Yuan et al. [45] studied the efficient computation of skylines for every subspace; Tao et al. [41] proposed a technique for retrieving the skyline for a given subspace; They also proposed a sampling scheme that allowed getting an early impression of the skyline for subsequent query refinement. Chaudhuri et al. [12] and Godfrey [18] developed techniques for estimating the skyline cardinality; Godfrey [26] and Sarkas et al. [39] studied continuous maintenance of the skyline over a data stream; Morse et al. [29] examine continuous time-interval skyline queries.

Some studies [8, 9, 12, 30, 42, 43] went beyond skyline evaluation for totally ordered numerical domains and consider partially ordered domains involving categorical or nominal dimensions. Most of them adopt a partial-to-total domain mapping mechanism and then apply existing total order methods, which however suffer from the complex and large size of partially ordered domains [9]. Finally, a lattice skyline (LS) algorithm, introduced in [30], uses a lattice structure to answer skyline queries with dimensions drawn from low-cardinality domains. This method becomes inefficient if the number or size of the domains is large and is not applicable if more than one high-cardinality domain is present. In [46], their focus is on totally ordered domains of high cardinality. They discuss how their methods can be adapted for partially ordered domains. All the above works concerned only the pure dominant relationship and outputted those points which are not dominated by others.

Note that in addition to the original meaning in [6], "dominated" here can be a variant, i.e., k-dominant [10], which found interesting skyline points in high-dimensional space. To tackle the curse of dimensionality, several proposals extended or adapted the definition of skyline in order to consider dimensional subspaces in the dominance relationships between objects [10, 11, 27, 41, 45]. In addition, a top-k query that considers dominance relationships was proposed in [44]. Moreover, efforts have been devoted to dynamic skyline search [13, 38], probabilistic skyline computation [35] and skyline computation over uncertain data [20, 25]. Different from the previous work, Fung et al. [17] further considered not only horizontal comparison but also vertical relationship among dimension, which focus on extracting interesting skyline points in high dimensional space.

8 Conclusion

For Web applications with a large number of data objects described using a rich set of attributes, it is a highly challenging task to make high quality recommendations efficiently. While the concept of skyline query has been proposed to address the problem of the lack of a preference function that can combine multiple attributes to support linear object ranking, this type of query can return too many results when the number of attributes is large, a common case for many Web applications. After an analysis of deficiencies of existing approaches that attempt to reduce the size of skyline-based recommendations, we proposed a novel concept called k-dominant p-core skyline for extracting interesting points from a skyline. This concept is the first to consider the relationship among objects vertically, to reflect the intuition that for an object to be recommended it should perform better in some dimensions than a large portion of the objects that perform better than the object in other dimensions. We use parameter k to denote the number of dimensions that a point has to dominate-back and p to denote the fraction of points in a given dimension that a point has to dominate-back. We gave a theoretical analysis in this paper to establish the foundation of our idea. This idea can be quite computationally demanding to implement as conceptually core skyline points are extracted from the skyline points and the cost to compute the skyline can be very high due to its large size. To make our proposal practical, we designed a new indexing structure called Linked Multiple B'-Trees (LMB) that can generate k-dominate p-core skyline results directly and progressively *without* the need to generate the entire skyline set first. Our main contributions in this paper include (1) a proposal for a new way of making recommendation based on the skyline concept but to make the result set much smaller and meaningful; and (2) a method to implement the proposal to achieve better execution performance than the traditional skyline solutions.

Acknowledgements This work was done partially when Yang, Fung and Lu were at The University of Queensland. Yang and Chen's research is partially supported by the National Science Foundation of China (grant number: 61070056) and National High Technology Research and Development Program 863 of China (grant number: 2008AA01Z120). Lu and Du research is partially supported by National Natural Science Foundation of China (grant number: 60873017).

