Knowledge-Based Systems 89 (2015) 250–264

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Efficient algorithms for finding the most desirable skyline objects ☆

Yunjun Gao [a,*], Qing Liu [a], Lu Chen [a], Gang Chen [a], Qing Li [b]

[a] *College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China*
[b] *Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong*

ARTICLE INFO

Article history:
Received 29 September 2014
Received in revised form 4 April 2015
Accepted 11 July 2015
Available online 17 July 2015

Keywords:
Skyline
Query processing
Algorithm
Spatial database

ABSTRACT

The skyline query is a powerful tool for multi-criteria decision making. However, it may return too many skyline objects to offer any meaningful insight. In this paper, we introduce a new operator, namely, the *most desirable skyline object* (MDSO) *query*, to identify manageable size of *truly interesting* skyline objects. Given a multi-dimensional object set and an integer *k*, a MDSO query returns the *most preferable k* skyline objects, based on the *newly* defined ranking criterion that considers, for each skyline object *s*, the number of the objects dominated by *s* and their accumulated (potential) weights. We devise the ranking criterion, formalize the MDSO query, and propose three algorithms for processing MDSO queries. In addition, we extend our methods to tackle the *constrained* MDSO (CMDSO) query. Extensive experimental results on both real and synthetic datasets show that our presented ranking criterion is significant, and our proposed algorithms are efficient and scalable.

1. Introduction

Given a set *P* of multi-dimensional data objects, a *skyline query* returns all the data objects from *P*, called *skyline objects*, which are *not dominated* by any other object in *P*. Here, an object *p dominates* another object p' if and only if *p* is not worse than p' in all dimensions, and strictly better than p' in at least one dimension. The skyline query is useful in many real-life applications. Consider, for instance, a classical example of the *hotel reservation system*. Fig. 1 illustrates this case in a 2-dimensional space, where each point corresponds to a hotel record. The room *price* of a hotel is represented as the *y*-axis (the vertical coordinate), and the *x*-axis captures its *distance* to the beach (the horizontal coordinate). Hotel p_2 dominates p_5 since the former is cheaper and closer to the beach. As hotels $p_1,\ p_2,\ p_3$, and p_4 are not dominated by any other hotel, they constitute the *skyline* of a dataset $P = \{p_1, p_2, \ldots, p_8\}$, which offer various trade-offs between price and distance: p_4 has the cheapest room price, p_1 is the closest to the beach, and $p_2,\ p_3$ may be a good compromise of the two attributes.

Since the skyline operator was first introduced to the database community in [5], it has received considerable attention due to its wide applications related to multi-criteria decision making. A large number of algorithms have been proposed for efficient *traditional/full* skyline computation. These approaches can be mainly classified into two categories depending on whether they use indexes or not. The first one [2,5,10,17,48] involves solutions that do not assume any index on the underlying dataset, but they retrieve the skyline by scanning the entire dataset at least once. Methods of the other category [21,27,32,36] avoid accessing the whole dataset by performing the search on an appropriate index structure, e.g., an R*-tree [4]. Other variations of skyline queries include, to name just a few, *subspace* skyline computation [25,34,39], *reverse* skyline query [12,15,28], *metric* skyline computation [9,13], *continuous* skyline retrieval [19,23], *distributed* skyline computation [8,18,40], *uncertain* skyline query [33,47], skyline computation on *data streams* [29,35,38] and *incomplete data* [16], and so forth.

As pointed out in [1,7], however, the skyline query may output an *overwhelming* number of skyline objects, and thus no longer offer any interesting insights, especially in high dimensional spaces. It has been shown in [17] that, for a random dataset, the expected skyline cardinality equals $(\ln m)^{n-1}/n!$, in which *m* is the dataset cardinality, *n* represents the number of dimensions,

☆ This paper is a *full* and *extended* version of the paper, titled "*Finding the Most Desirable Skyline Objects*", which has been published (as a *short paper*) in the *proceedings of the 15th International Conference on Database Systems for Advance Applications* (*DASFAA 2010*), April 1–4, 2010, Tsukuba, Japan. Specifically, the paper extends the *short conference* paper by including (i) informative related work (Section 2); (ii) detailed the *Most Desirable Skyline Object* (MDSO) query processing algorithms, namely, CB and SB, more illustrative examples, and more analyses and proofs (Section 4); (iii) a new MDSO query processing algorithm, i.e., reused based algorithm (Section 4); (iv) processing of the *constrained MDSO* (CMDSO) *query*, a natural variant of MDSO queries (Section 5); and (v) enhanced experimental evaluation that incorporates the new type of queries (Section 6).

* Corresponding author. Tel.: +86 571 8765 1613; fax: +86 571 8795 1250.
E-mail addresses: gaoyj@zju.edu.cn (Y. Gao), liuq@zju.edu.cn (Q. Liu), luchen@zju.edu.cn (L. Chen), cg@zju.edu.cn (G. Chen), itqli@cityu.edu.hk (Q. Li).

http://dx.doi.org/10.1016/j.knosys.2015.07.007
0950-7051/© 2015 Elsevier B.V. All rights reserved.

and $n!$ denotes the factorial of n. To this end, several efforts have been proposed in the literature. In particular, they control the size of skyline objects by either *relaxing the dominance relationship* or *integrating user-specific preference*, as to be surveyed in Section 2. Nonetheless, none of them takes into account the potential weights of non-skyline objects when determining the importance of skyline objects, which is certainly useful. Motivated by this, in this paper, we study *how to find the most desirable skyline objects from the skyline that consists of too many skyline objects* by considering both the number of the non-skyline objects dominated by skyline objects and the accumulated (potential) weights of non-skyline objects. Towards this, we introduce a new operator, namely, the *most desirable skyline object* (MDSO) *query*, to identify *manageable* size of *truly interesting* skyline objects.

Given a set of multi-dimensional objects and an integer k, a MDSO query returns the *most preferable k* skyline objects based on the new *ranking criterion* (defined in Definition 6) that considers, for each skyline object s, the number of the objects dominated by s and their accumulated (potential) weights. Take Fig. 1 as an example. The most desirable 1 skyline object is p_3, because it dominates the maximum number of object/points. The MDSO query is particularly helpful in web-based recommender systems. For instance, consider a tourist looking for a suitable hotel via a *web-based hotel booking system* (e.g., http://hotels.com). Most hotels in a certain city may have to be included in the skyline since, for each hotel p, there might be no one hotel that dominates p on all attributes, even if it is better than p on many attributes. On the other hand, it is difficult for the tourist to make a good, quick selection, by referencing the skyline which contains numerous hotels. In this case, MDSO queries can be employed to retrieve a few preferable hotels, such that the tourist can find a desired hotel as soon as possible.

As to be discussed in Section 2, the MDSO operator is different from the existing works. Hence, the existing techniques are not directly applicable to tackle the MDSO query efficiently. In this paper, we develop three efficient algorithms, i.e., *Cell Based algorithm* (CB), *Sweep Based algorithm* (SB), and *Reuse Based algorithm* (RB), to obtain the most desirable k skyline objects. Our methods are based on a conventional data-partitioning index (e.g., R*-tree [4]) and do not require any preprocessing. Consider that, in some real-life applications, users might enforce some constraints (e.g., spatial region) on MDSO queries. Thus, we extend our techniques to handle a natural variant of MDSO queries, namely, the *constrained most desirable skyline object* (CMDSO) *query*, which returns the *most preferable k* skyline objects in a specified *constrained region*. We also present three efficient algorithms, viz., *Constrained Cell Based algorithm* (CCB), *Constrained Sweep Based algorithm* (CSB), and *Constrained Reuse Based algorithm* (CRB), to deal with CMDSO retrieval.

In brief, the key contributions of this paper are summarized as follows:

- We devise a new ranking criterion and formalize the MDSO query, a new addition to the family of skyline operators for the skyline size control.
- We propose three algorithms, i.e., CB, SB, and RB, for efficiently processing MDSO queries, and analyze their correctness and complexities, respectively.
- We investigate a MDSO query variation, namely, CMDSO retrieval, and present three efficient algorithms to tackle it.
- We conduct extensive experiments with both real and synthetic datasets to demonstrate the effectiveness of our devised ranking criterion and the performance of our proposed algorithms in terms of efficiency and scalability.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formalizes our studied problem. Section 4 elaborates the three algorithms for answering MDSO queries using an R*-tree index on the dataset, and analyzes their correctness and complexities, respectively. The CMDSO query and its processing algorithms are described in Section 5. Extensive experimental evaluation and our findings are reported in Section 6. Finally, Section 7 concludes the paper with some directions for future work.

2. Related work

In this section, we briefly survey the previous work on the skyline query and its variants, and then the skyline operators for skyline size control.

The skyline operator, also known as the maximal vector problem [22], was first introduced into the database community in [5]. Since then a number of algorithms for conventional (i.e., full) skyline queries have been proposed in the literature. They can be mainly classified into (i) non-index based approaches and (ii) index based methods, depending on whether they use indexes or not. Non-index based solutions, including Block-Nested-Loop (BNL) [5], Divide and Conquer (D&C) [5], Sort-First-Skyline (SFS) [10], Linear Elimination Sort for Skyline (LESS) [17], Sort and Limit Skyline algorithm (SaLSa) [2], and Object-based Space Partitioning (OSP) [48], do not require any index on the data set to compute skyline. Index based techniques, including Bitmap [36], Index [36], Nearest Neighbor (NN) [21], Branch and Bound (BBS) [32], and ZSearch [27], require specific indexes for skyline retrieval. It has been proved [32] that BBS is I/O optimal, i.e., it accesses fewer disk pages than any algorithm based on R-trees.

Recently, numerous variations of skyline queries have been studied as well. Examples include, to name but a few, (i) *subspace* skyline computation [25,34,39]; (ii) *reverse* skyline query [12,15,28]; (iii) *metric* skyline retrieval [9,13]; (iv) *continuous* skyline query [19,23]; (v) *distributed* skyline retrieval [8,18,40]; (vi) *uncertain* skyline query [33,47]; and (vii) skyline computation on *data streams* [29,35,38] and *incomplete data* [16], respectively.

However, the skyline operator may return too many skyline objects to offer any meaningful insights. To address this, several efforts on the skyline size control have also been made, by *relaxing the dominance relationship* or *integrating user-specific preference*. Specifically, Koltun and Papadimitriou [20] introduce *approximately dominating representatives*, a refinement of the skyline query that remedies the output volume problem at a small (and controlled) loss of accuracy. Zhang et al. [46] aim to find *strong skyline objects* in high dimensional spaces, which is the union of the skyline objects in all *δ-subspace* (of a space) that contains the

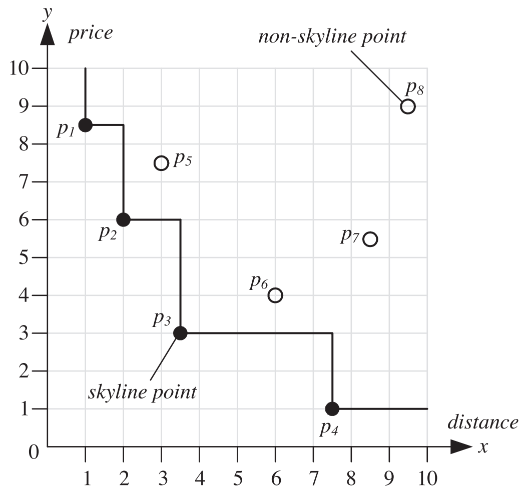


Fig. 1. Example of dataset and skyline.

number of skyline objects less than δ. Chan et al. [7] leverage a new metric, *skyline frequency*, to rank skyline objects. In [6], Chan et al. extend the dominant concept of traditional skyline to *k-dominant*. Xia et al. [43] propose the ε-*skyline* to tune the skyline size. Vlachou and Vazirgiannis [42] propose a framework, namely, SKYRANK, to discover top-*k* most interesting objects of the skyline set through subspace dominance relationships. Moreover, Zhang et al. [45] study the probabilistic top-*k* skyline operator, which retrieves the *k* uncertain objects with the highest skyline probabilities. It is worth noting that, unlike the aforementioned work, our MDSO operator still adopts the conventional dominance relationship (Definition 1), while it takes the potential weight of non-skyline objects into consideration, which is neglected by the above work.

