2019 IEEE 35th International Conference on Data Engineering (ICDE)

An MBR-Oriented Approach for Efficient Skyline Query Processing

Ji Zhang[†], Wenlu Wang[†], Xunfei Jiang[‡], Wei-Shinn Ku[†], Hua Lu[§]
[†]*Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA*
[‡]*Department of Computer Science, Earlham College, Richmond, IN, USA*
[§]*Department of Computer Science, Aalborg University, Aalborg, Denmark*
{jizhang, wenluwang, weishinn}@auburn.edu, jiangxu@earlham.edu, luhua@cs.aau.dk

***Abstract*—This research proposes an advanced approach that improves the efficiency of skyline query processing by significantly reducing the computational cost on object comparisons, i.e., dominance tests between objects. Our solutions are based on two novel concepts. The *skyline query over Minimum Bounding Rectangles (MBRs)* receives a set of MBRs and returns the MBRs that are not dominated by other MBRs. In the dominance test for MBRs, the detailed attribute values of objects in the MBRs are not accessed. Moreover, the *dependent group of MBRs* reduces the search space for dominance tests. Objects in an MBR are only compared with the ones in the corresponding dependent groups of the MBR rather than with the entire dataset. Our solutions apply the two concepts to the R-tree in order to use its hierarchical structure in which every node is a natural abstraction of an MBR.**

Specifically, given the R-tree index of an input dataset, we first eliminate unqualified objects by utilizing the skyline query over MBRs (i.e., intermediate nodes in the R-tree). Subsequently, we generate dependent groups for the skyline MBRs. Two dependent group generation methods that rely on either the sorting technique or the R-tree index are developed. Further, we apply an existing skyline algorithm to every dependent group, and the results of the original skyline query are the union of skyline objects in the dependent groups. In addition, we also analyze the cardinality of the two new concepts based on a probabilistic model, which enables us to analyze the computational complexity of the proposed solutions. Our experimental results show that the proposed solutions are clearly more efficient than the state-of-the-art approaches.

I. Introduction

Given a set of objects in a d-dimensional space, the skyline query returns objects that are not dominated by any other objects in the object set [2]. In a d-dimensional space $\mathbb{R}^d$, an object q can be represented as $q = \{x^1, x^2, ..., x^d\}$ where $q.x^i$ is the attribute value of q on the i^{th} dimension. Without loss of generality, in this paper we assume that the minimum values are preferred in all dimensions. Formally, the dominance among objects and the skyline query are defined as follows.

Definition 1: (*Object Domination*) Given two objects q, q' $\in Q$, q dominates q', denoted by $q \prec q'$, if $\forall\ i \in \{1, ..., d\}$, $q.x^i \le q'.x^i$ and $\exists\ j \in \{1, ..., d\},\ q.x^j < q'.x^j$.

Definition 2: (*Skyline Query over Objects*) The skyline of an object set Q in a d-dimensional data space, denoted by $SKY(Q)$, is a complete subset of objects that are not dominated by any other objects in Q.

$$SKY(Q) = \{q\ \in Q \mid \nexists\ q' \in Q,\ q' \prec q\} \tag{1}$$

Skyline queries have been used as convenient means for multi-criteria decision-making since their introduction to database community. An example of a skyline query over hotels in a 2-dimensional space is illustrated in Fig. 1. In the example, the query takes the price and the distance from hotel to beach into account, and returns hotels {a, e, h, i, j} as skyline objects because there does not exist any hotel that is either cheaper (with the same distance), closer to the beach (with the same price), or better in both dimensions.

So far many skyline algorithms have been proposed for efficiently addressing skyline queries. There are algorithms that do not require indexes on the data set, e.g., BNL (Block-Nested-Loop) [2], D&C (Divide and Conquer) [2], SFS (Sort-Filter-Skyline) [6], LESS (Linear Elimination Sort for Skyline) [10], OSPS (Object-based Space Partitioning Skyline) [29], and VSkyline [5]. Others, such as Bitmap [27], Index [27], NN (Nearest Neighbor) [14], BBS (Branch-and-Bound Skyline) [22] [23], ZSearch [18] [17], and SSPL [11], rely on indexes built in a pre-processing stage.

To minimize I/O accesses, BBS searches for skyline candidates by expanding unvisited nodes with the smallest *mindist* in an R-tree. Specifically, BBS iteratively picks a node with the smallest *mindist* from a heap for dominance test against all skyline candidates found so far. Since the dominance test is performed at every level of the R-tree, unqualified objects can be discarded by pruning out their ancestor nodes. ZSearch works in a similar manner except that all objects are indexed in ZBtree by their addresses in the Z-order curve.

However, we observe that BBS and ZSearch suffer from intensive object comparisons in the query evaluation over large object sets due to large heap sizes. In general, all skyline candidates in both algorithms are checked twice with the dominance test. The first dominance test happens before inserting objects into the heap. All newly visited objects are examined against

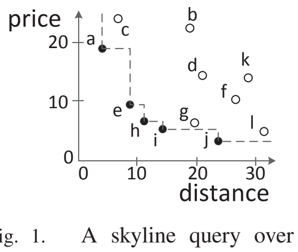


Fig. 1. A skyline query over hotels.

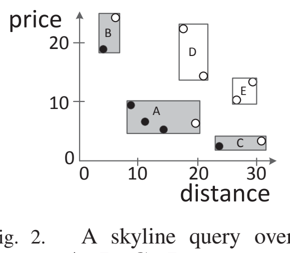


Fig. 2. A skyline query over MBRs. A, B, C, D, and E indicate five MBRs.

skyline candidates found so far, and unqualified objects are discarded in order to minimize the size of the heap. The second dominance test is conducted when the object is selected from the heap for skyline computation.

On the other hand, Sorted Positional Index List (SSPL) is very sensitive to varied object sets because its efficiency relies heavily on the pruning power of the pivot object. In SSPL, all objects are pre-sorted and indexed in every dimension in advance. With the indexes, SSPL can efficiently find a pivot object and discard all objects that have not been scanned. Subsequently, a sorting-based skyline algorithm (e.g., SFS or LESS) is applied to the remaining objects for producing global skyline objects. As our experiments in Section V-B show, the object elimination rate of the pivot object is very low over anti-correlated datasets, and the cost of skyline computation over the remaining objects is very high.

Motivated by the limitations of the state-of-the-art, in this paper we propose a novel approach that improves skyline computation efficiency by minimizing the cost of object comparison. Our solutions combine two novel concepts. The concept of **skyline of Minimum Bounding Rectangles (MBRs)** enables us to perform dominance test among MBRs and maintain a much shorter skyline candidate list than BBS or ZSearch. In this way, the cost of dominance test among candidates in our approach is significantly lower than BBS or ZSearch. Moreover, the **dependent group of MBRs** specifies a smaller search space, in which objects are only needed to compare with the ones that they are dependent on rather than all objects in the input dataset.

Specifically, we first propose new definitions of domination and dependency among MBRs, and develop a number of theorems and properties based on the definitions. The dominance and dependency test do not need to access the detailed attribute values of objects in the MBRs. Then, we construct our advanced solutions by utilizing the two new concepts. Motivated by the hierarchical structure of the R-tree index, in which every intermediate node is an abstraction of an MBR, we apply the two concepts to the R-tree index in our solutions, which can efficiently address the skyline query in three steps. First, the skyline of MBRs is produced by conducting a skyline query over the R-tree of the input dataset. The skyline of MBRs is a subset of parent nodes of leaves not dominated by any other nodes. Second, dependent groups of MBRs (or nodes) are generated for minimizing the cost of object comparison. We develop two dependent group generation methods that rely on either the sorting technique or the R-tree index. Third, the skylines are output by sequentially applying a skyline algorithm (e.g., BNL or SFS) to every dependent group, and the global skylines are the union of the results in the dependent groups. We also present the cardinality estimation of the two new concepts based on a probabilistic model, which is used for further analyzing the computational complexity of the proposed algorithms in Section IV.

We make the following contributions in this paper:

- We propose a new definition of domination among MBRs. In the dominance test for MBRs, no concrete attribute values are required from individual data objects. A number of theorems and properties are derived based on the new definition.
- We propose a novel concept of skyline query over MBRs, which can be used to aggressively filter out unqualified MBRs and objects.
- We also propose a novel concept of dependent group of MBRs to reduce expensive dominance tests.
- We develop a novel skyline solution for efficient skyline query evaluation based on our MBR-oriented designs.
- We use a probabilistic model to estimate the cardinalities of the MBR-oriented skyline and dependent groups. The estimations enable us to analyze the computational complexity of our algorithms.
- We evaluate the performance of the proposed solutions through extensive experiments on synthetic and real datasets.

II. MBR-Oriented Approach

A. Framework Overview

Skyline queries are evaluated in three steps in our solutions, displayed in Fig. 3. In addition to an input object set, our solutions also receive the R-tree index of the object set as inputs. Two algorithms are proposed to compute skyline queries over MBRs. One algorithm (Alg. 1) loads all intermediate nodes of the R-tree into memory, and outputs the exact result of the skyline query, while the other algorithm (Alg. 2) produces a superset result for reducing I/O cost if the R-tree is large. All false positives will be eliminated in the third step. Then, we generate dependent groups of every skyline MBR. Alg. 3 assumes the size of skyline candidates produced in the first step is small, while Alg. 4 and Alg. 5 utilize external sorting and the R-tree index for dependent group generation, respectively. With the dependent groups, objects in an MBR are only compared with the ones in the dependent group of the MBR rather than objects in all skyline MBRs. Finally, all dependent groups are sequentially scanned by using a skyline algorithm (e.g., BNL), and the results of skyline query are the union of skyline objects in the dependent groups.

