Efficient Computation of G-Skyline Groups

Changping Wang, Chaokun Wang, *Member, IEEE*, Gaoyang Guo, Xiaojun Ye, and Philip S. Yu, *Fellow, IEEE*

Abstract—The skyline of a data point set is made up of the best points in the set, and is very important for multi-criteria decision making. In these years, the skyline problem attracts more and more attention, and many variants of the traditional skyline emerge in the database field. One recent and important variant is group-based skyline, which aims to find the best groups of points in a given set. In this paper, we bring forward an efficient approach, called *minimum dominance search* (MDS), to solve the g-skyline problem, a latest group-based skyline problem. MDS consists of two steps: In the first step, we construct a novel g-skyline support structure, i.e., minimum dominance graph (MDG), which proves to be a minimum g-skyline support structure. In the second step, we search for g-skyline groups based on the MDG through two searching algorithms, and a skyline-combination based optimization strategy is employed to improve these two algorithms. We conduct comprehensive experiments on both synthetic and real-world data sets, and show that our algorithms are orders of magnitude faster than the state-of-the-art in most cases.

Index Terms—Skyline, group skyline, g-skyline, minimum dominance search, minimum dominance graph

1 Introduction

The skyline of a set of multi-dimensional data points is the collection of points, called skyline points, each of which is not dominated by any other point in the set. A point x dominates a point y if and only if x is not worse than y on all the dimensions and is better on at least one dimension. Generally speaking, the skyline of a data point set P is the collection of best points in P, and different skyline points represent various trade-offs between the dimensions. Therefore, skyline is very important for multi-criteria decision making, and it provides all the best options.

Fig. 1 shows a classical skyline example. There are twelve hotels with two attributes: the price and the distance to the beach. The horizontal axis of Fig. 1 corresponds to the price and the vertical axis represents the distance to the beach. Assume that a hotel closer to the beach and cheaper is always better. Thus, we can find the skyline of these hotels is {P1, P2, P4, P7, P9}, and they are the best options if we want to select one hotel.

However, what if a large travel agency needs to select three hotels? A trivial approach is to generate the set of all the combinations of three skyline points. But, this set probably does not include all the necessary options. For instance, if the agency slightly prefers the hotels closer to the beach and cares less about the price, it would like to select the hotels P7, P9, and P11 (P12 is not selected because it is only a bit closer to the beach than P7 but is far more expensive than P7). Clearly, P11 is not a skyline point, and thus this option cannot be provided by the trivial approach. Therefore, we need to find a method which is really effective to this problem.

The recent study on skyline groups [1], [2], [3], [4] aims to solve the above problem. A group is a subset of points, and similar to skyline points, skyline groups are the groups that are not dominated by any other groups. The problem of finding skyline groups is very useful, as it can provide the options when we want to find the best group of data points.

Earlier studies [1], [2], [3] represent each group with a virtual aggregation point whose attribute values are the aggregations over all the points in this group on corresponding attributes. The dominance relationship between two groups is determined by the dominance relationship between the two virtual aggregation points. However, the skyline groups under this definition of group dominance cannot capture all the optimal groups [4]. Liu et al. propose a new group dominance definition, and the skyline groups under this definition are called *g-skyline groups* [4].

Besides the example of travel agency mentioned above, there exist many other applications for g-skyline problem. For example, if we want to build an NBA team of five players from many candidates, we can compute the g-skyline groups of the candidates based on their performance data and select one of these groups according to our preference. Another example is about advertising. When an advertising company needs to select some billboards to serve its ads. It can first compute g-skyline groups of billboards according to both their prices and the coverage of different groups of people, and then the company can find the best group according to some complex strategies. In summary, computing skyline groups is useful for decision making when the goal is to find a group from candidates, since it can filter out lots of candidate groups at a lower cost.

In this paper, we focus on the problem of computing g-skyline and aim to find a more efficient way to search for g-skyline groups. The state-of-the-art approach to finding g-skyline groups is to construct a *directed skyline graph*

- *C. Wang, C. Wang, G. Guo, and X. Ye are with the School of Software, Tsinghua University, Beijing 100084, P.R. China. E-mail: {wang-cp12, ggy16}@mails.tsinghua.edu.cn, {chaokun, yexj}@tsinghua.edu.cn.*
- *P.S. Yu is with the Department of Computer Science, University of Illinois at Chicago, IL 60607. E-mail: psyu@uic.edu.*

Manuscript received 31 Mar. 2017; revised 4 Oct. 2017; accepted 7 Nov. 2017. Date of publication 27 Nov. 2017; date of current version 5 Mar. 2018. (Corresponding author: Chaokun Wang.) Recommended for acceptance by R. Cheng. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2017.2777994

(DSG), and to search for the g-skyline groups based on DSG with two different searching algorithms called PWise and UWise+ [4]. However, we argue that this method has the following shortcomings.