Lin et al. [30] first introduce the concept of *representative skyline*, which is based on dominance relationship. Tao et al. [37] propose a new definition of representative skyline, namely, *distance-based representative skyline*. Vlachou et al. [41] present a novel framework for discovering the representative skyline over distributed data sources, which incorporates the above two metrics, i.e., dominance-based representative [30] and distance-based representative [37]. Das et al. [11] define a new *k* representative skyline object by ensuring that the probability which a random user would click on one of them is maximized. More recently, Lee and Hwang [24] develop an efficient greedy algorithm for the *k* representative skyline using the *skytree*. As proved in [11,30,37,41], the problem of representative skyline retrieval is NP-hard for the dimensionality at least 3. Compared with the representative skyline, the MDSO query employs a new ranking criterion defined in Definition 6, and it is not NP-hard in high dimensional spaces. Furthermore, the representative skyline retrieval does not consider the potential weights of non-skyline objects, which is also taken into account in the MDSO query.

Balke et al. [1] and Lee et al. [26] obtain manageable number of the skyline objects by integrating user-specific qualitative preferences/functions. Vlachou and Vazirgiannis [42] extend SKYRANK to handle the top-*k* preference skyline query, when the user's preferences are available. Bartolini et al. [3] proposes the collaborative filtering skyline, a general framework that generates a personalized skyline for each active user based on scores of other users with similar scoring patterns. Different from [1,3,26,42], our MDSO operator does not rely on any user-specified function, since it is not always easy for users to provide such a function.

In addition, it is worth mentioning that the MDSO query is fundamentally different from the existing *top-k dominating* (TKD) *query* [44] that retrieves *k* data objects dominating the highest number of objects in the dataset. First, their result sets are different. The MDSO query always returns skyline objects, whereas the TKD query may return non-skyline objects. Second, they adopt different ranking criteria. The ranking criterion (Definition 6) utilized in the MDSO query considers the number of the objects dominated by each skyline object as well as their accumulated (potential) weights. On the other hand, the TKD query only takes the number of the dominated objects into consideration. Thus, the existing algorithms for TKD queries cannot be applied to handle MDSO queries efficiently.

Table 1
Symbol and description.

Notation	Description
P	A set of data objects
n	Dimensionality of a data space
D	A data space
k	The number of required skyline objects
d_i	The *i*th dimension
$p,\ p',\ p''$	A data point/object
$p.d_i$	The *i*th dimensional value of p
$p \prec p'$	p dominates p'
S or s or ns	The set of skyline objects or a skyline object or a non-skyline object
$\mu(s)$	The dominating score of a skyline object s
$\omega(ns)$	The dominated score (i.e., potential weight) of a non-skyline object ns
$\tau(s)$	The preference score of a skyline object s
$s \vdash s'$	A skyline object s is more desirable than another skyline object s'

3. Preliminaries

In this section, we formally define the ranking criterion and the MDSO query in Sections 3.1 and 3.2, respectively. Table 1 lists the symbols frequently used in the rest of the paper.

3.1. Ranking criterion

Let P be a set of data objects in an *n*-dimensional space $D = (d_1, d_2, \ldots, d_n)$, where $d_i (i \in [1, n])$ is the *i*th dimension. For any object $p \in P$, we use $p.d_i$ to denote the *i*th dimensional value of p. Assume that there exists a total order relationship, either '<' or '>', on each dimension. Without loss of generality, in this paper, we consider '<' relationship, i.e., smaller values are more preferable.

Definition 1 (***Dominance***). For any two objects $p, p' \in P, p$ is said to *dominate* p', denoted by $p \prec p'$, iff (i) $\forall d_i \in D, p.d_i \leqslant p'.d_i$, and (ii) $\exists d_j \in D, p.d_j < p'.d_j$.

As an example, in Fig. 1, p_3 dominates p_6 because $p_3.x(= 3.5) < p_6.x(= 6)$ and $p_3.y(= 3) < p_6.y(= 4)$.

Definition 2 (***Skyline Object, Skyline, and Non-skyline Object***). An object $p \in P$ is a *skyline object s* iff p is not dominated by any other object $p'\ (\neq p) \in P$, i.e., $\nexists p' \in P - \{p\}, p' \prec p$. The *skyline* of P is the set S of skyline objects on the dataset P. An object p'' is a *non-skyline object ns* iff there exists at least one object $p'\ (\neq p'') \in P$ dominating p'', i.e., $\exists p' \in P - \{p''\}, p' \prec p''$.

Continuing the above example, p_3 is a skyline object and $p_3 \prec p_6$, and hence p_6 is a non-skyline object. To sum up, in Fig. 1, the skyline object set $S = \{p_1, p_2, p_3, p_4\}$, and points $p_5,\ p_6,\ p_7$, and p_8 are all non-skyline objects.

Definition 3 (***Dominating Score***). Let S be the skyline of P, for a skyline object $s \in S$, the *dominating score* of s, denoted by $\mu(s)$, could be defined as:

$$\mu(s) = \mid \{ns \in P - S \mid s \prec ns\} \mid \tag{1}$$

In other words, the score $\mu(s)$ is the number of the non-skyline objects dominated by a skyline object s. The higher the $\mu(s)$ is, the more interesting the skyline object is, as pointed out in [30,44]. In Fig. 1, for instance, p_3 is more desirable than p_1 since $\mu(p_3) = 3$ and $\mu(p_1) = 1$. Thus, we can derive a natural ordering of skyline objects according to the dominating score. Nevertheless, when two skyline objects share the same dominating score, the tie needs to be broken. Towards this, one possible approach is to request users to provide some weight assignments for their preferred attributes. Unfortunately, providing such weight assignments are not always easy without any initial knowledge about the data. On the other hand, as stated in [5], the goal of offering the skyline to the users is to help them determine the weight assignment. Another possible work-around is to take the potential weight of every non-skyline object into consideration, by computing the number of the objects dominated by it. However, this method incurs expensive computational cost, as it has to count the number of the objects that are dominated by each non-skyline object. Based on these observations, in the following, we present an alternative approach to break the possibly occurred tie.

Definition 4 (***Dominated Score***). Let S be the skyline of P, for a non-skyline object $ns \in P - S$, the *dominated score* (i.e., potential weight) of ns, denoted by $\omega(ns)$, could be defined as:

$$\omega(ns) = \frac{1}{|\{s \in S | s \prec ns\}|} \tag{2}$$

In fact, the dominated score of a non-skyline object considers the number of the skyline objects dominating it. Intuitively, a non-skyline object might have larger weight if it is dominated by as few skyline objects as possible, indicating that it may dominate more other non-skyline objects. As an example, in Fig. 1, the weight of p_5 is greater than that of p_8, due to $\omega(p_5) = 1$ and $\omega(p_8) = 1/4$. Consequently, we define the preference score of a skyline object based on the intuition: a skyline object is more interesting if it dominates as many the non-skyline objects with higher dominated scores as possible.

Definition 5 (***Preference Score***). Let S be the skyline of P, for a skyline object $s \in S$, the *preference score* of s, denoted by $\tau(s)$, is defined as:

$$\tau(s) = \sum_{ns' \in \{ns \in P - S | s \prec ns\}} \omega(ns') \tag{3}$$

Take Fig. 1 as an example. p_2 is more preferred than p_4 as $\tau(p_2) = 5/4$ and $\tau(p_4) = 3/4$, although $\mu(p_2) = \mu(p_4) = 2$. Actually, the preference score of a skyline object takes into account the accumulated (potential) weight of all non-skyline objects dominated by it. A skyline object s having a higher score $\tau(s)$ might be more preferable for users. Based on Eqs. (1)–(3), the *ranking criterion* is formalized as follows:

Definition 6 (***Ranking Criterion***). Let S be the skyline of P, for two skyline objects $s, s' \in S$, s is more desirable than s', denoted by $s \vdash s'$, iff

$$s \vdash s' \iff \begin{cases} \mu(s) > \mu(s') \\ \mu(s) = \mu(s') \wedge \tau(s) > \tau(s') \end{cases} \tag{4}$$

For example, in Fig. 1, $p_3 \vdash p_4$ and $p_2 \vdash p_4$ according to Definition 6.

3.2. Problem formulation

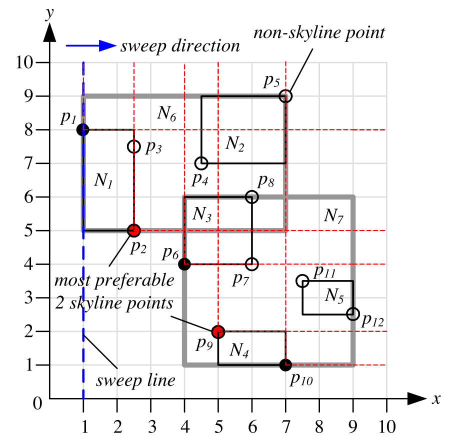
We now present the formal definition of the MDSO query based on the ranking criterion defined in Section 3.1.

Definition 7 (***Most Desirable Skyline Object Query***). Given a set P of data objects, an integer k ($\geqslant 1$), and let S be the skyline of P, the *most desirable skyline object* (MDSO) *query* retrieves the set S_r of skyline objects, such that (i) S_r contains k skyline objects, i.e., $|S_r| = k$; and (ii) none of object $p' \in S - S_r$ is superior to any answer object $p \in S_r$ according to the ranking criterion (see Definition 6), i.e., $\forall p \in S_r$ and $p' \in S - S_r$, $\mu(p) > \mu(p')$ or $\mu(p) = \mu(p') \wedge \tau(p) > \tau(p')$. Note that, when $|S| \leqslant k$, S is the result set, i.e., $S_r = S$.

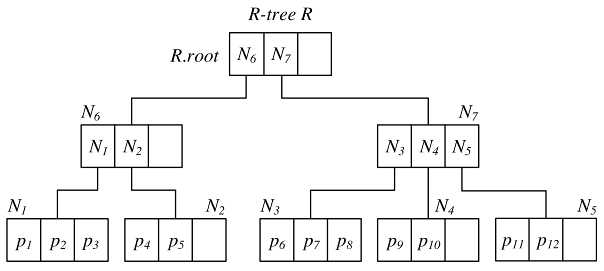
It is worth noting that, if two skyline objects with the same dominating score and preference score are tie, only one of them is returned. The MDSO query is a new addition to the family of skyline operators for the skyline size control. Compared with the existing technique, it takes a slightly different perspective on the problem of controlling the size of skyline object set. Take Fig. 1 as an example again. If $k = 2$, the MDSO query on the dataset illustrated in Fig. 1 returns p_3 (with $\mu(p_3) = 3$ and $\tau(p_3) = 7/4$) and p_2 (with $\mu(p_2) = 2$ and $\tau(p_2) = 5/4$) as the most preferable 2 skyline objects.

4. MDSO query processing

In this section, we propose three algorithms for processing MDSO queries, i.e., *Cell Based algorithm* (CB), *Sweep Based algorithm* (SB), and *Reuse Based algorithm* (RB), and analyze their correctness and complexities, respectively. All the algorithms take an R-tree R on a dataset and an integer k as input, and output the most desirable k skyline objects. For ease of understanding, a running example, as shown in Fig. 2, is employed, where the two-dimensional (2D) data point set in Fig. 2(a) is organized in the R-tree depicted in Fig. 2(b) with node capacity three. Assume that the number k



(a) The data point placement



(b) The R-tree R

Fig. 2. A running example.

of required skyline objects is 2 (i.e., $k = 2$). The final query result contains points p_9 and p_2.

4.1. Cell based algorithm

Once the skyline S of a given data set P is got, the region dominated by all skyline objects in S can be divided into a lot of *cells*. For simplicity, we consider the partition of cells in a 2D space (x, y). The presented concepts, however, can be easily extended to high dimensional spaces.

Suppose $S = \{s_1, s_2, \ldots, s_m\}$ is the skyline of a specified data set P in a 2D space. The region dominated by skyline objects in S can be partitioned into $m \cdot (m+1)/2$ cells $\{C_{ab} | 1 \leqslant a \leqslant b \leqslant m\}$. The *lower-left* corner and the *upper-right* corner of every cell C_{ab} are $(s_b.x, s_a.y)$ and $(s_{b+1}.x, s_{a-1}.y)$, respectively. Note that, when $a = 1$ $(b = m)$, $s_0.y$ $(s_{m+1}.x)$ is defined as the maximal value of $y(x)$ coordinate in the data space. As illustrated in Fig. 2(a), for instance, the shadowed region dominated by $\{p_1, p_2, p_6, p_9, p_{10}\}$ is divided into 15 cells. Based on the concept of cells, we can derive Lemma 1 below.