Since the size of input datasets may vary greatly among cases, we will present both in-memory and external algorithms in the first and second steps. Either of them can be selected according to applications. In this paper, we present skyline solutions that automatically select either Alg. 1 or Alg. 2 by detecting the size of the R-tree, and use Alg. 3, Alg. 4 or Alg. 5 for dependent group generation.

B. Skyline Query over MBRs

Before illustrating our algorithms for the skyline query over MBRs, we will start with the formal definition of the query and its properties. In this paper, an MBR can be abstracted by a triple $M = \langle min, max, ob\_list \rangle$, where min and max

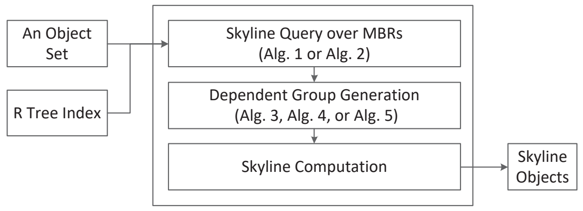

An Object Set
R Tree Index
Skyline Query over MBRs (Alg. 1 or Alg. 2)
Dependent Group Generation (Alg. 3, Alg. 4, or Alg. 5)
Skyline Computation
Skyline Objects

Fig. 3. A framework of the proposed skyline solutions.

specify the minimum and maximum attribute values of objects in the MBR in all dimensions, and *ob_list* is a list of object references. M does not contain detailed attributes of objects in M if we do not explicitly mention them in this paper. The novel query is formally defined as follows.

Definition 3: (*Domination between two MBRs*) Given two MBRs M and M', M dominates M', denoted by $M \prec M'$, if there must exist an object in M, which dominates all possible objects in M'.

The definition can be easily extended from object domination (Definition 1); however they differ at: 1) an MBR may contain many objects. If M only dominates a subset of objects in M', then M does not dominate M'; 2) the detailed attributes of objects in the MBRs are not available in the dominance test. The domination can be decided only by using min and max of MBRs.

Additionally, in a special case, **the domination between an MBR** M **and an object** q is conceptually equivalent to the domination between M and another MBR that contains only one object q. The domination test between two MBRs becomes the object dominance test if both MBRs only contain one object, i.e., min and max are the same for both MBRs.

Fig. 4 shows an example of domination between two MBRs. M dominates B because any object in M dominates all objects in B. Accessing the detailed attributes of objects in M or B is unnecessary. But we cannot determine the domination between M and A by using their min and max. A may contain an object d that is not dominated by any objects in M (M may only contain m_1 and m_3). So, M is incomparable to A.

The transitivity of domination among MBRs holds.

Property 1: (**Domination Transitivity**) Given three MBRs M, M', and M^*, then

$$M \prec M^* \quad \text{if} \quad M \prec M' \quad \text{and} \quad M' \prec M^* \tag{2}$$

Theorem 1: Given two MBRs, M and M', $M \prec M'$ if and only if M' is dominated by at least one pivot point of M, denoted by $PIVOT(M)$,

$$M \prec M' \iff \exists\, q \in PIVOT(M),\ \ q \prec M' \tag{3}$$

where $PIVOT(M) = \{p_k \mid 1 \leq k \leq d\}$,

$$p_k.x^i = \begin{cases} M.min.x^i & \text{if } i = k \\ M.max.x^i & \text{if } i \neq k,\ 1 \leq i \leq d \end{cases} \tag{4}$$

Proof: If $M \prec M'$, we can create a third MBR M^* that contains all objects in $PIVOT(M)$. Apparently, $M^* \prec M'$, because $M^*.min = M.min$ and $M^*.max = M.max$. Thus, we can conclude that there must exist an object in M^* or $PIVOT(M)$, which dominates M'.

Suppose there exists an object $p \in PIVOT(M)$ dominating M'; $p = \{M.max.x^1, ..., M.min.x^i, ..., M.max.x^d\}$. Because M has a lower bound $M.min.x^i$ on the i^{th} dimension, there must exist an object q in M, $q.x^i = M.min.x^i$. Thus, q either dominates or is equal to p. By domination transitivity, q dominates M', and thus $M \prec M'$. ∎

It is worth noting that object attributes are not used in dominance test between two MBRs. In Fig. 4 for example, there might be an object at m_2, but M does not dominate A because the attributes of m_2 are not used in the dominance test. All information used only includes $M.min$, $M.max$, $A.min$, and $A.max$.

The dominance region of an MBR and the power of domination of an MBR can be obtained by extending the dominance region of an object [8] and utilizing the pivot points of the MBR defined in Theorem 1.

Property 2: (**Dominance Region**) By Theorem 1, the dominance region of an MBR M, denoted by $DR(M)$, is the union of the dominance regions of q, $q \in PIVOT(M)$.

$$DR(M) = \bigcup_{q \,\in\, PIVOT(M)} DR(q) \tag{5}$$

Fig. 4 shows an example of the dominance region of M in a 2-dimensional space. Object a is dominated by M because all objects in M dominate a. Moreover, object b is also dominated by M by Theorem 1. There must exist at least one object on the line segment from m_1 and m_2 in M; otherwise the line segment cannot be the minimum boundary of M on x axis. Similarly, object c is dominated by any object on the line segment from m_2 to m_3 in M. Thus, the dominance region of M in the figure is the union of the dominance regions of m_1 and m_3, which is highlighted in grey.

Property 3: (**The Power of Domination**) Given an MBR M, let $PIVOT(M) = \{p_1, ..., p_d\}$, the domination power of M can be evaluated by the fused domination power of $p \in PIVOT(M)$, which can be calculated by

$$\begin{aligned} V_{DR}(M) &= V_{DR}(p_1) + \sum_{1<i\leq d} \left(V_{DR}(p_i) - V_{DR(p_i)\cap DR(p_1)}\right) \\ &= \sum_{p \,\in\, PIVOT(M)} V_{DR}(p) - (|PIVOT(M)| - 1) \times V_{DR}(M.max) \end{aligned} \tag{6}$$

where $V_{DR}(p)$ denotes the volume of the dominance region of p and $|PIVOT(M)|$ represents the number of objects in $PIVOT(M)$. Equ. 6 is straightforward except that the overlapping region of $DR(p_i)$ and $DR(p_1)$ is equal to $DR(M.max)$, the proof of which is given below.

Proof: Given any two objects p_i and p_j in $PIVOT(M)$, we can calculate that $p_i.x^k = p_j.x^k = M.max.x^k$, where $k \neq i$ and $k \neq j$ by Theorem 1. And in the 2-dimensional space containing objects $\{x^i, x^j\}$, $p_i = \{M.min.x^i, M.max.x^j\}$ and $p_j = \{M.max.x^i, M.min.x^j\}$. Thus, it is easy to see that if any object q is dominated by both p_i and p_j, q must be either dominated by or equal to $M.max$. Thus, the overlapping region of the dominance regions of p_i and p_j is equal to the dominance region of $M.max$. Property 3 is proved. ∎

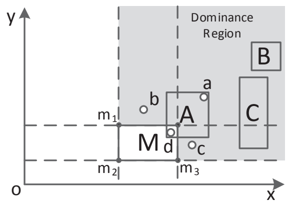


Fig. 4. Examples of dominance regions of MBRs in a two dimensional space.

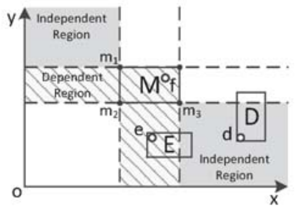


Fig. 5. Examples of dependent regions of MBRs in a two dimensional space.

Property 4: (**Domination Inheritance**) Given two MBRs, M and M', if M dominates M', then M dominates all subsets of M'.

$$M \prec M' \implies M \prec M^*, \text{ where } M^* \subseteq M' \quad (7)$$

Definition 4: (**Skyline Query over MBRs**) Given a set of MBRs, the skyline query over MBRs returns all MBRs that are not dominated by any other MBRs.

Fig. 2 displays an example of the skyline query over MBRs. There are five MBRs in the figure, and $\{A, B, C\}$ are skyline MBRs because D and E are dominated by A.

With the definition of the new skyline query, we will propose two algorithms that receive the R-tree index of the input dataset and output skyline MBRs. In general, the input dataset has been partitioned into small MBRs in the R-tree; these MBRs are abstracted by the intermediate nodes at the bottom of the R-tree. The output of our algorithms is the skylines of these smallest MBRs; however, both algorithms proceed from top to bottom by using the depth-first search. Moreover, the in-memory method (Alg. 1) assumes that all intermediate nodes of the R-tree can fit in memory, while the second method decomposes the R-tree into sub-trees in such a manner that each sub-tree can be loaded into memory. In order to avoid the cost of dominance test among sibling sub-trees, we turn to an alternative method, in which dominance test is only performed in sub-trees. The false positives (the ones dominated by MBRs in sibling sub-trees) in the results will be detected in dependent group generation (the next step) and eliminated in the third step.