- The structure of DSG contains lots of redundant data points, which are not useful when searching for the g-skyline groups. We think a better data structure should be carefully designed without unnecessary redundancy.
- The constructing algorithm for DSG is not friendly to higher-dimensional data, because it is only optimized for two-dimensional data.
- The searching algorithms based on DSG are not efficient enough, i.e., the pruning strategies could be improved.
- Both PWise and UWise+ do not consider the fact that the set of combinations of skyline points takes an actually high proportion in the g-skyline, and there is a lack of targeted optimization strategies.

In this study, to address the above problems, we propose a new approach, called *minimum dominance search* (MDS), to computing g-skyline groups, which has good performance and is especially efficient on higher-dimensional data. MDS consists of two steps: constructing the *minimum dominance graph* (MDG) and searching for g-skyline groups based on the MDG. We summarize the main contributions of this paper as follows:

- We introduce the concept of g-skyline support structures, and propose a new g-skyline support structure MDG to replace the DSG. We prove that MDG is a minimum g-skyline support structure, i.e., it has no redundancy.
- We propose a special R-tree variant to help construct the MDG for better dealing with multi-dimensional data. In this R-tree variant, a heuristic node splitting strategy based on the sum of attribute values of each point is proposed to optimize the efficiency of both node splitting and searching in our algorithms.
- We propose two searching algorithms, P-MDS and G-MDS, to find g-skyline groups based on MDG. They employ different strategies to generate candidate groups. P-MDS generates new candidate groups through adding single points while G-MDS adds parent groups to generate candidates. The pruning strategies in P-MDS and G-MDS are based on our Verification Theorem. Considering that the g-skyline contains lots of groups which are combinations of skyline points, we bring forward the skyline-combination optimization strategy to make our searching algorithm, P-MDS and G-MDS, more efficient.
- We conduct comprehensive experiments on both synthetic and real-world data sets to compare our algorithms with the state-of-the-art. The experimental results about g-skyline support structures show that the size of MDG is far less than that of DSG, and the construction time of MDG is one or two orders of magnitude less than that of DSG. The results also show that our searching algorithm is more than one order of magnitude faster than the state-of-the-art except for the situation that MDG is very small. Moreover, we confirm that our algorithms, including both MDG constructing and searching algorithms, are more friendly to higher-dimensional data.

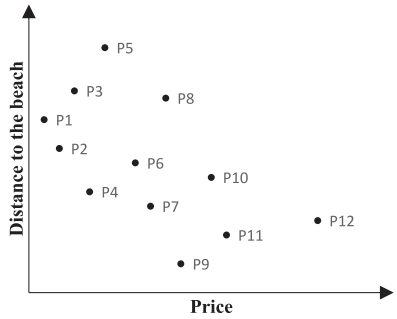


Fig. 1. A skyline example with 12 hotels. {P1, P2, P4, P7, P9} is the skyline of them.

The rest of this paper is organized as follows. The preliminaries of this study are brought forward in Section 2, including several useful concepts related to g-skyline groups. In Section 3, we design a novel g-skyline support structure, propose its construction algorithms, and compare it with DSG. The algorithms to find g-skyline groups based on MDG and an important optimization strategy are given in Section 4. In Sections 5 and 6, we demonstrate the experimental results about the proposed algorithms and present some discussions, respectively. Section 7 reviews the related work. At last, Section 8 concludes this paper.

2 Preliminaries

In this section, we first introduce the concept of skyline from the point level and group level, respectively. Then, we describe the structures of skyline layers and directed skyline graphs briefly.

2.1 Point Dominance and Skyline

The definitions about skyline are first presented in [5]. Without loss of generality, in the rest of this paper we suppose that the lower value is preferred on all dimensions.

Definition 1 (Point dominance). *Let P be a set of d-dimensional data points and a and b be two points in P $(a \neq b)$. Point dominance is a partial order defined on P, and a dominates b, denoted as $a \prec b$, if and only if $a[i] \leq b[i]$ for all i $(1 \leq i \leq d)$ and $a[i] < b[i]$ for at least one i $(1 \leq i \leq d)$. Notation $a \preceq b$ means a dominates or equals to b.*

For example, for the data points in Fig. 1, we have P2$\prec$P8, P4$\prec$P6, P9$\prec$P11, and so on. It is obvious that for two points a and b, $a \prec b$ means a is better than b.