Lemma 1. *Any data object or R-tree node falling completely into a single cell is only dominated by the same skyline objects.*

Proof. The proof is straightforward by the partition of cells. □

In other words, data objects that are within a single cell, but not on the boundaries of the cell, have *the same* (*potential*) *weight*. For example, in Fig. 2(a), points p_{11} and p_{12} contained in an R-tree node N_5 are dominated by the same skyline points p_9 and p_{10} since they are located inside cell C_{45}. Nevertheless, when an R-tree node intersects *multiple* cells, the data objects included in the R-tree node may be dominated by *different* skyline objects. Take Fig. 2(a) as an example again. As the R-tree node N_2 crosses four cells, point p_4 in N_2 are only dominated by skyline points p_2 and p_6, whereas point p_5 in N_2 is dominated by skyline points p_2, p_6, p_9, and p_{10}.

Algorithm 1. Cell Based Algorithm (CB)

```
Input: an R-tree $R$ on a set of data objects, the number $k$ of skyline objects to return
Output: the most desirable $k$ skyline objects
1:  initialize a set $S = \emptyset$ accepting skyline objects $s$ in the form $\langle s, \mu(s), \tau(s) \rangle$ and two min-heaps $H = H' = \emptyset$   // $H$ and $H'$ are sorted in ascending order of $L_1$-norm
2:  insert all entries of the root $R.root$ into $H$ and $H'$ respectively   // $R.root$: root of an R-tree $R$
3:  compute skyline $S$ using BBS algorithm with $H'$   // algorithm of [32]
4:  while $H \neq \emptyset$ do
5:    de-heap the top entry $e$ of $H$
6:    if $e$ is a data object and $e \notin S$ then   // $e$ is a non-skyline object
7:      $S' = \{s \in S \mid s \prec e\}$   // $s \in S$ dominates $e$
8:      for each object $s \in S'$ do
9:        $\mu(s) = \mu(s) + 1$ and $\tau(s) = \tau(s) + 1/|S'|$
10:   else   // $e$ is an intermediate (i.e., a non-leaf) node
11:     if $e$ is inside a single cell then
12:       $cnt$ = ObjectCount ($e$)
13:       $S'' = \{s \in S \mid s \prec e\}$
14:       for each object $s \in S''$ do
15:         $\mu(s) = \mu(s) + cnt$ and $\tau(s) = \tau(s) + 1/\mid S'' \mid \times cnt$
16:     else   // $e$ spans multiple cells
17:       for each child entry $e_i \in e$ do
18:         en-heap $e_i$ into $H$
19: sort all skyline objects in $S$ according to the ascending order based on Eq. (4)
20: return the top-$k$ objects in $S$

Function ObjectCount ($N$)
Input: a node MBR $N$ being evaluated
Output: the number $counter$ of data objects in the subtree rooted at $N$
1:  initialize a stack $st = \emptyset$ and $counter = 0$
2:  push $N$ into $st$
3:  while $st \neq \emptyset$ do
4:    pop the top entry $e$ out of $st$
5:    if $e$ is a leaf node then   // $e$ lies in the second lowest level of the R-tree
6:      let $num$ be the number of child entries in $e$
7:      $counter = counter + num$
8:    else   // $e$ is a non-leaf node
9:      for each child entry $e_i \in e$ do
10:       push $e_i$ into $st$
11: return $counter$
```

Our first approach, namely, *Cell Based algorithm* (CB), utilizes the aforementioned cell properties. The basic idea of CB is to compute the skyline S using BBS algorithm [32], and then, for each skyline object $s \in S$, calculate its dominating score $\mu(s)$ and preference score $\tau(s)$ respectively, and finally return the top-k skyline objects in S according to our proposed ranking criterion (Definition 6). The pseudo-code of CB is presented in Algorithm 1. Initially, CB initializes a set $S = \emptyset$ accepting skyline objects s of the form $\langle s, \mu(s), \tau(s) \rangle$, and two min-heaps H and H' sorted in ascending order of the minimum L_1-norm distance from the origin to an R-tree node or a data object (line 1). Then, it employs BBS algorithm to obtain the skyline preserved in S (line 3). Here, $\mu(s) = \tau(s) = 0$ for each skyline object $s \in S$. Thereafter, CB computes $\mu(s)$ and $\tau(s)$ by traversing the R-tree R (lines 4–18). In particular, if a non-skyline object is accessed, for every object s in the set S' of all the skyline objects dominating it, $\mu(s)$ and $\tau(s)$ are increased by 1 and $1/|S'|$, respectively (lines 6–9). When an R-tree node e is encountered, CB distinguishes two cases. (i) If e is within a single cell, the algorithm first invokes *ObjectCount* function to get the number *cnt* of data objects in the subtree rooted at e, and then, for each object s in the set S'' of all the skyline objects dominating e, $\mu(s)$ and $\tau(s)$ are increased by *cnt* and $1/|S''| \times cnt$, respectively (lines 11–15). (ii) If e intersects multiple cells, the algorithm inserts the child entries of e into the heap H for expansion (lines 17–18). This traversal terminates once $H = \emptyset$. Finally, CB sorts all skyline objects in S based on Eq. (4) and returns the top-k objects in S as the final query result (lines 19–20).

Back to the running example depicted in Fig. 2. First, CB uses BBS algorithm to compute the skyline $S = \{\langle p_9, 0, 0 \rangle, \langle p_2, 0, 0 \rangle, \langle p_{10}, 0, 0 \rangle, \langle p_6, 0, 0 \rangle, \langle p_1, 0, 0 \rangle\}$. Then, for each skyline object $s \in S$, it calculates $\mu(s)$ and $\tau(s)$ by traversing the R-tree R, after which $S = \{\langle p_9, 5, 61/30 \rangle, \langle p_2, 4, 61/30 \rangle, \langle p_{10}, 3, 6/5 \rangle, \langle p_6, 4, 23/15 \rangle, \langle p_1, 1, 1/5 \rangle\}$. After sorting, objects p_9 and p_2 are returned as the most desirable 2 skyline objects.

4.2. Sweep based algorithm

CB first traverses the R-tree R on the data set P once to obtain the skyline S of P, and then, for each skyline objects $s \in S$, it calculates $\mu(s)$ and $\tau(s)$ by traversing R once. Totally, the CB algorithm traverses R twice. Hence, it is not efficient in terms of the I/O cost (i.e., the number of node accesses) and CPU time, especially in high dimensional spaces. Motivated by this, we present an alternative, namely, *Sweep Based algorithm* (SB). First, we offer the lemma that is used by SB.

Lemma 2. *If a non-skyline object is encountered by the sweep line (i.e., a vertical line to iteratively sweep along coordinate axis), it is only dominated by the skyline objects that have been obtained so far.*

Proof. Suppose a non-skyline object ns encountered by the sweep line is dominated by a skyline object s found later. Then, according to Definition 1, s is smaller than or equal to ns in all dimensions and strictly smaller than ns in at least one dimension. This means that s must be encountered by the sweep line prior to ns, which contradicts with our assumption. Consequently, the proof completes. □

The main idea of SB is to identify skyline objects, and meanwhile calculate their dominating scores and preference scores, via a *single* traversal of the R-tree R used to organize a dataset P. Algorithm 2 shows the pseudo-code of SB algorithm. Without loss of generality, SB uses a vertical line to iteratively sweep along x dimension *from left to right* for obtaining skyline objects s, $\mu(s)$, and $\tau(s)$. In the first place, SB initializes a set $S = \varnothing$ accepting skyline objects s in the form $\langle s, \mu(s), \tau(s)\rangle$, and a min-heap H sorted lexicographically on n-dimensional space $D = (d_1, d_2, \ldots, d_n)$ (line 1), e.g., the data points in Fig. 2(a) are sorted as $\{p_1, p_2, p_3, p_6, p_4, p_9, p_7, p_8, p_{10}, p_5, p_{11}, p_{12}\}$. It then recursively finds skyline objects s and computes $\mu(s)$ and $\tau(s)$ until $H = \varnothing$ (lines 3–17). Specifically, for the data objects encountered by the sweep line, SB processes them one by one from bottom to up. If a data object p evaluated currently is a skyline object, it is added to S directly. Otherwise, p must be a non-skyline object, and its dominated score $\omega(p)$ can be determined because, as proved by Lemma 2, it is only dominated by some skyline objects that have been identified so far, but not dominated by skyline objects retrieved later. Thus, for every object s in the set S' of all skyline objects dominating p, $\mu(s)$ and $\tau(s)$ are increased by 1 and $1/|S'|$, respectively. The sweep stops after all data objects are accessed. Finally, the top-k objects in S are returned as the final query result after sorting (lines 18–19).

Algorithm 2. Sweep Based Algorithm (SB)

```
Input: an R-tree $R$ on a set of data objects, the number $k$ of
   skyline objects to be returned
Output: the most desirable $k$ skyline objects
1:  initialize a set $S = \varnothing$ accepting skyline objects $s$ in the
    form $\langle s, \mu(s), \tau(s)\rangle$ and a min-heap $H = \varnothing$   // $H$ is sorted
    lexicographically on $n$-dimensional space
2:  insert all entries of the root $R.root$ into $H$   // $R.root$: the
    root of an R-tree $R$
3:  while $H \neq \varnothing$ do
4:    de-heap the top entry $e$ of $H$
5:    if $e$ is a data object then
6:      if $S = \varnothing$ then
7:        add $\langle e, 0, 0\rangle$ to $S$   // $e$ is the first skyline object
8:      else   // $S \neq \varnothing$
9:        $S' = \{s \in S | s \prec e\}$   // $s \in S$ dominates $e$
10:        if $S' = \varnothing$ then   // $e$ is a skyline object
11:          insert $\langle e, 0, 0\rangle$ into $S$
12:        else   // $e$ is a non-skyline object
13:          for each point $s \in S'$ do
14:            $\mu(s) = \mu(s) + 1$ and $\tau(s) = \tau(s) + 1/|S'|$
15:    else   // $e$ is an intermediate node
16:      for each child entry $e_i \in e$ do
17:        en-heap $e_i$ into $H$
18:  sort all skyline objects in $S$ according to the ascending
     order based on Eq. (4)
19:  return the top-$k$ objects in $S$
```

Again, back to our running example. First, point p_1 is encountered by the sweep line (blue[1] dashed line in Fig. 2(a)). Since the current skyline S is empty, p_1 is added to $S = \{\langle p_1, 0, 0\rangle\}$ as the first skyline object. The second object accessed is p_2. It is also inserted into $S = \{\langle p_1, 0, 0\rangle, \langle p_2, 0, 0\rangle\}$, as p_2 is not dominated by p_1. Then, SB visits p_3 and updates S to $\{\langle p_1, 0, 0\rangle, \langle p_2, 1, 1\rangle\}$. The algorithm proceeds in the same manner until all data objects are visited, after which $S = \{\langle p_1, 1, 1/5\rangle, \langle p_2, 4, 61/30\rangle, \langle p_6, 4, 23/15\rangle, \langle p_9, 5, 61/30\rangle, \langle p_{10}, 3, 6/5\rangle\}$. Finally, SB reports objects p_9 and p_2 as the final query result after sorting S.

4.3. Reused based algorithm

As mentioned in Section 4.2, the CB algorithm traverses R two times: one for obtaining the skyline of the dataset, and another for calculating the dominating score and preference score of every skyline object. If we keep the visited nodes in the first step (i.e., the skyline computation) and then use them in the second step (i.e., the calculation of the dominating score and preference score for each skyline object), it only needs to traverse R a single once. In addition, *aggregation R-Tree* aR-Tree [31] preserves the value of the aggregation function for all objects that are enclosed by it, which is very useful for an aggregation query. To this end, in this section, we present an efficient algorithm, i.e., *Reuse Based algorithm* (RB), which further boosts the query performance by employing the reuse technique and the aR-Tree. In what follows, we firstly introduce the aR-Tree.