Algorithm 1 I-$SKY^{DS}(R_Q)$

```
1: $SKY^{DS}(R_Q)$ = [];
2: while Visit all nodes in $R_Q$ do
3:     Let $M$ be the newly visited node;
4:     for $M' \in SKY^{DS}(R_Q)$ do
5:         if $M' \prec M$ then
6:             Discard $M$ and its descendants (Property 4);
7:         if $M \prec M'$ then
8:             Discard $M'$ from $SKY^{DS}(R_Q)$;
9:     if $M$ is at bottom and is not dominated then
10:        Append $M$ to $SKY^{DS}(R_Q)$;
11: return $SKY^{DS}(R_Q)$;
```

Our in-memory skyline algorithm is shown in Alg. 1, which receives the R-tree index of input dataset Q, denoted by R_Q, and produces all incomparable intermediate nodes (MBRs) at bottom of the R-tree. The fundamental idea of the algorithm is to use the depth-first search method to visit all nodes from top to bottom. During the process, a list, $SKY^{DS}(R_Q)$ (defined at line 1), maintains a set of skyline nodes found so far. All new nodes are compared with these candidates. If newly visited nodes are dominated by any nodes in $SKY^{DS}(R_Q)$, they and their descendants are discarded (at lines 5-6) because child nodes represent subsets of objects covered by the parent node (Property 4). The nodes in $SKY^{DS}(R_Q)$ are removed if they are dominated by any newly visited nodes (at lines 7-8). If new nodes are at the bottom and incomparable with all nodes in $SKY^{DS}(R_Q)$, they are appended to $SKY^{DS}(R_Q)$ as new skyline MBRs (at line 9-10). After all nodes in the R-tree are visited, the nodes in $SKY^{DS}(R_Q)$ are returned as the skyline MBRs of the query.

For the cases of large R-trees, we also propose an external skyline algorithm that decomposes R-trees into sub-trees. Every sub-tree can be loaded into memory and the skyline query over the sub-tree can be addressed by using Alg. 1. There are many tree decomposition methods; Alg. 2 adopts one that maximizes the memory utilization. Specifically, let F be the fan-out of the R-tree and W be the size of memory in nodes, the input R-tree is first partitioned into sub-trees with depth $\lfloor \log_F W \rfloor$ (at line 4). Then, we visit sub-trees from top to bottom and calculate skyline MBRs of the sub-trees using Alg. 1 (at line 8). In the process, a sub-tree is discarded if its root node is eliminated in the dominance test in its parent sub-tree. Thus, we only push the skyline MBRs of $R_{root'}$ to ds (at line 14). If we reach the bottom of the R-tree, the skyline MBRs of sub-trees are written to the output data stream $output$ (at line 12); otherwise, the roots of intermediate sub-trees are pushed to ds, and will be expanded in subsequent iterations. We make a clone of the visited sub-tree at line 7 to highlight that $SKY^{DS}(R_{root'})$ contains skyline MBRs of the sub-tree rather than the ones of the input R-tree. The stop condition of Alg. 1 requires that all nodes at the bottom of sub-trees cannot have any children. Finally, all nodes in $output$ are returned as skyline MBRs.

Algorithm 2 E-$SKY^{DS}(R_Q, W, F)$

```
1: $DataStream\ ds, output$;
2: Let $root$ be the root node of $R_Q$;
3: $ds$.write($root$);
4: $depth$ = $\lfloor \log_F W \rfloor$;
5: repeat
6:     $node$ = $ds$.read();
7:     $root'$ = $node$.clone($depth$);
8:     $SKY^{DS}(R_{root'})$ = I-$SKY^{DS}(R_{root'})$;
9:     for $\forall\ M' \in SKY^{DS}(R_{root'})$ do
10:        Find the corresponding node $M$ in $R_Q$;
11:        if $M.child = \emptyset$ then
12:            $output$.write($M$);
13:        else
14:            $ds$.write($M$);
15: until $ds$ is empty;
16: return $output$;
```

Theoretically, an additional merging step is required for producing exact results; a skyline node in a sub-tree can be dominated by other nodes in its sibling sub-trees. However, we turn to an alternative solution. The reasons are (1) the merging process would suffer from high cost of object comparison and I/O access in large R-trees; skyline candidates in a sub-tree have to compare with all candidates in its sibling sub-trees; (2) all false positives can be detected in dependent group generation process with subtle cost. Instead of comparing with objects in all sibling nodes, we only need to visit dependent sub-trees with the dependent group information.

Fig. 6 displays an example where the R-tree is partitioned into five sub-trees. In the figure, F and W are fixed at 2 and 8, and the depth of sub-trees is 3. If the result of skyline query in the root sub-tree (sub-tree 11) is $\{M_{11}, M_{13}\}$, then only the sub-trees rooted at M_{11} and M_{13} are expanded, and other two sub-trees are discarded.

C. Dependent Groups of MBRs

With skyline MBRs, an intuitive method to find the results of a skyline query over objects is to use a skyline algorithm (e.g., BNL or SFS) that sequentially checks objects in the MBRs. If too many objects are in the MBRs, we have to write intermediate results to a temporary file, and read them back for iterative dominance test, which would be expensive due to the large number of skyline candidates (or the number of iterations). To minimize the cost of the dominance test, we propose a novel concept, dependent groups of MBRs, which specifies the minimum dependent MBRs of a given MBR. With the dependent group information, only objects in the dependent MBRs are retrieved during dominance test rather than reading all objects. In this subsection, we first introduce the definition of dependent group over MBRs and its properties. Then, three dependent group generation algorithms that utilize either main memory or external memory are presented.

Definition 5: (*Dependency between Two MBRs*) Given two MBRs M and M', M is dependent on M' if the determination of skyline objects in M relies on at least one object in M'. In other words, M is independent on M' if the determination of skyline objects in M does not rely on any objects in M'.

Fig. 5 shows an example of the dependency in a two dimensional space. The independent region and dependent region of M are highlighted in light grey. M is independent of object d and MBR D, because none of the objects in D potentially dominates any objects in M. On the other hand, M is dependent on object e and MBR E, as E may contain an object at e, which dominates objects (e.g., f) in M. Thus, all objects in E are required in the dominance test of objects in M.

Theorem 2: Given two MBRs, M and M', M is dependent on M' if $M'.min$ dominates $M.max$ and M is not dominated by M'.

Proof: Theorem 2 provides a method to find dependent MBRs of a given MBR, which can be proved by contradiction. Suppose $M'.min$ dominates $M.max$ and M is not dominated by M', but M is not dependent on M', then there must not exist an object $q \in M'$ that potentially dominates any object in M, which contradicts the assumption that $M'.min$ dominates $M.max$. This completes the proof. ■

It is noteworthy that Theorem 2 focuses primarily on determining the **dependency** of MBRs by using domination of MBRs, while Cui *et al.* [8] developed a way to find **incomparable** MBRs for parallelization by utilizing the dominance region of objects.

Definition 6: (*Dependent Groups of MBRs*) Given an MBR M, the dependent group of M is a set of MBRs on which M is dependent.

Take MBRs in Fig. 5 for example, the dependent group of M is $\{E\}$. The dependent group of E is an empty set, which indicates E is not dependent on any MBR in the figure. D is dominated by E.

Property 5: Given an MBR Q, and a set of disjoint subsets of Q, denoted by $\mathfrak{M}$ where $\bigcup_{M \in \mathfrak{M}} M = Q$, let $DG(M)$ be the dependent group of an MBR M in $\mathfrak{M}$, then $SKY(Q)$ is the union of $SKY^{DG}(M, DG(M))$, where $SKY^{DG}(M, DG(M))$= $\{q \mid q \in M \; and \; q \in SKY(M \bigcup DG(M))\}$.

$$SKY(Q) = \bigcup_{M \in \mathfrak{M}} SKY^{DG}(M, DG(M)) \tag{8}$$

Property 5 demonstrates the basic idea of the third step of our proposed solutions. This step sequentially scans all dependent groups of skyline MBRs, and returns the union of skyline objects in the dependent groups as the global skyline of the input dataset Q. Note that we only produce skylines in M when scanning the dependent group of M ($SKY^{DG}(M, DG(M))$ only contains skyline objects in M), so that there are not duplicate skyline candidates in the global result. The dependent groups of MBRs can be generated by using Alg. 3, Alg. 4, or Alg. 5.

Algorithm 3 I-$DG(\mathfrak{M})$

```
1: $DGMap$ = [];
2: for $M \in \mathfrak{M}$ do
3:     $dependent$ = [];
4:     for $M' \in \mathfrak{M}$, $M' \neq M$ do
5:         if $M \prec M'$ then
6:             Set $M'$ to be dominated;
7:         if $M' \prec M$ then
8:             Set $M$ to be dominated;
9:         if $M$ is dependent on $M'$ (Theorem 2) then
10:            $dependent$.append($M'$);
11:    $DGMap$.append($\langle M$, $dependent \rangle$);
12: return $DGMap$;
```

Alg. 3 describes an in-memory dependent group generation algorithm, which assumes that the input set of MBRs $\mathfrak{M}$ can be loaded into memory. In particular, a map $DGMap$ is created for keeping dependent groups of all input MBRs. We check the dependency of every pair of MBRs in the for-loop (from line 2 to 11), and mark all MBRs being dominated. These sets (false positives generated by Alg. 2) will be skipped (or eliminated) in the third step. Finally, $DGMap$ is returned as the result of Alg. 3.

It is worth noting that Alg. 3 can be used for the cases where the R-tree of an input dataset can be loaded into memory. $\mathfrak{M}$ abstracts the set of all nodes (or MBRs) at the bottom of the R-tree, and $DGMap$ only contains the nodes that are not dominated by any other nodes.

We also propose two external alternatives for generating dependent groups on large datasets. Alg. 4 first sorts the input

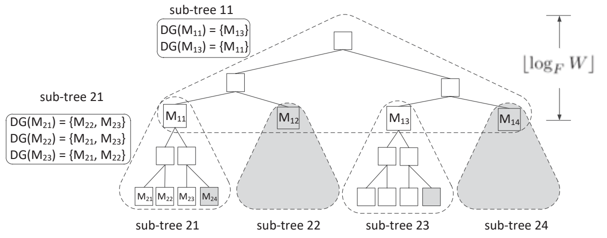


Fig. 6. The sub-tree structure in an R-tree.