Definition 2 (Skyline). *Let P be a set of d-dimensional data points and a be one point in P. a is a skyline point if and only if there exists no point in P dominating a. The skyline of P, denoted as $skyline(P)$, consists of all the skyline points.*

For example, the skyline of the data set shown in Fig. 1 consists of points P1, P2, P4, P7 and P9, because all of these points are not dominated by any points. The skyline of a data point set provides the best options if we want to select one best point in the set.

2.2 Group Dominance and G-Skyline

The skyline points are Pareto optimal in the perspective of economics. However, if we want to find Pareto optimal groups of points, we need to extend the definition of the skyline point. Then, we give the definitions about g-skyline, which is a group-based skyline.

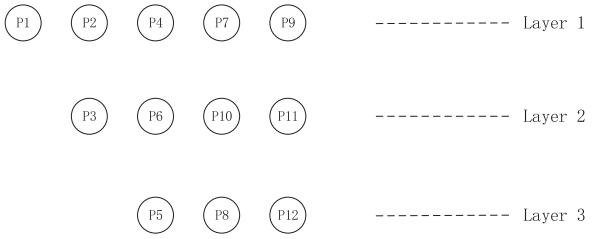


Fig. 2. Skyline layers, $l = 3$.

Definition 3 (Group dominance). *Let P be a set of d-dimensional data points, and $A = \{a_1, a_2, \ldots, a_l\}$ and $B = \{b_1, b_2, \ldots, b_l\}$ be two groups (subsets) with l points of P. A g-dominates B, denoted as $A \prec_g B$, if and only if there exist two permutations, $A = \{a_{p1}, a_{p2}, \ldots, a_{pl}\}$ and $B = \{b_{p1}, b_{p2}, \ldots, b_{pl}\}$, such that $a_{pi} \preceq b_{pi}$ for all i $(1 \leq i \leq l)$ and $a_{pi} \prec b_{pi}$ for at least one i.*

For the data set in Fig. 1, we have {P1, P2, P10} $\prec_g$ {P5, P8, P10}, because P1 $\prec$ P8, P2 $\prec$ P5, and P10 $\preceq$ P10. Based on the definition of group dominance, we can extend the concept of skyline to group-based skyline naturally.

Definition 4 (G-Skyline). *Let P be a set of d-dimensional data points and A be an l-point group of P. A is a g-skyline group if and only if there exists no l-point group g-dominating A. The l-point g-skyline of P consists of all the g-skyline groups with l points.*

In Fig. 1, {P1, P2, P4} is a 3-point g-skyline group. Because P1, P2, and P4 are all skyline points, it is easy to verify that this group cannot be g-dominated by any other 3-point groups. Although P11 is not a skyline point, {P7, P9, P11} is still a g-skyline group of 3 points. P11 is only dominated by P9 which is in the group already, so we can verify that no other groups can g-dominate this group.

2.3 Skyline Layers and Directed Skyline Graph

The work of [4] brings forward two structures, i.e., skyline layers and directed skyline graph (DSG), to represent data points and their dominance relationships, based on which the algorithms are proposed to find g-skylines. In order to compare our proposed structure with skyline layers and DSG theoretically, we briefly introduce their definitions for the self-containment of this paper, and more details can be found in [4].

Definition 5 (Skyline layers). *Let P be a set of d-dimensional data points. The first skyline layer $layer_1$ is defined as the set of skyline points of P, i.e., $layer_1 = skyline(P)$. The ith skyline layer $layer_i$ is defined as the set of skyline points of P excluding the points in the first $(i-1)$ skyline layers, i.e., $layer_i = skyline(P \backslash \bigcup_{j=1}^{i-1} layer_j)$.*

Fig. 2 shows the skyline layers of the data set shown in Fig. 1. There are three skyline layers in this example. The first skyline layer consists of P1, P2, P4, P7 and P9, which are all the skyline points of the data set. The second skyline layer includes points P3, P6, P10 and P11, and the rest points form the third layer.

Definition 6 (Directed skyline graph). *Let P be a set of d-dimensional data points. The directed skyline graph (DSG) of P is a directed acyclic graph, where each node represents a point in P and each edge represents a dominance relationship between two points. Only the points in the first l skyline layers and the dominance relationships between them are in the DSG.*

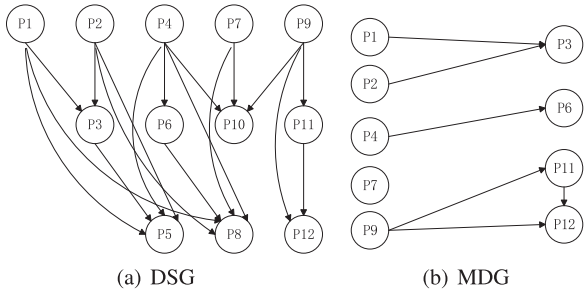


Fig. 3. DSG and MDG, $l = 3$.