The aR-tree was first proposed in [31], which combines a spatial index with the materialization technique. The aR-tree is an R-Tree, which stores for each minimum bounding rectangle (MBR), the value of the aggregation function (in this paper, the aggregation function is *COUNT*) for all objects that are contained by it. Therefore, an aggregation query does not need to access all the enclosed objects, since the answer can be found in the intermediate nodes of the tree, which reduces the I/O overhead. Fig. 3 illustrates the aR-tree *aR* that indexes the dataset in Fig. 2(a). For the leaf nodes, *aR* stores the real data objects just like the R-tree R. As for the entries of intermediate nodes, *aR* not only maintains the information of MBRs, but also records the total number of data objects included in their corresponding sub-tree (i.e., the value of the function *COUNT*). In Fig. 3, the leaf nodes $N_1, N_2, \ldots, N_5$ are the same as those Fig. 2(b). Moving one level up, MBR N_1 contains three objects p_1, p_2, and p_3, and hence, the *COUNT* of N_1 is 3. Moving one more level up, in the *aR.root*, there are two entries N_6 and N_7. Since N_6 contains N_1 and N_2, the total number of the objects in N_6 is 5.

By using the aR-tree, when the node MBR falls completely into a single cell, we can get the dominating score and preference score directly, rather than calling the function *ObjectCount*, which results in the fewer I/O cost. In addition, we provide the following lemma to guarantee the correctness of the algorithm.

[1] For interpretation of color in Fig. 2, the reader is referred to the web version of this article.

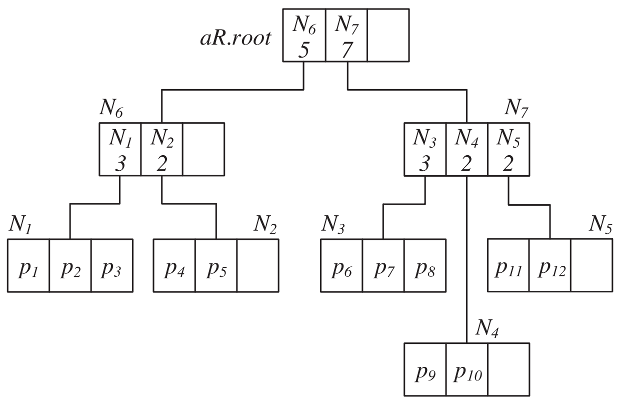


Fig. 3. Example of the aR-tree *aR* for Fig. 2(a).

Lemma 3. *The nodes visited in the computation of skyline can also be reused in the calculation of the dominating score and the preference score for every skyline object.*

Proof. It is obvious because both the computation of the skyline and the calculation of the dominating score and the preference score use the same aR-tree structure. □

Algorithm 3. Reuse Based Algorithm (RB)

```
Input: a COUNT aR-tree aR on a set of data objects, the
  number k of skyline objects to be returned
Output: the most desirable k skyline objects
/*aR.root: the root of the COUNT aR-tree aR */
 1:  initialize a set $S = \varnothing$ accepting skyline objects $s$ in the
     form $\langle s, \mu(s), \tau(s)\rangle$ and two min-heaps $H = H' = \varnothing$
 2:  insert all entries of the root aR.root into $H'$
 3:  compute skyline $S$ using BBS algorithm with $H'$, and
     meanwhile maintain all the data objects and node MBRs
     dominated by some skyline object in $S$ into $H$
 4:  while $H \neq \varnothing$ do    // reuse all the entries preserved in $H$
 5:    de-heap the top entry $e$ of $H$
 6:    if $e$ is a data object then    // $e$ must be a non-skyline
     object
 7:      $S' = \{s \in S \mid s \prec e\}$    // $s \in S$ dominates $e$
 8:      for each object $s \in S'$ do
 9:        $\mu(s) = \mu(s) + 1$ and $\tau(s) = \tau(s) + 1/|S'|$
10:    else    // $e$ is an intermediate node
11:      if $e$ is inside a single cell then
12:        $S'' = \{s \in S \mid s \prec e\}$
13:        for each object $s \in S''$ do
14:          $\mu(s) = \mu(s)$ + COUNT($e$) and
     $\tau(s) = \tau(s) + 1/|S''| \times$ COUNT($e$)
15:      else// $e$ spans multiple cells
16:        for each child entry $e_i \in e$ do
17:          en-heap $e_i$ into $H$
18:  sort all skyline objects in $S$ according to the ascending
     order based on Eq. (4)
19:  return the top-$k$ objects in $S$
```

Our third approach, i.e., RB algorithm, utilizes the aforementioned aR-tree and reuse technique. The basic idea of RB is to compute the skyline S using BBS algorithm just like CB. But, unlike CB, during the skyline computation, all the data objects and node MBRs dominated by some skyline objects in S have to be kept. Then, for each skyline object $s \in S$, RB calculates its dominating score $\mu(s)$ and preference score $\tau(s)$ respectively, using the visited nodes stored in the previous step. Finally, RB returns the top-k skyline objects in S according to our proposed ranking criterion (Definition 6). The pseudo-code of CB is presented in Algorithm 3. Initially, CB initializes a set $S = \varnothing$ accepting skyline objects s of the form $\langle s, \mu(s), \tau(s)\rangle$, and two min-heaps H and H' sorted in ascending order of the minimum L_1-norm distance from the origin to an aR-tree node or a data object (line 1). Then, it employs the BBS algorithm to obtain the skyline by traversing the aR-tree *aR* and meanwhile storing the visited nodes (line 3). After this step is completed, all the data objects and node MBRs dominated by some skyline objects in S are maintained in H, which is used for computing $\mu(s)$ and $\tau(s)$ later (lines 4–17). In particular, if a non-skyline object is accessed, for every object s in the set S' of all the skyline objects dominating it, $\mu(s)$ and $\tau(s)$ are increased by 1 and $1/|S'|$ respectively (lines 6–9). When an aR-tree node e is encountered, RB distinguishes two cases. (i) If e is within a single cell, for each object s in the set S'' of all the skyline objects dominating e, $\mu(s)$ and $\tau(s)$ are increased by *COUNT*(e) and $1/|S''| \times$ *COUNT*(e) respectively (lines 11–14). (ii) If e crosses multiple cells, the algorithm inserts the child entries of e into the heap H for expansion (lines 16–17). This traversal terminates once $H = \varnothing$. Finally, RB sorts all skyline objects in S based on Eq. (4), and returns the top-k objects in S as the final query result (lines 19–20).

Back to the running example depicted in Fig. 2 again. First, RB uses BBS algorithm to compute skyline $S = \{\langle p_9, 0, 0\rangle, \langle p_2, 0, 0\rangle, \langle p_{10}, 0, 0\rangle, \langle p_6, 0, 0\rangle, \langle p_1, 0, 0\rangle\}$, after which $H = \{p_1, p_2, p_3, p_6, p_7, p_8, p_9, p_{10}, N_2, N_5\}$. Then, for each skyline object $s \in S$, it calculates $\mu(s)$ and $\tau(s)$ by traversing the entries in H, after which $S=\{\langle p_9,5,61/30\rangle,\langle p_2,4,61/30\rangle,\langle p_{10},3,6/5\rangle,\langle p_6,4,23/15\rangle,\langle p_1,1,1/5\rangle\}$. After sorting, objects p_9 and p_2 are output as the most desirable 2 skyline objects.

4.4. Discussion

In this section, we analyze the correctness and complexities of our proposed three algorithms, viz., CB, SB, and RB.

Lemma 4. *The CB, SB, and RB algorithms traverse the index on the data set P two times, one time, and one time, respectively.*

Proof. As shown in Algorithm 1, CB first traverses the R-tree R on the data set P once to obtain the skyline S of P, and then, for each skyline object $s \in S$, it computes $\mu(s)$ and $\tau(s)$ by traversing R once. Thus, the CB algorithm traverses R two times. SB algorithm is based on the sweep line, which identifies skyline objects, and meanwhile calculates their dominating scores and preference scores. RB algorithm employs the reuse technique. Therefore, both SB and RB algorithms traverse the index a single once. □

Theorem 1. *The proposed three algorithms can find exactly the most desirable k skyline objects.*

Proof. CB invokes BBS algorithm [32] to compute the skyline S accurately. Hence, no answer objects are missed (i.e., *no false misses*). Then, every data object is examined in order to calculate $\mu(s)$ and $\tau(s)$ for each skyline object $s \in S$, which ensures *no false hits*. Thus, the correctness of the CB algorithm is guaranteed. The correctness of the SB algorithm is obvious since all data objects are evaluated during the execution, which guarantees no false misses and no false hits. As for the RB algorithm, the aR-tree and Lemma 3 ensure its correctness. □

Let $|R|$ and $|aR|$ be the cardinalities of R-tree and aR-tree indexing a data set P (i.e., the number of total entries including intermediate nodes and data nodes, respectively), m be the number of skyline objects, $|H|$ be the size of a min-heap H, $|H'|$ be the size of a min-heap H', α be the maximal memory space during the execution of ObjectCount function, and β be the percentage of the $|aR|$ that have been visited by RS algorithm.

Theorem 2. *The time complexities of CB, SB, and RB algorithms are* $O((3 \times |R| \times \log|R| + \log|m|) \times m)$, $O((2 \times |R| \times \log|R| + \log|m|) \times m)$, *and* $O(((1 + 2\beta) \times |aR| \times \log|aR| + \log|m|) \times m)$, *respectively.*

Proof. CB algorithm requires $O(|R| \times \log|R| \times m)$ time to compute the skyline S of P using BBS algorithm (line 3 of Algorithm 1). Then, it recursively checks each data object to derive $\mu(s)$ and $\tau(s)$ for each skyline object $s \in S$ (lines 4–18 of Algorithm 1). Every examination takes $O(2 \times \log|R| \times m)$. Thus, the checking in total incurs $O(2 \times |R| \times \log|R| \times m)$ cost. In addition, line 19 of Algorithm 1 can be completed in $O(m \times \log m)$ time. Hence, the time complexity of CB algorithm is $O((3 \times |R| \times \log|R| + \log|m|) \times m)$. SB should evaluate $|P|$ data objects (lines 3–17 of Algorithm 2), and every evaluation can be finished in $O(2 \times \log\ |R| \times m)$ time. Therefore, the evaluations in total incur $O(2 \times |R| \times \log|R| \times m)$ cost. Line 18 of Algorithm 2 takes $O(m \times \log m)$ to perform sorting. Thus, the time complexity of SB algorithm is $O((2 \times |R| \times \log|R| + \log|m|) \times m)$. RB algorithm requires $O(|R| \times \log|R| \times m)$ time to compute the skyline S of P using BBS algorithm (lines 3 of Algorithm 3). It then recursively checks each data object to derive $\mu(s)$ and $\tau(s)$ for every skyline object $s \in S$ by using H (lines 4–17 of Algorithm 3). Each examination takes $O(2 \times \log|aR| \times m)$. Hence, the checking in total incurs $O(2\beta \times |aR| \times \log|aR| \times m)$ cost. In addition, line 18 of Algorithm 3 can be completed in $O(m \times \log m)$ time. Consequently, the time complexity of RB algorithm is $O(((1 + 2\beta) \times |aR| \times \log|aR| + \log|m|) \times m)$. □

Theorem 3. *The space complexities of CB, SB, and RB algorithms are* $O(|H| + |H'| + \alpha)$, $O(|H|)$, *and* $O(|H| + |H'|)$, *respectively.*

Proof. The storage of CB is dominated by heap H' (used in line 3 of Algorithm 1), heap H (utilized in lines 4–18 of Algorithm 1), and the space required to perform the ObjectCount function. Therefore, the space complexity of CB algorithm is $O(|H| + |H'| + \alpha)$. The storage of SB is dominated by heap H. Thus, the space complexity of SB algorithm is $O(|H|)$. The storage of RB is dominated by heap H' (used in line 3 of Algorithm 3) and heap H (utilized in lines 4–17 of Algorithm 3). Hence, the space complexity of RB algorithm is $O(|H| + |H'|)$. □

5. CMDSO query processing

In this section, we extend our techniques (presented above) to handle a natural variant of MDSO queries, namely, the *constrained most desirable skyline object* (CMSDO) query, which aims to compute the MDSO in a specified region. In the sequel, we formalize the CMSDO query in Section 5.1, and then propose three algorithms for computing CMSDO and offer their theoretical analysis accordingly in Section 5.2.