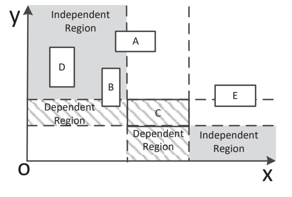


Fig. 7. The dependent group of MBR C.

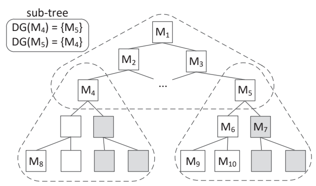


Fig. 8. An example of dependent group generation by using Alg. 5.

MBRs in an ascending order on the i^{th} dimension, and then sweeps the MBRs from the minimum value on the dimension to the maximum value. With the sorting, the dependency detection of an MBR M can stop at the place $M.max.x^i$, because M cannot be dependent on any MBR M', $M'.min.x^i$ > $M.max.x^i$ (at line 11). We also mark the MBRs that are dominated and write them to the output stream instead of applying an additional elimination step, because this can be completed by skipping all these MBRs in the third step of our solutions.

Algorithm 4 E-DG-1($\mathfrak{M}$)

```
1: $DataStream\ output$;
2: Sort $M \in \mathfrak{M}$ by $M.min.x^i$ in an ascending order;
3: for ($i = 0; i < |\mathfrak{M}|; i++$) do
4:     $dependent = []$;
5:     for ($j = 0; j < |\mathfrak{M}|; j++$) do
6:         if $\mathfrak{M}[j] \prec \mathfrak{M}[i]$ then
7:             Set $\mathfrak{M}[i]$ to be dominated;
8:             Break;
9:         if $\mathfrak{M}[i] \prec \mathfrak{M}[j]$ then
10:            Set $\mathfrak{M}[j]$ to be dominated;
11:        if $\mathfrak{M}[i].max.x^i < \mathfrak{M}[j].min.x^i$ then
12:            $output$.write($\langle\mathfrak{M}[i]$, $dependent\rangle$);
13:            Break;
14:        if $\mathfrak{M}[i]$ is dependent on $\mathfrak{M}[j]$ (Theorem 2) then
15:            $dependent$.append($\mathfrak{M}[j]$);
16: return $output$;
```

Fig. 7 displays an example of Alg. 4 in a two dimensional space. When MBR C is visited ($\mathfrak{M}[i]$ = C in Alg. 4), then we only scan MBRs D, B, and A. C is not dependent on E. Thus, the dependent group of C is $\{B\}$.

Before describing the second external dependent group generation algorithm, we first present two properties used in the algorithm.

Property 6: Given two MBRs, M and M', if M is independent of M', then M is not dependent on any subsets of M'.

Proof: The proof is trivial, and thus is omitted. ∎

Property 7: Given two MBRs, M and M', if M is dependent on M', then M may be dependent on either all or a portion of disjoint subsets of M'.

Proof: The proof is trivial, and thus is omitted. ∎

Alg. 5 relies on the R-tree index of the input dataset with an assumption that the dependent group of every sub-tree has been calculated and associated with the root node of the sub-tree in advance. The dependent groups of sub-trees can be generated by applying Alg. 3 to the sub-trees during the process of skyline calculation in Alg. 1 or Alg. 2. Specifically, Alg. 5 starts with sub-trees at the bottom, and traces back to the root sub-tree in order to find all possibly dependent nodes by iterating the R-tree. At an ancestor sub-tree, we only expand the search space at the sibling nodes that the visited node is dependent on (Property 7). All independent nodes can be skipped by using Property 6. In the for-loop from line 3 to 24, M refers to a node in the input set, and M' always points to the roots of sub-trees. In the while-loop from line 6 to 9, the nodes that any ancestors of M are dependent on are pushed to a data stream ds. In the while-loop from line 10 to 22, all descendant nodes of the ones in ds are visited and checked for dependency with M. Similar to Alg. 4, any nodes that are marked as dominated in the process will be eliminated in the final step of our solutions.

Algorithm 5 E-DG-2($\mathfrak{M}$, R_Q)

```
1: $DataStream\ ds$, $output$;
2: Let $root$ be the root node of $R_Q$;
3: for $\forall M \in \mathfrak{M}$ do
4:     $M' = M$
5:     $w = DG(M)$ in the sub-tree of $M$;
6:     while $M' \neq root$ do
7:         $M' = M'$.parent;
8:         if $M'$ has a dependent group map then
9:             Push all dependent nodes of $M'$ in $ds$;
10:    while $ds$ is not empty do
11:        $M' = ds$.read();
12:        if $M' \prec M$ then
13:            Set $M$ to be dominated;
14:            Break;
15:        if $M \prec M'$ then
16:            Set $M'$ to be dominated;
17:            Continue;
18:        if $M$ is dependent on $M'$ then
19:            if $M'$ is a node at bottom then
20:                $w$.append($M'$);
21:            else
22:                Push $SKY^{DS}(M')$ to $ds$;
23:    if $M$ is not dominated then
24:        $output$.write($\langle M$, $w\rangle$)
25: return $output$;
```

Fig. 6 also illustrates an example of dependent groups associated with sub-trees. The example assumes that the dependent groups in sub-tree 11 are $DG(M_{11})$ = $\{M_{13}\}$, and $DG(M_{13})$ = $\{M_{11}\}$. Fig. 8 displays an example of finding the dependent group of M. Assume that Alg. 5 is visiting M_8 and its ancestor node M_4, and pushing all nodes in the dependent group of M_4 to ds. Subsequently, when M_4 is read at line 12, if M_4 is dependent on M_5, then we push the skyline nodes in the sub-tree rooted at M_5 for the next iteration. If M_7 is eliminated in the skyline query of the sub-tree, then M_8 is only compared with M_9 and M_{10} for the dependency test in the example.

Important Optimization. The last step of our solutions is to sequentially calculate the skyline objects in the dependent groups. There are two optimization methods that can further reduce the cost of the skyline computation. First, the order of processing dependent groups is important; starting with small dependent groups would be more efficient than starting with big ones; loading a small group incurs less I/O cost. There is a higher possibility that all objects can be fit in memory. Second, when objects in an MBR M and its dependent MBRs are loaded, we (1) discard objects in M if they are dominated by any other objects in M or its dependent MBRs; (2) discard objects in the dependent MBRs if they are dominated by any objects in M. After the dominance test, objects remaining in M are the output of the skyline query in the dependent group. The dependent MBRs would have fewer objects, which potentially reduces the cost of skyline computation in other dependent groups containing one of these MBRs. Moreover, the dominance test is not performed on objects between two dependent MBRs, because their dependency is not described by the dependent group of M.

Comparison with BNL and SFS. We compare the second and third steps of our solutions with the one that directly uses BNL or SFS after obtaining the skyline MBRs (the first step) in terms of number of object comparisons. Using BNL for example, given a set of MBRs $\mathfrak{M}$ and the average size of MBRs $|M|$, the number of object comparisons of BNL over $\mathfrak{M}$ is $n(n-1)/2$, where $n = |\mathfrak{M}| \times |M|$ denotes the total number of objects in $\mathfrak{M}$. On the other hand, in our solutions, the dependent group generation takes $|\mathfrak{M}|^2$ in the worst case. Moreover, let A be the average size of dependent groups, then the number of object comparisons in the third step of our solutions is $A \times |M| \times |M| \times |\mathfrak{M}|$, where $A \times |M| \times |M|$ indicates the cost of comparing objects in M with all other objects in the dependent group, and there are $|\mathfrak{M}|$ dependent groups in total for the query evaluation. The cost can be further reduced by applying the optimization, which only reads the skylines in MBRs once they have been calculated. Thus, the cost of the second and third step in our solution is $|\mathfrak{M}|^2 + A \times |SKY(M)|^2 \times |\mathfrak{M}|$, which is less than the cost of BNL ($\frac{(|\mathfrak{M}|\times|M|)(|\mathfrak{M}|\times|M|-1)}{2}$) in most cases. It is worth noting that our solutions degrade to BNL in the worst case, in which all MBRs are dependent on each other. A similar method can be applied to the comparison between our methods and SFS with pre-sorted objects.

III. Cardinality Estimation

We provide an estimation of the cardinality of skyline query and dependent group of MBRs in this section. The cardinality estimation will be used for analyzing the computational complexity of our proposed algorithms in Section IV.

A. Cardinality of Skyline Query over MBRs

1) Discrete Data Space: Given a discrete data space $\mathbb{R}^d = [0, n^i)^d$, where n^i indicates the upper bound of the data space in the i^{th} dimension. The attribute values of objects under a uniform data distribution are evenly distributed across the data space in each dimension.

Theorem 3: Let $|M|$ be the number of objects in an MBR M, and $P(M = [x_l^i, x_u^i]^d)$ denote the probability that M is bounded by $[x_l^i, x_u^i]^d$ (x_u^i - x_l^i > 1), then

$$P(M = [x_l^i, x_u^i]^d, |M|) = \prod_{1 \le i \le d} \left(\frac{\sum_{j=1}^{|M|-1} \sum_{k=1}^{|M|-j} \left(\binom{|M|}{j} \binom{|M|-j}{k} (x_u^i - x_l^i - 1)^{|M|-j-k} \right)}{(n^i)^{|M|}} \right) \tag{9}$$

where in the i^{th} dimension, $\binom{|M|}{j}$ represents the number of ways of selecting j objects q_j ($q_j.x^i$ = $M.x_l^i$) from an MBR of $|M|$ objects; similarly, $\binom{|M|-j}{k}$ indicates the number of ways of selecting k objects q_k ($q_k.x^i$ = $M.x_u^i$) from the remaining objects, where $j \ge 1$, $k \ge 1$, and $j + k \le |M|$. Then, after selecting q_j and q_k, the number of combinations of the remaining $|M| - j - k$ objects in the range of ($M.x_l^i$, $M.x_u^i$) is $(x_u^i - x_l^i - 1)^{|M|-j-k}$. Thus, with the consideration of the independence of object attributes and the total number of object selections ($(n^i)^{|M|}$), Equ. 9 calculates the probability of an MBR with a given bound $[x_l^i, x_u^i]^d$ under a uniform object distribution. There are two special cases: (1) if x_u^i = x_l^i, then the attribute values of all objects must be $M.x_l^i$, and we have $P(M = [x_l^i, x_u^i]^d, |M|) = \prod_{1 \le i \le d} (\frac{1}{n^i})^{|M|}$; (2) if x_u^i - x_l^i = 1, then there is no object falling in the range of ($M.x_l^i$, $M.x_u^i$), and in the i^{th} dimension, the probability that M is bounded by $[x_l^i, x_u^i]$ is $\left(\sum_{j=1}^{|M|-1} \binom{|M|}{j} \right) / (n^i)^{|M|}$.