References

1. Agrawal, R., and Wimmers, E.L.: A framework for expressing and combining preferences. In: Proc. of the 2000 ACM SIGMOD International Conference on Management of Data(SIGMOD 2000), vol. 29, pp. 297–306. ACM, New York (2000)
2. Bartolini, I., Ciaccia, P., Patella, M.: Salsa: computing the skyline without scanning the whole sky. In: Proc. of the 15th ACM international conference on Information and knowledge management(CIKM 2006), pp. 405–414. ACM, New York (2006)
3. Bartolini, I., Ciaccia, P., Patella, M.: Efficient sort-based skyline evaluation. ACM Trans. Database Syst. (TODS), **33**(4), 1–49 (2008)
4. Bentley, J.L., Kung, H.T.T., Schkolnick, M., Thompson, C.D.: On the average number of maxima in a set of vectors and applications. Journal of the ACM (JACM) **25**(4), 536–543 (1978)
5. Böhm, C., Ooi, B.C., Plant, C., Yan, Y.: Efficiently processing continuous k-nn queries on data streams. In: Proc. of the IEEE 23rd International Conference on Data Enginering(ICDE 2007), pp. 156–165 (2007)
6. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. of the 17th International Conference on Data Engineering(ICDE 2001), pp. 421–430. IEEE Computer Society, Washington (2001)
7. Carey, M.J., Kossmann, D.: On saying "enough already!" in SQL. In: SIGMOD, pp. 219–230 (1997)
8. Chan, C.-Y., Eng, P.-K., Tan, K.-L.: Efficient processing of skyline queries with partially-ordered domains. In: Proc. of the 21st International Conference on Data Engineering(ICDE 2005), pp. 190–191. IEEE Computer Society, Washington (2005)
9. Chan, C.-Y., Eng, P.-K., Tan, K.-L.: Stratified computation of skylines with partially-ordered domains. In: Proc. of the 2005 ACM SIGMOD International Conference on Management of Data(SIGMOD 2005), pp. 203–214. ACM, New York (2005)
10. Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proc. of the 2006 ACM SIGMOD International Conference on Management of Data(SIGMOD 2006), pp. 503–514. ACM, New York (2006)

11. Chan, C.Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: On high dimensional skylines. In: Proc. of the 10th International Conference on Extending Database Technology(EDBT 2006), pp. 478–495 (2006)
12. Chaudhuri, S., Dalvi, N., Kaushik, R.: Robust cardinality and cost estimation for skyline operator. In: Proc. of the 22nd International Conference on Data Engineering(ICDE 2006), p. 64. IEEE Computer Society, Washington (2006)
13. Chen, L., Lian, X.: Dynamic skyline queries in metric spaces. In: Proc. of the 11th International Conference on Extending Database Technology(EDBT 2008), pp. 333–343. ACM, New York (2008)
14. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proc. of the 19th International Conference on Data Engineering(ICDE 2003), pp. 717–816 (2003)
15. Das, G., Gunopulos, D., Koudas, N., Sarkas, N.: Ad-hoc top-k query answering for data streams. In: Proc. of the 33rd International Conference on Very Large Data Bases(VLDB 2007), pp. 183–194. VLDB Endowment (2007)
16. Fung, G.P.C., Lu, W., Du, X.: Dominant and k nearest probabilistic skylines. In: Proc. of the 14th International Conference on Database Systems for Advanced Applications(DASFAA 2009), pp. 263–277. Springer-Verlag, Berlin (2009)
17. Fung, G.P.C., Lu, W., Yang, J., Du, X., Zhou, X.: Extract interesting skyline points in high dimension. In: Proc. of 15th International Conference on Database Systems for Advanced Applications (DASFAA 2010), pp. 94–108 (2010)
18. Godfrey, P.: Skyline cardinality for relational processing. In: Foundations of Information and Knowledge Systems, pp. 78–97 (2004)
19. Godfrey, P., Shipley, R., Gryz, J.: Maximal vector computation in large data sets. In: Proc. of the 31st international conference on Very Large Data Bases(VLDB 2005), pp. 229–240. VLDB Endowment (2005)
20. Khalefa, M.E., Mokbel, M.F., Levandoski, J.J.: Skyline query processing for incomplete data. In: Proc. of the 2008 IEEE 24th International Conference on Data Engineering(ICDE 2008), pp. 556–565. IEEE Computer Society, Washington (2008)
21. Kießling, W.: Foundations of preferences in database systems. InL Proc. of the 28th International Conference on Very Large Data Bases(VLDB 2002), pp. 311–322. VLDB Endowment (2002)
22. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proc. of the 28th International Conference on Very Large Data Bases(VLDB 2002), pp. 275–286. VLDB Endowment (2002)
23. Kung, H.T.T., Luccio, F.L., Preparata, F.P.: On finding the maxima of a set of vectors. Journal of the ACM (JACM) **22**(4), 469–476 (1975)
24. Lee, K.C.K., Zheng, B., Li, H., Lee, W.-C.: Approaching the skyline in z order. In: Proc. of the 33rd International Conference on Very Large Data Bases(VLDB 2007), pp. 279–290. VLDB Endowment (2007)
25. Lian, X., Chen, L.: Monochromatic and bichromatic reverse skyline search over uncertain databases. In: Proc. of the 2008 ACM SIGMOD International Conference on Management of Data(SIGMOD 2008), pp. 213–226. ACM, New York (2008)
26. Lin, X., Yuan, Y., Wang, W., Lu, H.: Stabbing the sky: efficient skyline computation over sliding windows. In: Proc. of the 21st International Conference on Data Engineering(ICDE 2005), pp. 502–513. IEEE Computer Society, Washington (2005)
27. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: the k most representative skyline operator. In: Proc. of the IEEE 23rd International Conference on Data Enginering(ICDE 2007), pp. 86–95 (2007)
28. Matoušek, J.: Computing dominances in e^n (short communication). Inf. Process. Lett. **38**(5), 277–278 (1991)
29. Morse, M., Patel, J.M., Grosky, W.I.: Efficient continuous skyline computation. Inf. Sci. **177**(17), 3411–3437 (2007)
30. Morse, M., Patel, J.M., Jagadish, H.V.: Efficient skyline computation over low-cardinality domains. In: Proc. of the 33rd International Conference on Very Large Data Bases(VLDB 2007), pp. 267–278. VLDB Endowment (2007)
31. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding windows. In: Proc. of the 2006 ACM SIGMOD International Conference on Management of Data(SIGMOD 2006), pp. 635–646. ACM, New York (2006)
32. Nielsen, F.: Output-sensitive peeling of convex and maximal layers. Inf. Process. Lett. **59**(5), 255–259 (1996)

33. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proc. of the 2003 ACM SIGMOD International Conference on Management of Data(SIGMOD 2003), pp. 467–478. ACM, New York (2003)
34. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. (TODS) **30**(1), 41–82 (2005)
35. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: Proc. of the 33rd International Conference on Very Large Data Bases(VLDB 2007), pp. 15–26. VLDB Endowment (2007)
36. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the best views of skyline: a semantic approach based on decisive subspaces. In: Proc. of the 31st International Conference on Very Large Data Bases(VLDB 2005), pp. 253–264. VLDB Endowment (2005)
37. Raghu, R., Johannes, G.: Database Management Systems, 3rd edn. McGraw-Hill Science/Engineering/Math (2003)
38. Sacharidis, D., Bouros, P., Sellis, T.: Caching dynamic skyline queries. In: Proc. of the 20th International Conference on Scientific and Statistical Database Management(SSDBM 2008), pp. 455–472. Springer-Verlag, Berlin (2008)
39. Sarkas, N., Das, G., Koudas, N., Tung, A.K.H.: Categorical skylines for streaming data. In: Proc. of the 2008 ACM SIGMOD International Conference on Management of Data(SIGMOD 2008), pp. 239–250. ACM, New York (2008)
40. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient progressive skyline computation. In: Proc. of the 27th International Conference on Very Large Data Bases(VLDB 2001), pp. 301–310. Morgan Kaufmann Publishers Inc., San Francisco (2001)
41. Tao, Y., Xiao, X., Pei, J.: Subsky: efficient computation of skylines in subspaces. In: Proc. of the 22nd International Conference on Data Engineering(ICDE 2006), p. 65. IEEE Computer Society, Washington (2006)
42. Wong, R.C.-W., Fu, A.W.-C., Pei, J., Ho, Y.S., Wong, T., Liu, Y.: Efficient skyline querying with variable user preferences on nominal attributes. PVLDB **1**(1), 1032–1043 (2008)
43. Wong, R.C.-W., Pei, J., Fu, A.W.-C., Wang, K.: Mining favorable facets. In: Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD 2007), pp. 804–813. ACM, New York (2007)
44. Yiu, M.L., Mamoulis, N.: Efficient processing of top-k dominating queries on multi-dimensional data. In: Proc. of the 33rd International Conference on Very Large Data Bases(VLDB 2007), pp. 483–494. VLDB Endowment (2007)
45. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: Proc. of the 31st International Conference on Very large Data Bases(VLDB 2005), pp. 241–252. VLDB Endowment (2005)
46. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: Proc. of the 35th SIGMOD International Conference on Management of Data(SIGMOD 2009), pp. 483–494. ACM, New York (2009)
47. Zhang, Z., Guo, X., Lu, H., Tung, A.K.H., Wang, N.: Discovering strong skyline points in high dimensional spaces. In: Proc. of the 14th ACM International Conference on Information and Knowledge Management (CIKM 2005), pp. 247–248. ACM, New York (2005)