5.1. Problem formulation

In some real-life applications, users might enforce some constraints (e.g., spatial region, distance, etc.) on MDSO queries. As an example, in web-based recommender systems, the users may want to the desired hotel whose room price is between \$200 and \$300. Note that, in a normal MDSO query, no such restriction can be directly specified for the query result. Therefore, the traditional MDSO query cannot tackle such query efficiently. To this end, in this paper, we introduce a new form of MDSO queries, i.e., CMDSO retrieval, which finds the *most preferable k* skyline objects in a given constrained region, as formally defined in Definition 8.

Definition 8 (***Constrained Most Desirable Skyline Object Query***). Given a set P of data objects, an integer k ($\geqslant 1$), a constrained region CR, and let S_c be the *constrained skyline* of P, a *constrained most desirable skyline object* (CMDSO) *query* retrieves a set S_{cr} of skyline objects, such that (i) S_{cr} is inside CR; (ii) S_{cr} contains k skyline objects, i.e., $|S_{cr}| = k$; and (iii) none of object $p' \in S_c - S_{cr}$ inside CR is superior to any answer object $p \in S_{cr}$ according to the ranking criterion (see Definition 6), i.e., $\forall p \in S_{cr},\ p' \in S_c - S_{cr},\ \mu(p) > \mu(p')$ or $\mu(p) = \mu(p') \wedge \tau(p) > \tau(p')$. Note that, when $|S_c| \leqslant k, S_c$ is the result set, i.e., $S_{cr} = S_c$.

The CMDSO query only returns the MDSO in a specified constrained region. For ease of understanding, Fig. 4 depicts an example of the CMDSO query, in which the shaded area represents the constrained region CR. As shown in Fig. 4, objects p_9 and p_6 are returned as the most desirable 2 skyline objects. The skyline object p_1 is located outside CR. Thus, it cannot be the final answer objects. Since both p_9 and p_6 dominate *three* points, while p_2 only dominates *two* points, p_2 is not the final answer object based on the ranking criterion. The CMDSO query has a large application base. Take the web-based recommender system again. The user may also want to the desired hotel whose distance to the beach is no more than one kilometer.

5.2. Algorithms for finding CMDSO

In this subsection, we present three efficient algorithms, namely, *Constrained Cell Based algorithm* (CCB), *Constrained Sweep Based algorithm* (CSB), and *Constrained Reuse Based algorithm* (CRB), for finding the CMDSO, and then analyze their correctness and complexities, respectively.

The CMDSO query is a natural extension of the MDSO query. Therefore, the algorithms designed for MDSO retrieval can also be extended to handle the CMDSO query. The only difference with respect to the aforementioned MDSO query processing algorithms (i.e., CB, SB, and RB) is that the algorithms for the CMDSO query have to consider the constrained region.

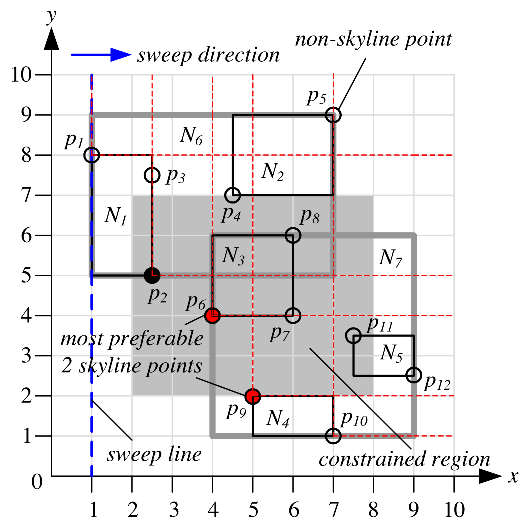


Fig. 4. Example of a MDSO query.

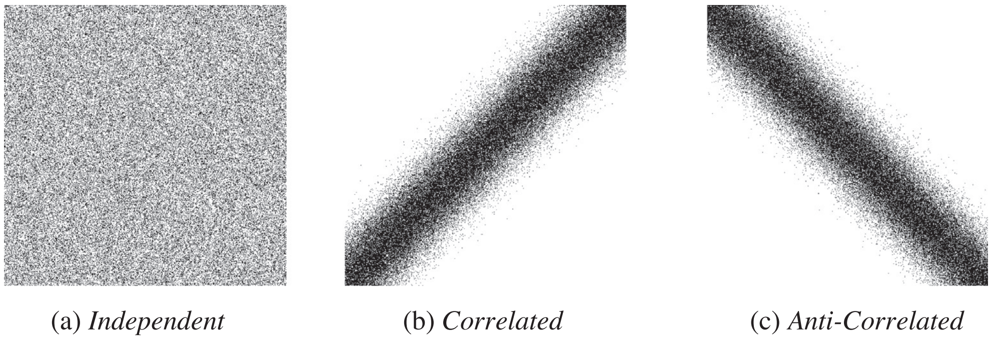


Fig. 5. Illustration of synthetic dataset distributions.

Specifically, there are three approaches to extend the traditional MDSO query algorithms to develop the algorithms for CMDSO query algorithms. (i) We first call the traditional MDSO query algorithms to get the *whole* most desirable skyline objects (i.e. the entire skyline with computed dominating scores and preference scores) without considering the constrained region. Then, we obtain the k skyline objects that are within the constrained region according to the ranking criterion. (ii) We first index the objects inside a specified constrained region in a single R-tree or aR-tree, and then, we perform the MDSO query on the constructed R-tree or aR-tree to get the final result. (iii) We integrate the constrained region examination into the conventional MDSO query processing algorithms, i.e., the entries not intersecting the constraint region are pruned away. For the first method, its final result may be not correct. This is because the CMDSO query only considers the data objects in the constrained region. If we firstly get the skyline of the whole dataset, it means that we expand the qualified datasets, i.e., the datasets fall into the constrained region. It will lead to enlarge some skyline objects' dominating scores and preference scores. Therefore, the final result is not correct. For instance, in Fig. 4, if we adopt the first approach, we firstly get the skyline set $S = \{\langle p_9, 5, 61/30\rangle, \langle p_2, 4, 61/30\rangle, \langle p_{10}, 3, 6/5\rangle, \langle p_6, 4, 23/15\rangle, \langle p_1, 1, 1/5\rangle\}$. Since p_1 and p_{10} are out of the constrained region, these two points are discarded. For the rest of the objects, according to their values of dominating scores and preference scores, the objects with p_9 and p_2 are returned as the final result. However, the correct result should be p_9 and p_6 due to $\mu(p_9) = 2,\ \mu(p_6) = 3$, and $\mu(p_9) = 3$. Consequently, the first method is *infeasible*. As for the second approach, it needs to create another index. Once the constrained region is changed, the index needs to be rebuilt. Thus, the second method is not efficient, and can be used as an alternative. For the third method, it integrates the constrained region checking into the traditional MDSO query processing, which can guarantee the correctness of the final result. In addition, it uses the index containing the entire dataset, which is not affected by the change of the constrained region. In summary, the third approach is superior to other two methods in both correctness and effectiveness. Therefore, in this paper, we adopt the third method to adapt the MDSO query algorithms in order to form the CMDSO query algorithms. Before presenting our algorithms, we present the following lemma.

Lemma 5. *Given a set P of data objects organized by an index (R-tree or aR-tree) and a constrained region CR, if an R-tree (or aR-tree) node does not cross CR, the node cannot contain any constrained most desirable skyline object.*

Proof. The proof is straightforward, and thus omitted. □

Lemma 5 ensures that the entries located outside the constrained region can be safely pruned because it does not contribute to the final result. Next, we propose our algorithms in details.

Our first algorithm, namely, *Constrained Cell Based algorithm* (CCB), is based on the CB algorithm. The basic idea of CCB is to compute the constrained skyline *CS* using BBS algorithm, and then, for each skyline object $cs \in CS$, calculate its dominating score $\mu(cs)$ and preference score $\mu(cs)$ respectively, and finally return the top-k skyline objects in *CS* according to our proposed ranking criterion (see Definition 6). It is worth noting that, when computing the dominating score $\mu(cs)$ and preference score $\tau(cs)$, we only consider the points in the constrained region. Since the algorithm of CCB is similar to CB algorithms, the pseudo-code of CCB is ignored. Here we only discuss the differences between CCB and CB algorithms. The first difference is that in line 3 of Algorithm 1, the CCB algorithm should "compute constrained skyline *CS* using BBS algorithm with H'" rather than "compute skyline *S* using BBS algorithm with H'". For the second difference, when CCB gets a non-skyline object, the algorithm needs to determine whether it is inside the constrained region (line 6 of Algorithm 1). If *no*, it can be discarded. When encountering an intermediate node, except to examine its location with respect to the *cell* (line 11 of

Table 2
Parameter statistics.

Parameter	Range	Default
k	10, 20, 30, 40, 50	30
Dimensionality	2, 3, 4, 5	3
Cardinality	250 K, 500 K, 1000 K, 2000 K, 4000 K	4
Constrained region (% of full space)	15, 30, 45, 60, 75	100

Table 3
Top-14 most desirable skyline records on the *NBA* data set.

NBA Player/Year	GP	PTS	REB	AST	$\mu(s)$	$\tau(s)$
Wilt Chamberlain/1967	82	1982	1952	702	14962	131
Kareem Abdul-Jabbar/1975	82	2275	1383	413	14392	121
Larry Bird/1985	82	2115	805	557	14286	131
John Havlicek/1971	82	2252	762	614	13862	131
Kareem Abdul-Jabbar/1969	82	2361	1190	337	13828	114
Wilt Chamberlain/1966	82	1956	1957	630	13457	131
Kareem Abdul-Jabbar/1970	82	2596	1311	272	13116	110
Michael Jordan/1989	82	2753	565	519	13078	131
Gary Payton/1999	82	1982	529	732	13035	98
Kareem Abdul-Jabbar/1971	81	2822	1346	370	12936	115
John Havlicek/1970	81	2338	730	607	12934	131
Alex English/1982	82	2326	601	397	12847	119
Michael Jordan/1988	81	2633	652	650	12692	131
Oscar Robertson/1962	80	2264	835	758	12618	111

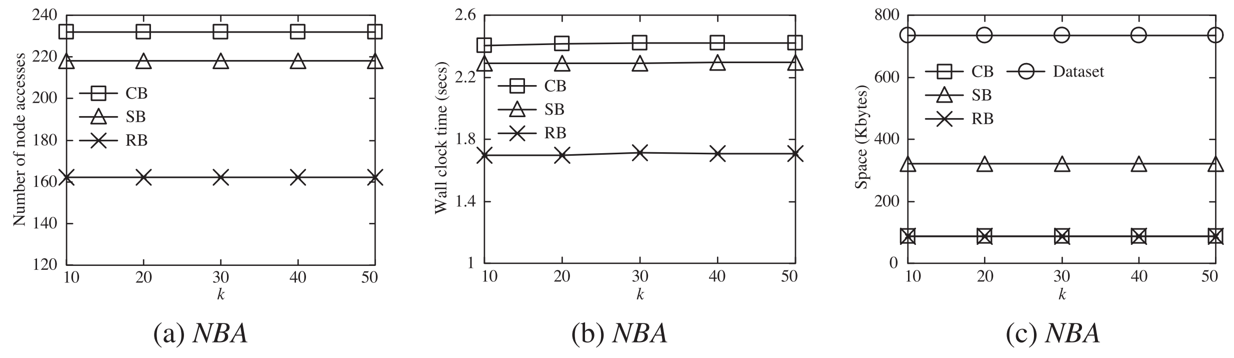


Fig. 6. MDSO query cost vs. varying k (dimensionality = 4).