Given an MBR M', the probability that a random MBR M is dominated by M', denoted by $P(M' \prec M)$, is equal to the probability that M is dominated by any object in $PIVOT(M')$ by Theorem 1. The probability that M is dominated by a point can be evaluated by the probability that M is in the dominance region of the point. Thus, we can obtain $P(M' \prec M)$ by using Property 3.

Theorem 4: Given an MBR $M' = [x_l'^i, x_u'^i]^d$, let $P(M' \prec M)$ denote the probability that a random MBR $M = [x_l^i, x_u^i]^d$ is dominated by M', then

$$P(M' \prec M) = \sum_{p \in PIVOT(M')} P(p \prec M) - (|PIVOT(M')| - 1) \times P(M'.x_u' \prec M) \tag{10}$$

where

$$P(p \prec M) = \sum_{\substack{p.x^i < M.x_l^i \le M.x_u^i < n^i \\ 1 \le i \le d}} P(M = [x_l^i, x_u^i]^d, |M|) \tag{11}$$

$P(M' \prec M)$ is the total sum of the probability that M is possibly dominated by pivot points of M'. Since the dominance regions of the pivot points overlap with each other, we borrow the idea of Property 3, in which the volume of the overlapping region is equal to the volume of the dominance region of $M'.x_u'$. Moreover, $P(p \prec M)$ indicates the probability that M is dominated by a point p, which is the total probability of all possible $M.x_l$ that are dominated by p. With the probability of domination between two MBRs, we can get the probability that an MBR is a skyline MBR in a given set of MBRs as follows.

Theorem 5: Given a set of MBRs $\mathfrak{M}$ and an MBR $M \in \mathfrak{M}$, let $P(M \in SKY^{DS}(\mathfrak{M}))$ be the expected probability that M is in skyline MBRs of $\mathfrak{M}$, $SKY^{DS}(\mathfrak{M})$, then

$$P(M \in SKY^{DS}(\mathfrak{M})) = (|\mathfrak{M}| - 1) \times \prod_{\substack{0 \le x_l^i \le x_u^i < n^i \\ 1 \le i \le d}} \Big((1 - P(M' \prec M)) \times P(M' = [x_l^i, x_u^i]^d, |M'|) \Big) \tag{12}$$

where $(1 - P(M' \prec M)) \times P(M', |M'|)$ represents the probability that M' is in $\mathfrak{M}$ and M' does not dominate M. In fact, $\mathfrak{M}$ has $|\mathfrak{M}|$ MBRs, and $P(M \in SKY^{DS}(\mathfrak{M}))$ calculates the probability that M is not dominated by any other MBRs in $\mathfrak{M}$.

Theorem 6: Given a set of MBRs $\mathfrak{M}$, let $|SKY^{DS}(\mathfrak{M})|$ be the expected cardinality of skyline MBRs of $\mathfrak{M}$, then

$$|SKY^{DS}(\mathfrak{M})| = |\mathfrak{M}| \times \sum_{\substack{0 \le x_l^i \le x_u^i < n^i, \\ 1 \le i \le d}} (P(M = [x_l^i, x_u^i]^d, |M|) \times P(M \in SKY^{DS}(\mathfrak{M}))) \tag{13}$$

where $P(M = [x_l^i, x_u^i]^d, |M|) \times P(M \in SKY^{DS}(\mathfrak{M}))$ represents the probability that M is in $\mathfrak{M}$ and M is in the skyline MBRs of $\mathfrak{M}$.

2) Continuous Data Space: In a continuous data space $\mathbb{R}^d$ = $[0, n_i]^d$, an object is represented by $\vec{x}$ for convenience in this paper. We assume that given a sufficiently large dataset, the data distribution can be represented by a joint density function $f(\vec{x})$, which describes the probability density of object $q = \{\vec{x}\}$ in the dataset. For example, the joint density function of a uniform dataset can be represented by $f(\vec{x}) = \prod_{1 \le i \le d}(\frac{1}{n_i})$

Theorem 7: In a continuous data space $\mathbb{R}^d$ = $[0, n_i]^d$, let $P(M = [\vec{x_l}, \vec{x_u}])$ denote the probability that M is bounded by $[\vec{x_l}, \vec{x_u}]$, where x_u^i > x_l^i in the i^{th} dimension, then

$$P(M = [\vec{x_l}, \vec{x_u}],\ |M|) = \left(\int_{\vec{x_l}}^{\vec{x_u}} f(\vec{x})\ d\vec{x} \right)^{|M|} \tag{14}$$

Theorem 8: Given an MBR $M' = [\vec{x'_l}, \vec{x'_u}]$, let $P(M' \prec M)$ denote the probability that a random MBR M is dominated by M', then $P(M' \prec M)$ can be calculated by Equ. 10, where

$$P(p \prec M) = \int_{M \in [p.\vec{x^i},\ \vec{n^i}]} P(M,\ |M|) \tag{15}$$

By using Equation 12, the expected cardinality of a set of MBRs can be calculated as follows.

Theorem 9: Given a set of MBRs $\mathfrak{M}$, let $|SKY^{DS}(\mathfrak{M})|$ be the expected cardinality of skyline MBRs of $\mathfrak{M}$, then

$$|SKY^{DS}(\mathfrak{M})| = |\mathfrak{M}| \times \int_{M \in \mathbb{R}^d} (P(M, |M|) \times P(M \in SKY^{DS}(\mathfrak{M}))) \tag{16}$$

B. Cardinality of Dependent Groups of MBRs

We provide the cardinality estimation of dependent groups of MBRs by using Theorem 2.

Theorem 10: Given an MBR M, let $P(M' \in DG(M))$ denote the probability that there is an MBR M' in the dependent group of M, then

$$\begin{aligned} P(M' \in DG(M)) &= P(M'.p_l \prec M.p_u) - P(M'.p_u \prec M.p_l) \\ &= \int_{M'.\vec{p_l} \in [\vec{0},\ M.\vec{p_u}]} P(M', |M'|) - \int_{M'.\vec{p_u} \in [\vec{0},\ M.\vec{p_l}]} P(M', |M'|) \end{aligned} \tag{17}$$

Theorem 11: Given an MBR M in a set of MBRs $\mathfrak{M}$, let $|DG(M)|$ be the expected size of the dependent group of M, then

$$|DG(M)| = (|\mathfrak{M}| - 1) \times \int_{M' \in \mathbb{R}^d} \left(P(M', |M'|) \times P(M' \in DG(M)) \right) \tag{18}$$

where $P(M', |M'|) \times P(M' \in DG(M))$ indicates the probability that M is dependent on a random MBR M' and M' is in $\mathfrak{M}$.

IV. Algorithm Analysis

A. Complexity of Skyline Query over MBRs

We assume that the input R-tree of our algorithms is a complete tree. All objects are generated under a uniform data distribution and randomly distributed among nodes at bottom of the tree. We will start with the analysis in our in-memory skyline query algorithm (Alg. 1), and then extend it to the external memory scenario (Alg. 2).

Alg. 1 loads all nodes into memory; if a node is dominated by any other nodes visited so far, the node and its descendants are discarded without access. We estimate the cost of Alg. 1 by evaluating the probability of each node that would be accessed. The cost of dominance test for every node is linear to the number of skyline nodes in the precedent nodes of currently visited node. Since the R-tree is a complete tree, it is easy to get the number of precedent nodes of a given node. We use $Prec(M)$ to denote the set of M's precedent nodes that directly contain references of input objects. Take Fig. 6 for example, $Prec(M_{12}) = \{M_{21}, M_{22}, M_{23}, M_{24}\}$. The cost of dominance test for M_{12} is $|SKY^{DS}(Prec(M_{12}))|$ (See Equ. 16), comparing M_{12} with skyline MBRs found so far.