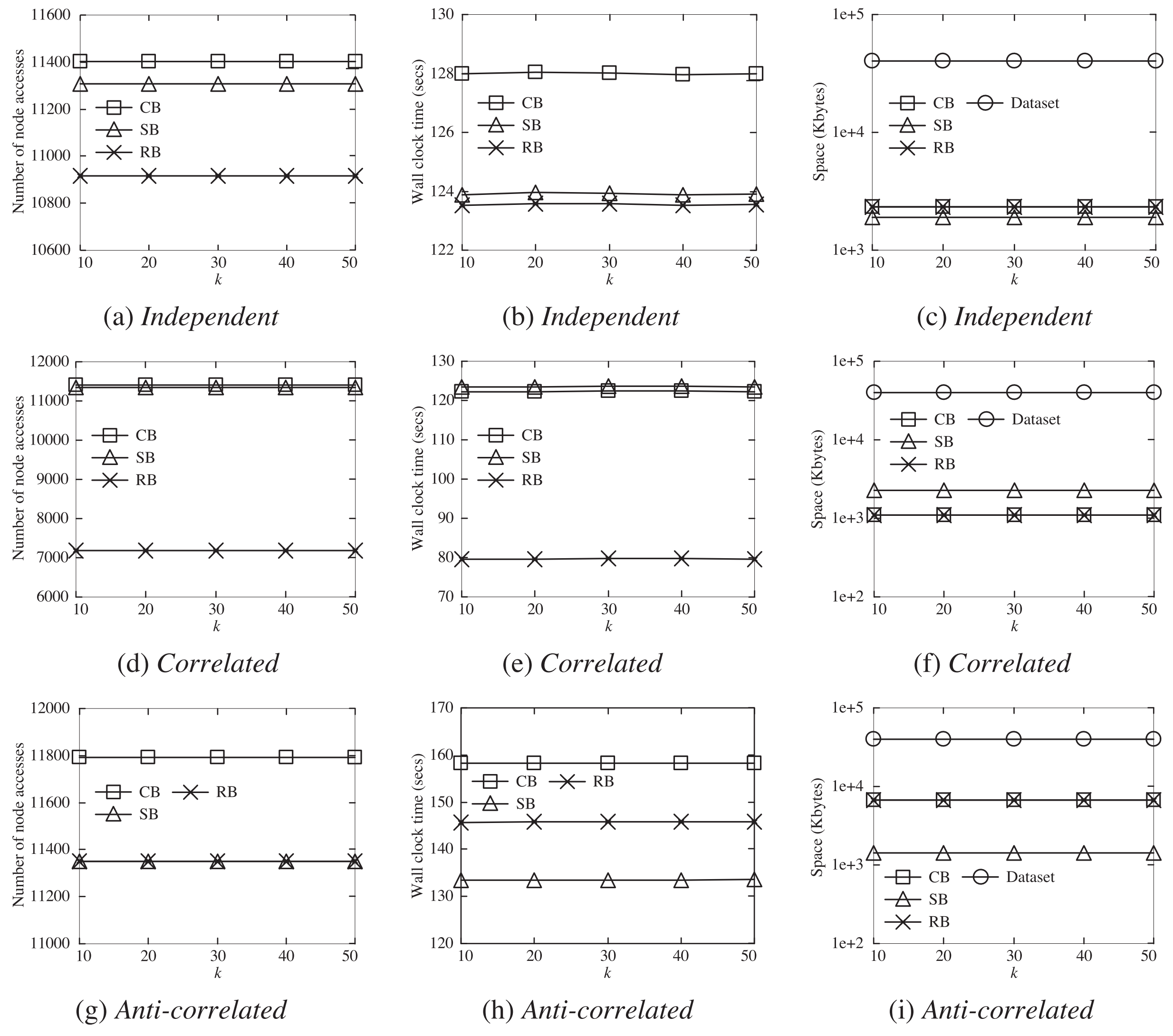


Fig. 7. MDSO query cost vs. varying k (dimensionality = 3, cardinality = 1000 K).

Algorithm 1), CCB also requires the constrained region checking. If the intermediate node needs to be expanded, CCB also examines whether or not its children are within the constrained region. Anyway, any entry needs constrained region examination before it is inserted into H and after it is popped from H.

Since the CCB algorithm needs to traverse the index twice for getting the final result, we propose another more efficient algorithm, i.e., *Constrained Sweep Based algorithm* (CSB), which only traverses the index a single once to retrieve the result. CSB algorithm is adapted from the SB algorithm by employing the third method mentioned previously. The main idea of CSB is to identify skyline objects in a specified constrained region and meanwhile calculate their dominating scores and preference scores, via a *single* traversal of the R-tree R used to organize a data set P. Compared with SB algorithm, there are two differences between CSB and SB algorithms. The first one is that, when evaluating a data object (line 5 of Algorithm 2), CSB has to determine whether it is inside the constrained region or not. The other is that, in line 16 of

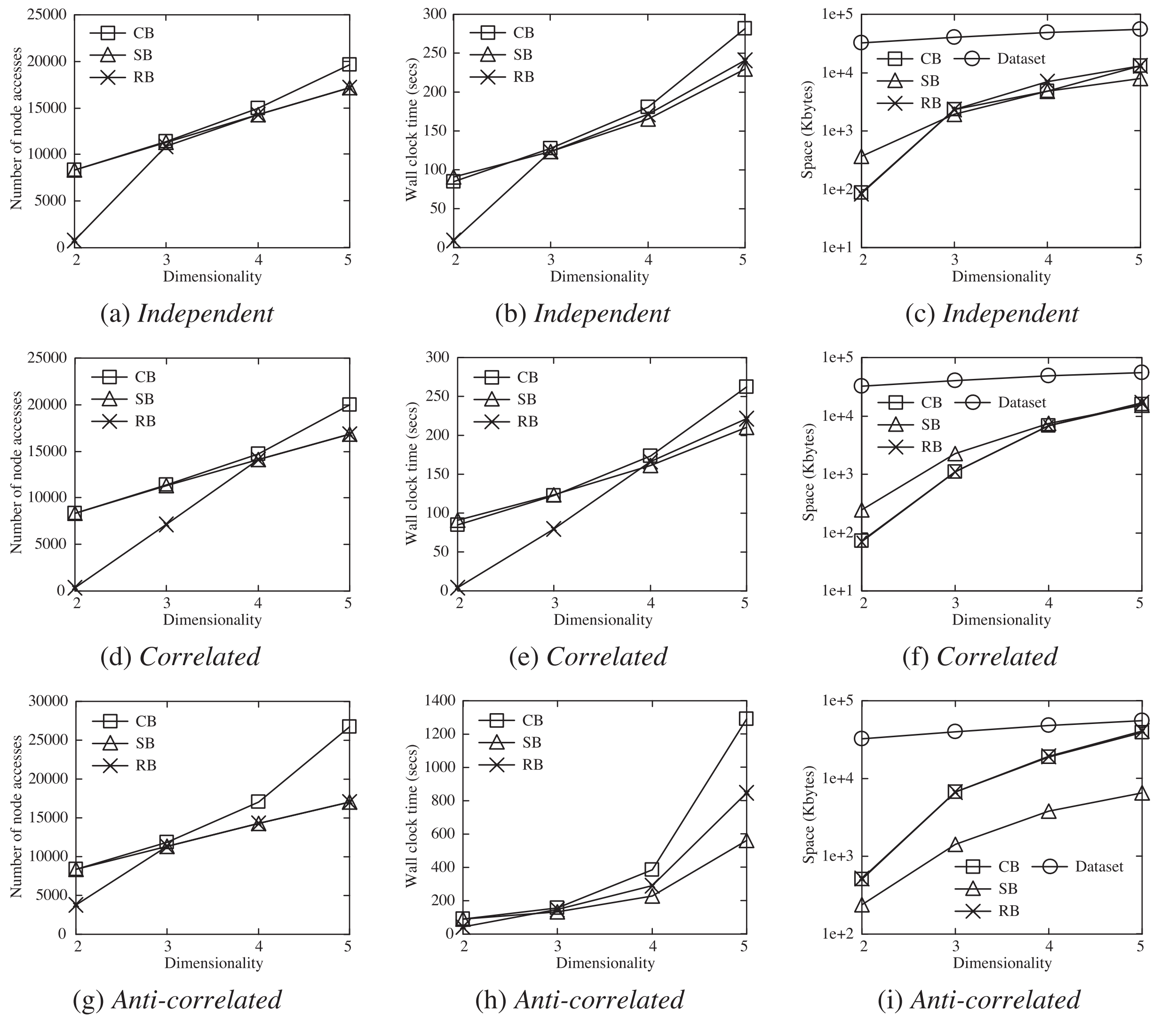


Fig. 8. MDSO query cost vs. varying dimensionality ($k = 30$, cardinality = 1000 K).

Algorithm 2, when expanding an intermediate node, CSB needs to examine whether or not its children entries intersect or falls into the constrained region.

Our third algorithm, i.e., *Constrained Reuse Based Algorithm* (CRB), is the customization of the RS algorithm. CRB also uses the aR-tree as the index and employs the reuse technique. The basic idea of CRB is, like CCB, to compute the constrained skyline *CS* using BBS algorithm, but unlike CCB, during this computation, all the data objects and node MBRs, which are dominated by some skyline objects in *CS* and within the constrained region, need to be kept. Then, for each skyline object $cs \in CS$, CRB calculates its dominating score $\mu(s)$ and preference score $\tau(s)$ respectively, using the visited nodes stored in the previous step. Finally, CRB returns the top-k skyline objects in *CS* according to our proposed ranking criterion. The pseudo-code of CRB is similar as that of RB except that, in Algorithm 3, lines 3, 6, 11, and 16 require constrained region checking, and thus skipped.

Next, we analyze the correctness of the algorithms for CMDSO queries.

Lemma 6. *The CCB algorithm traverses the index on the data set P two times, and both CSB and CRB algorithms traverse the index over P a single once.*

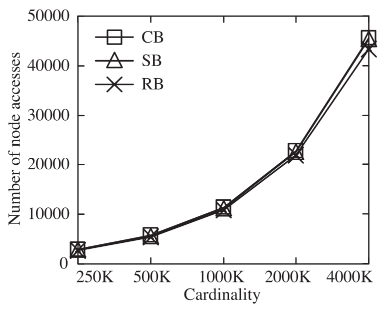
Proof. It is evident that CCB first traverses the aR-tree *aR* to obtain the constrained skyline *CS*, and then, for each skyline objects $cs \in CS$, it computes $\mu(s)$ and $\tau(s)$ by traversing *aR* once. Thus, the CCB algorithm traverses *aR* two times. CSB algorithm is based on the sweep line, which identifies constrained skyline objects and calculates their dominating scores and preference scores simultaneously. CRB algorithm utilizes the reuse technique. Therefore, both CSB and CRB algorithms traverse the index only once. □

Lemma 7. *All the entries in the heap H cross the specified constrained region.*

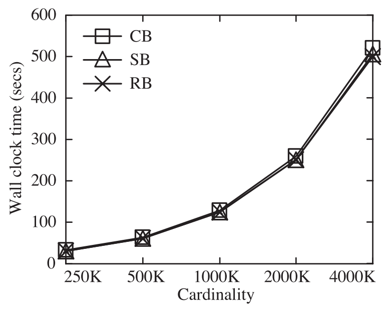
Proof. The proof is obvious because, in our algorithms, when a node entry is expanded, we only en-heap those child entries that are located inside or intersect the constrained region *CR*. □

Theorem 4. *All the three presented algorithms can find exactly the constrained most desirable k skyline objects.*

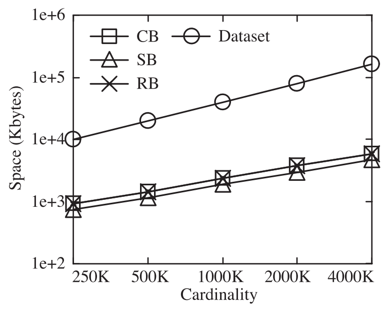
Proof. It is guaranteed by Theorem 1 and Lemma 5. □



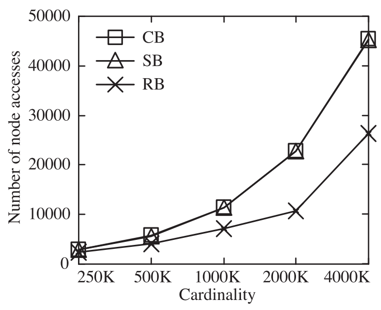
(a) *Independent*



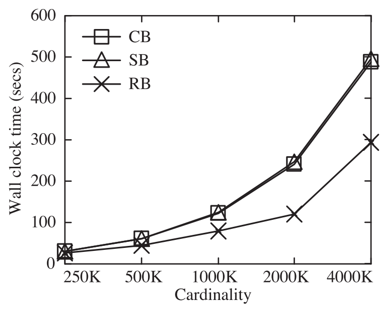
(b) *Independent*



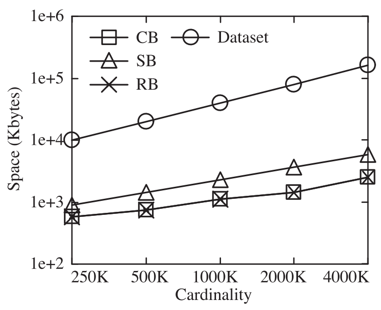
(c) *Independent*



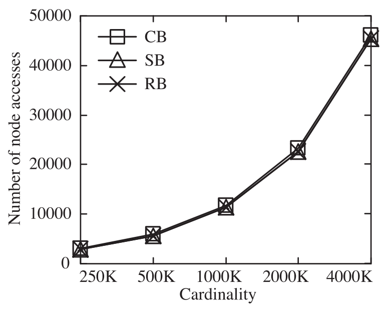
(d) *Correlated*



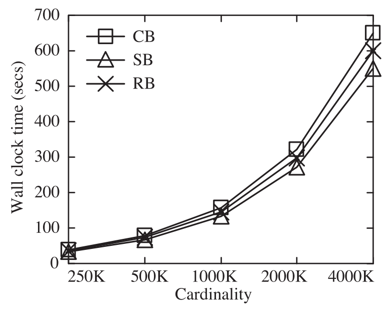
(e) *Correlated*



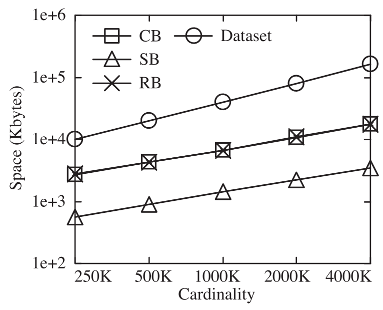
(f) *Correlated*



(g) *Anti-correlated*



(h) *Anti-correlated*



(i) *Anti-correlated*

Fig. 9. MDSO query cost vs. varying cardinality ($k = 30$, dimensionality = 3).

6. Experimental evaluation

This section experimentally evaluates the effectiveness of our devised ranking criterion and the performance of our proposed algorithms in terms of both efficiency and scalability. In what follows, we first describe the experimental settings in Section 6.1, and then we show the effectiveness of ranking criterion in Section 6.2. Considerable experimental results and our findings for MDSO and CMDSO queries are reported in Sections 6.3 and 6.4, respectively.

6.1. Experimental setup

We use both real *NBA* data and synthetic datasets. Specifically, the *NBA* data set records the NBA players' technical statistics from 1946 to 2004. It is available at the NBA official website (www.databasebasketball.com), and frequently adopted in the skyline literature [6,7,14,20,21,26]. *NBA* contains 15,280 records about 3542 players on 17 attributes (e.g., number of steals, number of blocks, and number of fouls, etc.) from regular seasons. Each record provides statistics of a player in a season. Consequently, a player may have several records if he played in NBA for more than one season. We selected the four attributes, namely, number of games played (GP), total points (PTS), total rebounds (REB), and total assists (AST), in our experiments. According to the semantics of this data set, the larger the attribute values are, the better the player is. Hence, player *A* dominates player *B* if *A* is not smaller than *B* on all attribute values and greater than *B* on at least one attribute value. In addition to the *NBA* data set, we also created *Independent*, *Correlated*, and *Anti-correlated* datasets with dimensionality varied from 2 to 5 and cardinality in the range [250 K, 4000 K]. Our generation follows exactly the description in [5]. In *Independent*, all attribute values are independent and uniformly distributed; the *Correlated* dataset represents an environment where points which are good in one dimension are also good in the other dimensions; in *Anti-correlated*, all attribute values are anti-correlated, meaning that if an object has a small value on one attribute, it tends to have a large value on at least another attribute. Fig. 5 shows the three distributions of the dataset with 100000 2D points.

Every dataset is indexed by an R*-tree [4] or an aR-tree [31] with a disk page size of 4096 bytes. We verify the effectiveness of our presented ranking criterion, and evaluate the efficiency and scalability of our proposed algorithms under several factors, including k, dimensionality, cardinality, and constrained region. Note that, in each experiment, only one factor varies, whereas the others are fixed to their default values. The range of the parameters and their default values are listed in Table 2. The number of

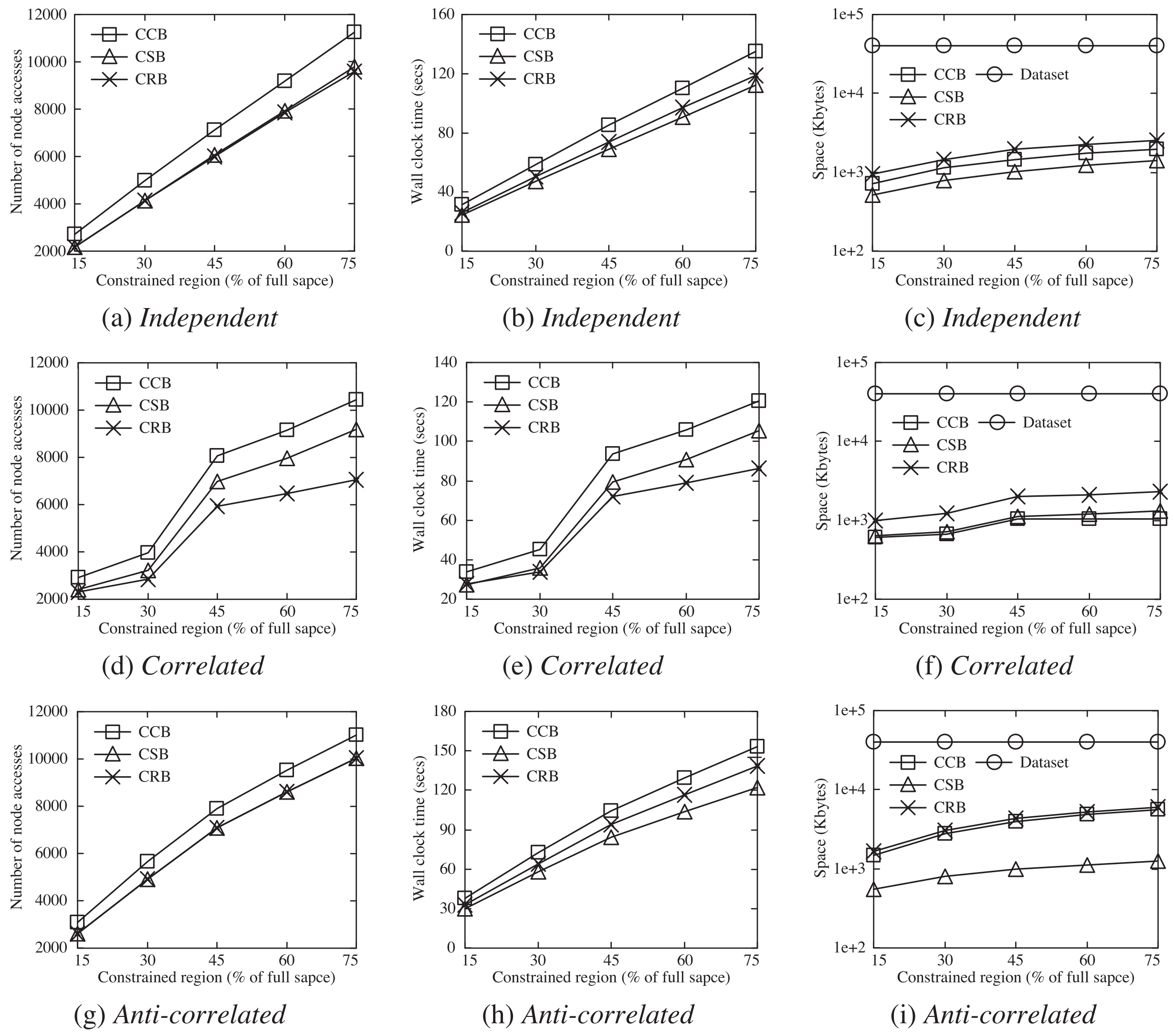


Fig. 10. CMDSO query cost vs. constrained region ($k = 30$, dimensionality = 3, cardinality = 1000 K).

node/page accesses (i.e., I/O cost), wall clock time (i.e., the sum of I/O cost and CPU time, where the I/O cost is computed by charging 10 ms for each page access, as with [28]), and the maximal space consumption are employed as the major performance metrics. Recall that the MDSO query takes a different perspective on the problem of controlling the size of skyline object set, and hence, we do not compare it with other methods, which return the fixed size of skyline, in our experiments.

All algorithms were implemented in C++, and all experiments were conducted on the PC with an Intel Core 2 Duo 2.13 GHz CPU and 2 GB RAM, running Microsoft Windows XP Professional Edition.

6.2. Effectiveness of ranking criterion

This set of experiments aims at evaluating the effectiveness of our proposed ranking criterion. Table 3 shows the top-14 most preferable skyline records on the *NBA* data set, together with their corresponding dominating scores and preference scores listed in the sixth and seventh columns, respectively. Note that, in this *NBA* data set, we identify 50 skyline records in the full 4-dimensional spaces, since it is fairly *correlated* as pointed out in [2,6,7]. It is observed that, we successfully find the superstars in NBA's history, such as Wilt Chamberlain, Kareem Abdul-Jabbar, and Michael Jordan. Most of readers who follow the basketball will agree that this is a reasonable set of great NBA players of all time. Those records not only have good attribute values so that they are in the skyline, but also dominate many other non-skyline records such that they have high dominating scores. Consequently, our devised ranking criterion is reasonable. In addition, compared with the top-14 players in NBA history selected by http://www.nba.com, the accuracy of MDSO is 71.4%, which can also demonstrate the quality of MDSO query results.

6.3. Results on MDSO queries

In this subsection, we present the experimental results on MDSO queries with respect to k, dimensionality, and cardinality, respectively.

The first set of experiments evaluates the effect of the number k of required skyline objects on the efficiency of the algorithms. Figs. 6 and 7 depict the results on *NBA* dataset and synthetic datasets, respectively. Specifically, each diagram plots the number of node accesses, wall clock time (in seconds), and space (in Kbytes), respectively, with respect to k for 4D *NBA* dataset and 3D synthetic datasets with cardinality = 1000 K. From the number of node accesses and the wall clock shown in the figures, we can observe that RB outperforms SB, and both them are better than CB. The reason behind is that CB requires traversing the R-tree R two times, which incurs a large number of redundant node accesses and high wall clock time cost. Hence, CB is the worst for all cases. As discussed in Section 4.4, both RB and SB traverse the index only once. Moreover, RB also employs the aR-tree (instead of the R-tree) to further reduce the number of node accesses. Therefore, RB exceeds SB. Notice that, there is an exception for the*Anti-correlated* dataset. It may caused by its data distribution. As shown in Figs. 6(c) and 7(c), (f), and (i), the space requirement of CB, SB, and RB is significantly less than the original dataset size. Note that, the space requirement of CB and RB are the same. This is because the space required to execute the *ObjectCount* function is very small, and thus can be ignored. It is observed that, the performance of CB, SB, and RB is not sensitive to different k, since they obtain the final query result from all ordered skyline objects no matter how large k is.

Next, we investigate the impact of the dimensionality on the efficiency of the algorithms, by fixing $k = 30$, cardinality = 1000 K, and changing dimensionality between 2 and 5. Fig. 8 illustrates the experimental results on synthetic datasets. As expected, the performance of CB and SB degrades with the growth of dimensionality. This generation is due to the growth of the skyline size and the poor performance of R-trees in high dimensions. SB and RB are better than CB at the most of cases, and their difference increases with dimensionality. A crucial observation is that, for CPU time and space, CB is comparable to SB in a 2D space. This is owing to the super-efficient BBS algorithm and numerous R-tree nodes located inside a single cell.