Next, we focus primarily on the probability of every node being accessed by Alg. 1. Let M be a node in the R-tree (M is not the root node), M_p be the parent node of M and M_{pp} be the parent node of M_p, the probability of a node being accessed can be calculated by using the following three rules (recursion): (1) the root node cannot be dominated; (2) if M_p is accessed, M can be pruned if M_p is dominated by a node in $Prec(M_p)$; (3) if M_p is accessed, M will be accessed if M_p is not dominated by any node in $Prec(M_p)$. Thus, let $P_A(M)$ and $P_D(M)$ be the probability that M is accessed and dominated in the evaluation process, respectively, $P_A(M)$ is equal to the probability that M_p is not dominated by any nodes in $Prec(M_p)$ in the condition that M_p is accessed. By the properties of conditional probability, we can get

$$\begin{aligned} P_A(M) &= P(\ M_p \not\prec Prec(M_p) \mid M_p \text{ is access }) \\ &= P(\ M_p \not\prec Prec(M_p)\ \cap\ M_p \text{ is access })/P_A(M_p) \\ &= P(\ M_p \not\prec Prec(M_p)\ \cap\ M_{pp} \not\prec Prec(M_{pp})\ )/P_A(M_p) \end{aligned} \tag{19}$$

To calculate $P(M_p \not\prec Prec(M_p) \cap M_{pp} \not\prec Prec(M_{pp}))$, two properties are important: (1) M_p is a subset of M_{pp}; if M_{pp} is not dominated by a set of MBRs, M_p cannot be dominated by the set; (2) $Prec(M_p)$ is a superset of $Prec(M_{pp})$, because there might be precedent nodes of M_p in the sub-tree rooted at M_{pp}. Thus, the cases of $M_p \not\prec Prec(M_p)$ contain the cases of $M_p \not\prec Prec(M_{pp})$, which contain the cases of $M_{pp} \not\prec Prec(M_{pp})$. Then, combined with Equ. 19, we can get

$$\begin{aligned} P_A(M) &= P(\ M_p \not\prec Prec(M_p)\ \cap\ M_{pp} \not\prec Prec(M_{pp})\ )/P_A(M_p) \\ &= P(\ M_p \not\prec Prec(M_p)\ )/P_A(M_p) \end{aligned} \tag{20}$$

Take Fig. 8 for example, M_1 is the root node; $P_A(M_1)$ = 1 and $P_D(M_1) = 0$. Then, M_2 and M_3 are two child nodes of M_1, so $P_A(M_2)$ = $P_A(M_3)$ = 1; $P_D(M_2)$ = 0, because $Prec(M_2)$ is an empty set; $P_D(M_3)$ = $P(Prec(M_3) \prec M_3)$. $P_A(M_4)$ and $P_A(M_5)$ can be calculated in a similar manner.

The expected computational complexity and I/O cost of Alg. 1 over the R-tree R_Q are

$$\begin{aligned} ECC_{I-SKY}(R_Q) &= \sum_{M \in R_Q} (P_A(M) \times |Prec(M)|) \\ EIO_{I-SKY}(R_Q) &= \sum_{M \in R_Q} (P_A(M)) \end{aligned} \tag{21}$$

The cost of our external skyline algorithm (Alg. 2) can be estimated by using Equ. 21. Alg. 2 iterates all sub-trees, the number of which can be estimated by the depth of the R-tree and the sub-trees. Let L be the level of sub-trees in the R-tree, then, the expected computational complexity and I/O cost of Alg. 2 are

$$\begin{aligned} ECC_{E-SKY}(R_Q) &= \left(\sum_{0 \le i < L} |SKY^{DS}(\mathfrak{M}_S)|^i \right) \times ECC_{I-SKY}(S) \\ EIO_{E-SKY}(R_Q) &= \left(\sum_{0 \le i < L} |SKY^{DS}(\mathfrak{M}_S)|^i \right) \times EIO_{I-SKY}(S) \end{aligned} \tag{22}$$

where S denotes a sub-tree of the R-tree (assuming the sizes of all sub-trees are the same), and $|SKY^{DS}(\mathfrak{M}_S)|^i$ represents the number of sub-trees accessed at level i. For example, Alg. 2 accesses one sub-tree at the top level (i=0), and $|SKY^{DS}(\mathfrak{M}_S)|$ sub-trees at the second level (i=1). There might be sub-trees that are discarded without access at the second level because their root nodes have been eliminated when the top sub-tree is accessed. Thus, the total number of sub-trees accessed in the R-tree is $\sum_{0 \leq i < L} |SKY^{DS}(\mathfrak{M}_S)|^i$.

B. Complexity of Dependent Group Generation

Our in-memory dependent group generation method examines the dependency between every pair of MBRs, the computational complexity of which is O($|\mathfrak{M}|^2$), where $\mathfrak{M}$ denotes the input set of MBRs. The I/O cost is O($|\mathfrak{M}|$). In Alg. 4, we first sort all datasets in order in a particular dimension, both the computational complexity and I/O cost of which are $O(|\mathfrak{M}| \times log_W(\frac{|\mathfrak{M}|}{W}))$, where W is the size of memory in MBRs. Then, we use a sliding window to detect the dependency among datasets. Since the expected size of dependent groups is provided in Theorem 11, the cost of the dependency test can be estimated by $O(A \times |\mathfrak{M}|)$, where A denotes the expected size of the dependent group of $\mathfrak{M}$, which can be calculated by using Equ. 18. Thus, the total computational complexity and I/O cost of Alg. 4 are

$$\begin{aligned} ECC_{E-DG-1}(\mathfrak{M}) = O(|\mathfrak{M}| \times (log_W(|\mathfrak{M}|/W) + A)) \\ EIO_{E-DG-1}(\mathfrak{M}) = O(|\mathfrak{M}| \times (log_W(|\mathfrak{M}|/W) + A)) \end{aligned} \tag{23}$$

Alg. 5 compares every MBR with all MBRs in its sibling sub-trees. All dependent group information in sub-trees is used for searching and expanding dependent sub-trees. Let L be the level of sub-trees in the input R-tree R_Q and M be a node at bottom of R_Q, to calculate the dependent group of M, the number of nodes needed to compare with M is $O(A^L)$, because A child nodes are expected to be visited when expanding a sub-tree. The expected skyline MBRs of the R-tree R_Q can be calculated by using Equ. 16. So, we arrive at the computational complexity and I/O cost of Alg. 5 as follows.

$$\begin{aligned} ECC_{E-DG-2}(\mathfrak{M}) = O(A^L \times |SKY^{DS}(R_Q)|) \\ EIO_{E-DG-2}(\mathfrak{M}) = O(A^L \times |SKY^{DS}(R_Q)|) \end{aligned} \tag{24}$$

V. Experimental Validation

In this section, we evaluate the performance of the proposed skyline solutions over both synthetic and real-world datasets. We implemented two proposed solutions: $SKY\text{-}SB$ uses the sorting-based (Alg. 4) dependent group generation method; $SKY\text{-}TB$ produces dependent groups by utilizing the R-tree index (Alg. 5). We also implemented BBS, ZSearch, and SSPL as three index-based skyline baselines, since they are among the most efficient index-based skyline solutions in literature. The R-tree and ZBtree were created by using the Nearest-X and Sort-Tile-Recursive (STR) bulk-loading methods in a pre-processing stage [19]. The average result of using the two methods will be displayed in the following subsections. The execution time of the index creation is not included in our experimental results. All objects in heap are kept in memory in BBS and ZSearch. SSPL pre-sorts input datasets in every dimension in a pre-processing stage and uses SFS to produce skylines in the second step. All experiments were conducted on a CentOS Linux server with two Intel Xeon E5530 2.40 GHz processors and 16 GB of memory. All solutions were implemented in Java.

Two real-world datasets were downloaded from IMDb[1] and Tripadvisor[2]. The IMDb dataset contains 680,146 movie reviews, each of which consists of an overall rating and the number of votes for the movie. The Tripadvisor dataset has 240,060 hotel ratings in 7 dimensions. We also randomly generated uniform and anti-correlated datasets in a $[0, 10^9]^d$ space. The skyline query over the two datasets is challenging due to the large number of object comparisons and I/O cost. The synthetic datasets vary the size from 20 K to 1 M objects with respect to 2-8 dimensions. The MBRs at the bottom of the R-tree generated by STR keeps a similar distribution with object distribution; while Nearest-X splits the space into MBRs of equal size on the first dimension. All datasets and R-tree indexes are initially on disk, and then loaded into memory only when they are required by solutions.

A. Effect of Dataset Cardinality

Fig. 9 displays the execution time, the number of accessed nodes, and the number of object comparisons of the five solutions with respect to varying dataset cardinality. The fanout of the R-tree and ZBtree is fixed at 500. The dimensionality of datasets is fixed at 5. We observe that all solutions consume more execution time over larger uniform and anti-correlated datasets. Although SKY-SB and SKY-TB access more nodes in the experiments, they outperform other solutions (on average 34% and 128% faster than ZSearch and SSPL over uniform datasets) by significantly reducing the number of object comparisons in the query evaluation (because the skyline computation is CPU-intensive [13]). This can be explained by the following four points. First, BBS seriously suffers from maintaining objects in heap. The cost of object comparison dominates the query evaluation process rather than the I/O cost. SKY-SB and SKY-TB access around one thousand more nodes in the worst case, which could be completed in 10 seconds[3]. Moreover, as Fig. 9(e) displays, the number of object comparisons for finding objects that have smallest $mindist$ ranges from 550 million (over 200 K uniform dataset) to 5.5 billion (over 1 M uniform dataset), the cost of which is significantly higher than the cost of the dominance test, since the number of objects in the heap is much greater than the number of skyline candidates.

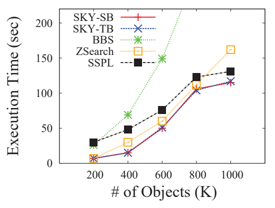
Second, because of the close relation between the skyline query and the Z-order curve, ZSearch has a smaller heap and less object comparisons (2.2 billion object comparisons over 1 M uniform dataset).

Third, SSPL performs 199 million object comparisons when answering skyline queries over 1 M uniform dataset. There are around 151 K candidates remaining after the first scan. Although a majority of objects have been eliminated, applying SFS to these candidates is still expensive. Moreover, there is an additional step that merges all visited objects in indexes after the first scan, which incurs additional cost of object comparison.

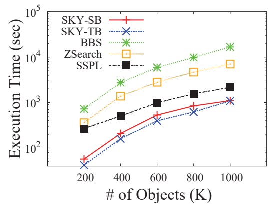
[1]http://www.imdb.com/interfaces

[2]http://www.tripadvisor.com/

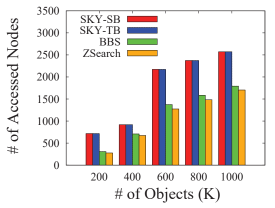
[3]The random access rate is around 1 page of 4 KBytes per 10 milliseconds.



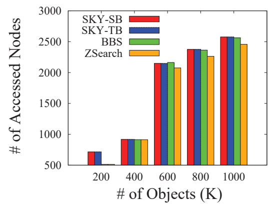
(a) Uniform datasets.



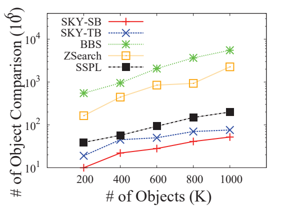
(b) Anti-correlated datasets.



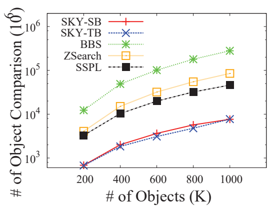
(c) Uniform datasets.



(d) Anti-correlated datasets.

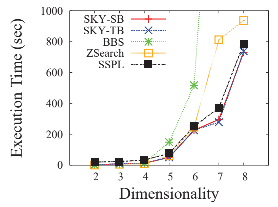


(e) Uniform datasets.

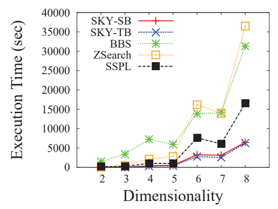


(f) Anti-correlated datasets.

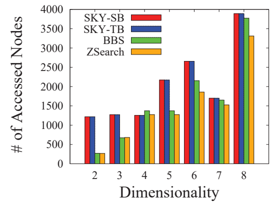
Fig. 9. Varying dataset cardinality.



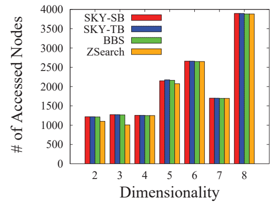
(a) Uniform datasets.



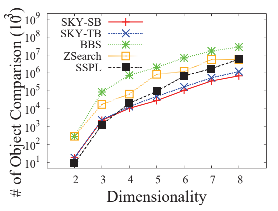
(b) Anti-correlated datasets.



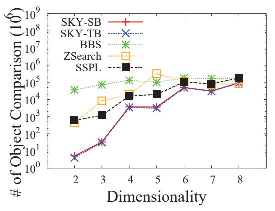
(c) Uniform datasets.



(d) Anti-correlated datasets.



(e) Uniform datasets.



(f) Anti-correlated datasets.

Fig. 10. Varying dataset dimensionality.

Last, SKY-SB and SKY-TB produce around 2K skyline MBRs after performing a skyline query over the R-tree index of 1M uniform dataset, which can complete in 4-5 seconds. Then, the dependent groups of these MBRs are generated, and the average size of the dependent groups is around 1K. Finally, global skylines are output by performing 51 million object comparisons in SKY-SB and 76 million object comparisons in SKY-TB, which is much less than that of BBS, ZSearch, or SSPL.

The benefits of using SKY-SB and SKY-TB become larger over 1M anti-correlated datasets (on average 1.0-14.4 times faster than other solutions), because BBS and ZSearch maintain relatively greater number of objects in heap during the query evaluation. In SSPL, the pivot object can only eliminate 2% of objects in the first scan, which is much smaller than 85% over uniform datasets. In SKY-SB and SKY-TB, there is no MBR eliminated in skyline query evaluation over MBRs; however, the average size of dependent groups is around 600, which is around half size of the total number of MBRs. As Fig. 9(f) displays, the total numbers of object comparisons in SKY-SB and SKY-TB are 7.6 and 7.7 billion, which is much smaller than BBS (290 billion), ZSearch (85 billion), and SSPL (46 billion).

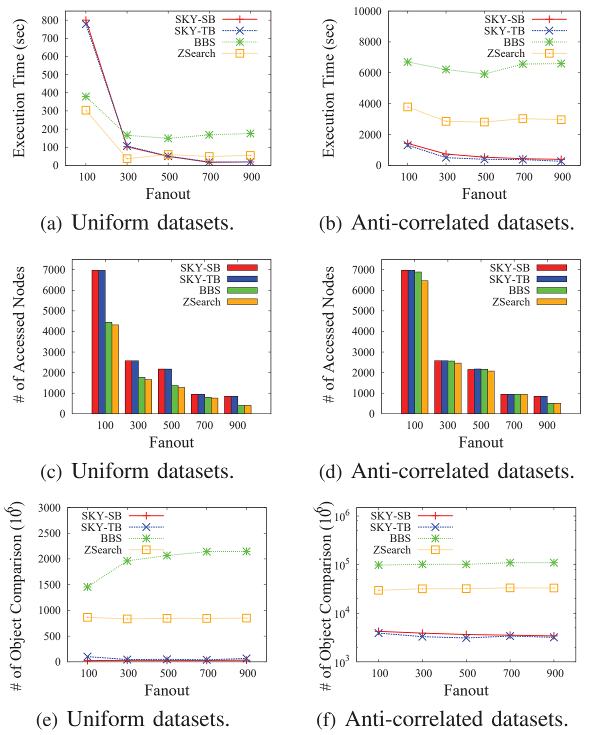
B. *Effect of Dataset Dimensionality*

The effect of dataset dimensionality is displayed in Fig. 10. The cardinality of uniform and anti-correlated datasets is fixed at 600K. As the dimensionality grows, the number of skyline candidates increases because the probability of objects being dominated is lower in a higher dimensional space. Thus, all solutions need more object comparisons and execution time in the skyline query evaluation. SKY-SB and SKY-TB do not eliminate MBRs in a high dimensional space by using the skyline query over MBRs (thereby accessing more nodes); however, dependent group information helps significantly reduce the number of object comparisons, and achieves, on average, 230% and 240% improvement in terms of execution time over other solutions. Moreover, the execution time of BBS and ZSearch grows fast because the number of intermediate objects in heap increases rapidly. In SSPL, the elimination rate of the pivot point decreases from 99.2% in 2-dimensional space to 30% in 8-dimensional space over uniform datasets, and keeps in the range of 0% to 10% over anti-correlated datasets. The increase in the execution time of SSPL is primarily contributed by SFS, the cost of which is quadratic to the number of intermediate objects. One more interesting observation is that the number of accessed nodes of all solutions in 7-dimensional space is smaller than the ones in 6-dimensional or 8-dimensional space, because the nodes in R-tree or ZBtree have fewer nodes in 7-dimensional space.[4]

C. *Effect of Fan-out of R-tree and ZBtree*

The fanout of the R-tree is important to our proposed solutions. The more objects there are in an MBR, the more time our solutions could save if the MBR is dominated during

[4]In our implementation of Sort-Tile-Recursive (STR) bulk-loading method, every tile contains an MBR in equal size. In a d-dimensional space, we sort all objects on the i^{th} dimension, and partition the space into N^i $\{d\text{-}i\}$-dimensional subspaces; each subspace contains an identical number of objects. Thus, the number of tiles, or the number of nodes at the bottom of R-tree or ZBtree, is N^d. Moreover, both datasets contain 600K objects; the fan-out is fixed at 500. So, there are at least 1,200 tiles in R-tree or ZBtree. We select the smallest N in our experiments, and the number of tiles in 7-dimensional space is $3^7 = 2187$, which is much smaller than the number of tiles in 6-dimensional ($4^6 = 4096$) and 8-dimensional ($3^8 = 6561$) space. The Nearest-X bulk-loading method only sorts objects on the 1^{st} dimension and generates identical number of tiles in experiments.



(a) Uniform datasets. (b) Anti-correlated datasets.

(c) Uniform datasets. (d) Anti-correlated datasets.

(e) Uniform datasets. (f) Anti-correlated datasets.

Fig. 11. Varying the fanout of R-tree and ZBtree.

the query evaluation. On the other hand, the probability of the MBR being dominated becomes lower, because the MBR has higher opportunity to have objects that are not dominated by other MBRs. Thus, to evaluate the effect of fan-out setting on skyline solutions, we vary the fanout of the R-tree and ZBtree from 100 to 900.[5] The datasets contain either 600K uniform objects or 600K anti-correlated objects in a 5-dimensional space.

Fig. 11 presents the execution time, the number of accessed nodes, and the number of object comparisons in the query evaluation by four solutions. SSPL is not presented because it does not rely on any tree-based index. Taking uniform datasets for example, 300 is an optimal point to BBS and ZSearch; while $SKY\text{-}SB$ and $SKY\text{-}TB$ performs 7 times and 150% faster than BBS and ZSearch when the fan-out is fixed at 700. If optimal fan-outs are set individually, $SKY\text{-}SB$ (17 seconds) and $SKY\text{-}TB$ (19 seconds) can achieve 117% and 95% improvement in terms of execution time over ZSearch (37 seconds). As Fig. 11(e) and Fig. 11(f) display, $SKY\text{-}SB$ and $SKY\text{-}TB$ always save significant time in object comparison. Moreover, the performance improvement in $SKY\text{-}SB$ and $SKY\text{-}TB$ is larger in the experimental results over anti-correlated datasets. $SKY\text{-}SB$ and $SKY\text{-}TB$ run around 16 and 24 times faster than BBS; and 7 and 11 times faster than ZSearch. It is interesting that the execution time of $SKY\text{-}SB$ and $SKY\text{-}TB$ changes slightly over anti-correlated datasets because there are only a small number of MBRs discarded in the first step. Most MBRs are fed into the dependent group generation and global skyline computation.