Finally, we study the influence of the dataset cardinality on the performance of the algorithms. Towards this, we fix k at 30 and use 3-dimensional synthetic datasets whose cardinality varies from 250 K to 4000 K. Fig. 9 shows the performance of CB, SB, and RB as a function of cardinality. Again, RB and SB outperform CB in all cases. Furthermore, the cost of CB, SB, and RB ascends as cardinality grows. This is because the size of skyline increases with cardinality.

To summarize, from the above experimental results on both real and synthetic datasets, we can conclude that: (i) the I/O overhead and the wall clock time of RB and SB are always *less* than those of CB, and RB is *better* than SB in most cases; (ii) the I/O overhead and the wall clock time of CB is *comparable* to those of SB in a 2D space, while SB significantly outperforms CB in high dimensions; (iii) the space requirement of CB, SB, and RB is *negligible* compared to the dataset size; and (iv) the space requirement of CB and RB are almost the same.

6.4. Results on CMDSO queries

In the third set of experiments, we verify the performance of the algorithms for CMDSO queries. It is worth mentioning that (i) the ratio of the constrained region is the percentage of the volume of the data universe; (ii) each value reported in the following diagrams is the average of 100 queries; and (iii) the locations of constrained regions were uniformly generated [32].

Having confirmed the efficiency of CB, SB and RB for conventional MDSO retrieval in Section 6.3 and the algorithms for CMDSO queries are adapted from traditional MDSO query algorithms, thus, one parameter is employed in this part's experiments, i.e., the constrained region, which varies from 15 to 75 (% of full space). Fig. 10 plots the number of node accesses, wall clock time, and space respectively, with respect to the constrained region for 3D synthetic datasets with $k = 3$ and cardinality = 1000 K. Once more, from the figure, we can observe that both CSB and CRB are better than CCB in terms of the number of node accesses and wall clock time. The number of node accesses of CSB and CRB are almost the same for *Independent* and *Anti-correlated* datasets, but the wall clock time of CRB is better than that of CSB because CSB takes more CPU time. In addition, with the growth of the constrained region, all the three performance metrics increase. The reason is that, as the constrained region expands, it contains more objects, and thus, more objects need to be checked, which results in higher I/O, wall clock time, and space.

To sum up, from the experimental results above, we can conclude that: (i) CRB and CSB are always better than CCB in terms of the I/O overhead and the wall clock time; (ii) CRB and CSB are comparable with respect to wall clock time; and (iii) the space requirement of our proposed algorithms is much smaller than the original dataset.

7. Conclusions

The skyline of a dataset might have an overwhelming number of skyline objects. Returning all of them may make it difficult for a user to make a good, quick selection. In this paper, we introduce a new operator, namely, the most desirable skyline object (MDSO) query, for finding manageable size of the most preferable/interesting skyline objects. First, we formalize the ranking criterion and the MDSO query, respectively. Then, three algorithms, i.e., CB, SB, and RB, are developed for efficiently processing MDSO queries. As a second step, we propose a variant of MDSO queries, i.e., the constrained most desirable skyline object (CMDSO) query, and extend our techniques to tackle it efficiently. Finally, extensive experimental evaluation on both real and synthetic datasets demonstrates that our presented ranking criterion is reasonable, and our proposed algorithms are efficient and scalable.

The work reported in this paper presents our first step with respect to the MDSO operator. In the future, we plan to develop more efficient algorithms for answering MDSO queries by using preprocessing. In addition, we also intend to investigate MDSO queries on arbitrary subspace, uncertain data, and so forth.

Acknowledgments

This work was supported in part by the National Basic Research Program of China (973 Program) No. 2015CB352502 and 2015CB352503, National Natural Science Foundation of China (NSFC Grants) No. 61379033 and 61472348, and the Fundamental Research Funds for the Central Universities.

References

[1] W.-T. Balke, U. Guntzer, C. Lofi, Eliciting matters – controlling skyline sizes by incremental integration of user preferences, in: Proc. Int. Conf. Database Systems for Advanced Applications (DASFAA), 2007, pp. 551–562.
[2] I. Bartolini, P. Ciaccia, M. Patella, Efficient sort-based skyline evaluation, ACM Trans. Database Syst. 33 (4) (2008) 1–45.
[3] I. Bartolini, Z. Zhang, D. Papadias, Collaborative filtering with personalized skylines, IEEE Trans. Knowl. Data Eng. 23 (2) (2011) 190–203.
[4] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in: Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD), 1990, pp. 322–331.
[5] S. Borzsony, D. Kossmann, K. Stocker, The skyline operator, in: Proc. Int. Conf. Data Engineering (ICDE), 2001, pp. 421–430.
[6] C.-Y. Chan, H.V. Jagadish, K.-L. Tan, A.K.H. Tung, Z. Zhang, Finding k-dominant skylines in high dimensional space, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2006, pp. 503–514.
[7] C.-Y. Chan, H.V. Jagadish, K.-L. Tan, A.K.H. Tung, Z. Zhang, On high dimensional skylines, in: Proc. Int. Conf. Extending Database Technology (EDBT), 2006, pp. 478–495.
[8] L. Chen, B. Cui, H. Lu, Constrained skyline query processing against distributed data sites, IEEE Trans. Knowl. Data Eng. 23 (2) (2011) 204–217.
[9] L. Chen, X. Lian, Efficient processing of metric skyline queries, IEEE Trans. Knowl. Data Eng. 21 (3) (2008) 351–365.
[10] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with presorting, in: Proc. Int. Conf. on Data Engineering (ICDE), 2003, pp. 717–719.
[11] A. Das Sarma, A. Lall, D. Nanongkai, R.J. Lipton, J. Xu, Representative skylines using threshold-based preference distributions, in: Proc. Int. Conf. on Data Engineering (ICDE), 2011, pp. 387–398.
[12] E. Dellis, B. Seeger, Efficient computation of reverse skyline queries, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2007, pp. 291–302.

[13] D. Fuhry, R. Jin, D. Zhang, Efficient skyline computation in metric space, in: Proc. Int. Conf. Extending Database Technology (EDBT), 2009, pp. 1042–1051.

[14] Y. Gao, J. Hu, G. Chen, C. Chen, Finding the most desirable skyline objects, in: Proc. Int. Conf. Database Systems for Advanced Applications (DASFAA), 2010, pp. 116–122.

[15] Y. Gao, Q. Liu, B. Zheng, G. Chen, On efficient reverse skyline query processing, Expert Syst. Appl. 41 (7) (2014) 3237–3249.

[16] Y. Gao, X. Miao, H. Cui, G. Chen, Q. Li, Processing *k*-skyband, constrained skyline, and group-by skyline queries on incomplete data, Expert Syst. Appl. 41 (10) (2014) 4959–4974.

[17] P. Godfrey, R. Shipley, J. Gryz, Maximal vector computation in large data sets, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2005, pp. 229–240.

[18] K. Hose, A. Vlachou, A survey of skyline processing in highly distributed environments, VLDB J. 21 (3) (2012) 359–384.

[19] Z. Huang, H. Lu, B.C. Ooi, A.K.H. Tung, Continuous skyline queries for moving objects, IEEE Trans. Knowl. Data Eng. 18 (12) (2006) 1645–1658.

[20] V. Koltun, C.H. Papadimitriou, Approximately dominating representatives, in: Proc. Int. Conf. Database Theory (ICDT), 2005, pp. 204–214.

[21] D. Kossmann, F. Ramsak, S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2002, pp. 275–286.

[22] H.T. Kung, F. Luccio, F.P. Preparata, On finding the maxima of a set of vectors, J. ACM 22 (4) (1975) 469–476.

[23] M.-W. Lee, S.-W. Hwang, Continuous skylining on volatile moving data, in: Proc. Int. Conf. on Data Engineering (ICDE), 2009, pp. 1568–1575.

[24] J. Lee, S. Hwang, Scalable skyline computation using a balanced pivot selection technique, Inf. Syst. 39 (2014) 1–21.

[25] J. Lee, S. Hwang, Toward efficient multidimensional subspace skyline computation, VLDB J. 23 (1) (2014) 129–145.

[26] J. Lee, G.W. You, S. Hwang, Personalized top-*k* skyline queries in high-dimensional space, Inf. Sci. 34 (1) (2009) 45–61.

[27] K.C.K. Lee, B. Zheng, H. Li, W.C. Lee, Approaching the skyline in z order, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2007, pp. 279–290.

[28] X. Lian, L. Chen, Reverse skyline search in uncertain databases, ACM Trans. Database Syst. 35 (1) (2010) 3.

[29] X. Lin, Y. Yuan, W. Wang, H. Lu, Stabbing the sky: efficient skyline computation over sliding windows, in: Proc. Int. Conf. Data Engineering (ICDE), 2005, pp. 502–513.

[30] X. Lin, Y. Yuan, Q. Zhang, Y. Zhang, Selecting stars: the *k* most representative skyline operator, in: Proc. Int. Conf. on Data Engineering (ICDE), 2007, pp. 86–95.

[31] D Papadias, P Kalnis, J Zhang, Y Tao, Efficient OLAP operations in spatial data warehouses, in: Proc. Int. Symposium Spatial and Temporal Databases (SSTD), 2001, pp. 443–459.

[32] D. Papadias, Y. Tao, G. Fu, B. Seeger, Progressive skyline computation in database systems, ACM Trans. Database Syst. 30 (1) (2005) 41–82.

[33] J. Pei, B. Jiang, X. Lin, Y. Yuan, Probabilistic skylines on uncertain data, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2007, pp. 15–26.

[34] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, Towards multidimensional subspace skyline analysis, ACM Trans. Database Syst. 31 (4) (2006) 1335–1381.

[35] N. Sarkas, G. Das, N. Koudas, A.K.H. Tung, Categorical skylines for streaming data, in: Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD), 2008, pp. 239–250.

[36] K.L. Tan, P.K. Eng, B.C. Ooi, Efficient progressive skyline computation, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2001, pp. 301–310.

[37] Y. Tao, L. Ding, X. Lin, J. Pei, Distance-based representative skyline, in: Proc. Int. Conf. on Data Engineering (ICDE), 2009, pp. 892–903.

[38] Y. Tao, D. Papadias, Maintaining sliding window skylines data streams, IEEE Trans. Knowl. Data Eng. 18 (3) (2006) 377–391.

[39] Y. Tao, X. Xiao, J. Pei, Subsky: efficient computation of skylines in subspaces, in: Proc. Int. Conf. on Data Engineering (ICDE), 2006, pp. 65–65.

[40] G. Trimponias, I. Bartolini, D. Papadias, Y. Yang, Skyline processing on distributed vertical decompositions, IEEE Trans. Knowl. Data Eng. 25 (4) (2013) 850–862.

[41] A. Vlachou, C. Doulkeridis, M. Halkidi, Discovering representative skyline points over distributed data, in: Proc. Int. Conf. Scientific and Statistical Database Management (SSDBM), 2012, pp. 141–158.

[42] A. Vlachou, M. Vazirgiannis, Ranking the sky: discovering the importance of skyline points through subspace dominance relationships, Data Knowl. Eng. (DKE) 69 (9) (2010) 943–964.

[43] T. Xia, D. Zhang, Y. Tao, On skylining with flexible dominance relation, in: Proc. Int. Conf. on Data Engineering (ICDE), 2008, pp. 1397–1399.

[44] M.L. Yiu, N. Mamoulis, Efficient processing of top-*k* dominating queries on multi-dimensional data, in: Proc. Int. Conf. Very Large Data Base (VLDB), 2007, pp. 483–494.

[45] Y. Zhan, W. Zhan, X. Lin, B. Jiang, J. Pei, Ranking uncertain sky: the probabilistic top-*k* skyline operator, Inf. Sci. 36 (5) (2011) 898–915.

[46] Z.J. Zhang, X.Y. Guo, H. Lu, A.K.H. Tung, N. Wang, Discovering strong skyline points in high dimensional spaces, in: Proc. ACM Int. Conf. Information and Knowledge Management (CIKM), 2005, pp. 247–248.

[47] W. Zhang, X. Lin, Y. Zhang, W. Wang, J.X. Yu, Probabilistic skyline operator over sliding windows, in: Proc. Int. Conf. Data Engineering (ICDE), 2009, pp. 1060–1071.

[48] S. Zhang, N. Mamoulis, D.W. Cheung, Scalable skyline computation using object-based space partitioning, in: Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD), 2009, pp. 483–494.