[5]Every intermediate node in the R-tree or ZBtree contains an abstraction of MBR and an array of entries referencing child nodes. Every entry is a 4-byte integer. So, an intermediate node in a page of 4 KBytes can have up to 1014 child nodes in our implementation.

D. *Evaluation on Real Datasets*

We also conducted experiments on real-world datasets from IMDb and Tripadvisor. Table I displays the execution time of the five solutions in seconds. The experimental results indicate that $SKY\text{-}SB$ and $SKY\text{-}TB$ achieve significant speedup against the state-of-the-art skyline solutions over both datasets. This is consistent with our previous experiments; $SKY\text{-}SB$ and $SKY\text{-}TB$ perform much less object comparisons than other solutions in the skyline query evaluation.

TABLE I. THE EXECUTION TIME (IN SECONDS) OVER REAL-WORLD DATASETS.

	$SKY\text{-}SB$	$SKY\text{-}TB$	BBS	ZSearch	SSPL
IMDb	1.45	1.20	1.86	1.76	19.11
Tripadvisor	31.98	31.20	41.16	50.05	59.03

VI. RELATED WORK

A. *Skyline Query*

The skyline query, also known as maximal vector computation [15], was first introduced in database management system by [2]. In their studies, BNL produces skyline objects by comparing every pair of objects. D&C recursively merges skyline objects in small subsets of input datasets by using a merging skyline algorithm. SFS [6] and LESS [10] reduce the cost of dominance test by pre-sorting objects on a particular dimension; no object can be dominated by subsequent objects. Sarma *et al.* [25] proposed a skyline solution (RAND), which can output skyline objects in three iterations. VSkyline utilizes vectorization for optimizing dominance test in SIMD (Single Instruction, Multiple Data) architecture [5]. Zhang *et al.* [29] partitioned the search space into hypercubes, and organized them in a tree structure to avoid unnecessary object comparisons. Lee *et al.* [16] explored an optimal skyline candidate, which partitions the search space in a balanced manner.

Bitmap uses bit-wise operations for object comparison [27]. Index combines the data transformation mechanism and B^+-tree index for evaluating skyline queries [27]. NN finds skyline objects by recursively applying nearest neighbor queries over input datasets [14]. Branch-and-Bound Skyline (BBS) utilizes R-tree to expand the search space by visiting MBRs and objects with minimum $mindist$ [22]. ZBtree was developed to store objects based on their Z-order [18]. Shang and Kitsuregawa proposed a determination and elimination framework for efficient skyline evaluation over anti-correlated datasets [26].

The Skyline with Sorted Positional index Lists (SSPL) that sorts objects in every dimension was developed to efficiently find an object, which can dominate all objects that have not been visited [11]. Then, an existing sorting-based skyline algorithm is applied to visited objects for producing skyline objects. To the best of our knowledge, the early pruning can only be applied to approximate skyline solutions, even if the probability of either discarding real skyline objects or returning false positives is very low (<0.01%).

A skyline ordering solution that relies on a skyline-based partitioning method was developed for size constrained skyline queries [20]. Skyline queries have also been studied in Peer-to-Peer systems and distributed systems [7] [28].

Hose *et al.* proposed a skyline query over regions [12]. The dominance test between two regions requires comparing all objects in the two regions, which differs from our proposed skyline query over MBRs. The dominance test between two MBRs does not access attribute values of objects in our solutions. A distributed Skyline method SkyPlan [24] describes the dominance relationships between two MBRs (M and M') in three categories: (1) M dominates M'; (2) M partially dominates M'; and (3) M and M' are incomparable. The dominance test is performed by using the minimum and maximum values of MBRs, which is also different from our proposed solutions.

Our proposed solutions differ from the aforementioned studies in the following three aspects. First, our method utilizes R-tree index that partitions objects into smaller sets and maintains them in a hierarchical structure. Second, we propose a novel skyline query over MBRs for efficient object elimination. The dominance test between two MBRs can be performed without retrieving any objects in the MBRs. Third, a novel concept of dependent group based on MBRs is proposed for reducing the cost of object comparison. Unlike the independent group of objects in [21], the dependent group of an MBR consists of all MBRs on which the given MBR is dependent. The detailed attributes of objects in the MBRs are not accessed in the dependency test.

B. Skyline Cardinality Estimation

Based on statistical independence and sparseness, Bentley *et al.* reported that the expected size of skylines over datasets of n objects in a d-dimensional space, $\mathbb{L}_{d,n}$, is $O((ln\ \mathrm{n})^{d-1})$ [1]. Buchta provided that $\mathbb{L}_{d,n} = \sum_{k=1}^{n}(-1)^{k+1} \times \binom{n}{k}\frac{1}{k^{d-1}}$ [3].

When duplicate object attributes are considered, Godfrey proposed that $\mathbb{L}_{d,n}$ is equal to the harmonic number $H_{d-1,n}$ [9]. Alternatively, Chaudhuri *et al.* [4] estimated $\mathbb{L}_{d,n}$ and the I/O cost of BNL and SFS algorithms based on a probabilistic model. Zhang *et al.* [30] approached the skyline cardinality by developing a kernel-based method, which is nonparametric and does not rely on any assumptions about data distribution. Shang and Kitsuregawa [26] developed a polynomial estimation method for skyline cardinality on anti-correlated distribution. However, we primarily target cardinality estimation of skyline query over MBRs and dependent groups of MBRs in this paper. All aforementioned methods cannot be applied to the analysis of the two concepts due to the difference in the nature of the problems.

VII. Conclusion

We propose two advanced skyline solutions utilizing novel concepts of skyline query and dependent groups over MBRs for minimizing the cost of dominance test. By using the R tree index of the input dataset, our solutions first search skyline MBRs in the depth-first search manner. Then, the dependent group information enables us to limit the search space; objects are only needed to compare with the ones on which they are dependent. The efficiency and effectiveness of our proposed solutions are verified through extensive experiments.

References

[1] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *J. ACM*, 25(4):536–543, 1978.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, pages 421–430, 2001.

[3] C. Buchta. On the Average Number of Maxima in a Set of Vectors. *Information Processing Letters*, 33(2):63–65, 1989.

[4] S. Chaudhuri, N. N. Dalvi, and R. Kaushik. Robust Cardinality and Cost Estimation for Skyline Operator. In *ICDE*, page 64, 2006.

[5] S.-R. Cho, J. Lee, S. won Hwang, H. Han, and S.-W. Lee. VSkyline: vectorization for efficient skyline computation. *SIGMOD Record*, 39(2):19–26, 2010.

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *ICDE*, pages 717–719, 2003.

[7] B. Cui, L. Chen, L. Xu, H. Lu, G. Song, and Q. Xu. Efficient skyline computation in structured peer-to-peer systems. *TKDE*, 21(7):1059–1072, 2009.

[8] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou. Parallel Distributed Processing of Constrained Skyline Queries by Filtering. In *ICDE*, pages 546–555, 2008.

[9] P. Godfrey. Skyline cardinality for relational processing. In *FoIKS*, pages 78–97, 2004.

[10] P. Godfrey, R. Shipley, and J. Gryz. Maximal Vector Computation in Large Data Sets. In *VLDB*, pages 229–240, 2005.

[11] X. Han, J. Li, D. Yang, and J. Wang. Efficient Skyline Computation on Big Data. *TKDE*, 25(11):2521–2535, 2013.

[12] K. Hose, C. Lemke, and K. Sattler. Processing relaxed skylines in PDMS using distributed data summaries. In *CIKM*, pages 425–434, 2006.

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

[14] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*, pages 275–286, 2002.

[15] H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *J. ACM*, 22(4):469–476, 1975.

[16] J. Lee and S. won Hwang. Bskytree: scalable skyline computation using a balanced pivot selection. In *EDBT*, pages 502–513, 2010.

[17] K. C. K. Lee, W. Lee, B. Zheng, H. Li, and Y. Tian. Z-SKY: an efficient skyline query processing framework based on Z-order. *VLDB J.*, 19(3):333–362, 2010.

[18] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the Skyline in Z Order. In *VLDB*, pages 279–290, 2007.

[19] S. T. Leutenegger, J. M. Edgington, and M. A. López. STR: A Simple and Efficient Algorithm for R-Tree Packing. In *ICDE*, pages 497–506, 1997.

[20] H. Lu, C. S. Jensen, and Z. Zhang. Flexible and Efficient Resolution of Skyline Query Size Constraints. *TKDE*, 23(7):991–1005, 2011.

[21] K. Mullesgaard, J. L. Pedersen, H. Lu, and Y. Zhou. Efficient Skyline Computation in MapReduce. In *EDBT*, 2014.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD Conference*, pages 467–478, 2003.

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

[24] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient execution plans for distributed skyline query processing. In *EDBT*, pages 271–282, 2011.

[25] A. D. Sarma, A. Lall, D. Nanongkai, and J. J. Xu. Randomized Multi-pass Streaming Skyline Algorithms. *PVLDB*, 2(1):85–96, 2009.

[26] H. Shang and M. Kitsuregawa. Skyline Operator on Anti-correlated Distributions. *PVLDB*, 6(9):649–660, 2013.

[27] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient Progressive Skyline Computation. In *VLDB*, pages 301–310, 2001.

[28] J. Zhang, X. Jiang, W.-S. Ku, and X. Qin. Efficient Parallel Skyline Evaluation using MapReduce. *TPDS*, 2015.

[29] S. Zhang, N. Mamoulis, and D. W. Cheung. Scalable skyline computation using object-based space partitioning. In *SIGMOD Conference*, pages 483–494, 2009.

[30] Z. Zhang, Y. Yang, R. Cai, D. Papadias, and A. K. H. Tung. Kernel-based skyline cardinality estimation. In *SIGMOD Conference*, pages 509–522, 2009.