For instance, Fig. 3a is the directed skyline graph based on the skyline layers in Fig. 2, where $l = 3$.

3 Minimum Dominance Graph

In this section, we focus on the first step of MDS, i.e., constructing MDG. We first introduce a theorem to support the verification of g-skyline. Next, the definition of MDG is given in Section 3.1. Then, in Section 3.2, we propose two algorithms to construct MDG. Finally, we compare our MDG with DSG theoretically in Section 3.3.

Theorem 1 (Verification theorem). *Let P be a set of d-dimensional data points and A be an l-point group of P. A is a g-skyline group if and only if for each point $a \in A$, there exists no point outside A that dominates a.*

Proof. ($\Leftarrow$) Assume for each $a \in A$, there exists no point outside A that dominates a. Let A be not a g-skyline group. Then, there exists a group of l points, denoted as B, g-dominating A. According to Definition 3, there are two permutations $A = \{a_{p1}, a_{p2}, \ldots, a_{pl}\}$ and $B = \{b_{p1}, b_{p2}, \ldots, b_{pl}\}$, such that $b_{pi} \preceq a_{pi}$ for all i $(1 \leq i \leq l)$ and $b_{pi} \prec a_{pi}$ for at least one i.

Without loss of generality, let $b_{pm} \prec a_{pm}$, and then b_{pm} must be in A, since there exists no point outside A that dominates a_{pm}. Supposing $b_{pm} = a_{pn}$, then b_{pn} must dominate a_{pn} because $b_{pn} \neq b_{pm} = a_{pn}$. Therefore, $b_{pn} \prec a_{pm}$ for the transitivity of dominance. Further, b_{pn} must be in A, for the same reason as b_{pm}. Repeating this procedure, we can prove that all the points in B dominate a_{pm}, i.e., there are at least l points dominating a_{pm}. Considering the size of A is l, there must exist at least one point outside A that dominates a_{pm}, a contradiction.

Therefore, A is a g-skyline group.

($\Rightarrow$) Conversely, assume A is a g-skyline group. Let point $a \in A$, and there exists a point b dominating a, $b \notin A$. Through replacing a with b in A, we can get a new group A'. It is obvious that $A' \prec_g A$, which contradicts that A is a g-skyline group. Hence, for each point $a \in A$, there exists no point outside A that dominates a. □

Theorem 1 offers us an efficient way to verify whether a point group is a g-skyline group. Given a point group A, for each point $a \in A$, if we can verify that the points which dominate a are all in the group A, we can be sure that A is a g-skyline group, and vice versa. For instance, in Fig. 1, {P2, P4, P6} is a 3-point g-skyline group. Both P2 and P4 are skyline points so that no points can dominate them, and P6 is only dominated by P4, which is in this group itself. However, {P7, P9, P12} is not a g-skyline group because P11 $\prec$ P12 but P11 is not in the group. In fact, we can easily find that {P7, P9, P11} $\prec_g$ {P7, P9, P12}.

Algorithm 1. Basic MDG Construction Algorithm

Require:
The data point set P, the size of each g-skyline group l.
Ensure:
G /* The minimum dominance graph of P. */
1: Initialize $G = (V, E)$ with an empty graph.
2: Sort the data points in P in the ascending order of the sum on all dimensions.
3: **for** i from 0 to $l-1$ **do**
4: $B[i] \leftarrow \emptyset$ /*$B[i]$ consists of all the points which have i parents*/
5: **end for**
6: **for all** $p \in P$ **do**
7: $flag \leftarrow$ TRUE /*flag indicates if p has less than l parents*/
8: $p.parents \leftarrow \emptyset$
9: **for** i from $l-1$ to 0 **do**
10: **if** $flag$ is FALSE **then**
11: break
12: **end if**
13: **for all** $p' \in B[i]$ **do**
14: **if** $p' \prec p$ **then**
15: $p.parents \leftarrow p.parents \cup \{p'\} \cup p'.parents$
16: **if** $|p.parents| \geq l$ **then**
17: $flag \leftarrow$ FALSE
18: break
19: **end if**
20: **end if**
21: **end for**
22: **end for**
23: **if** $flag$ is TRUE **then**
24: $V \leftarrow V \cup \{p\}$
25: **for all** $p' \in p.parents$ **do**
26: Add an edge into E from p' to p
27: **end for**
28: $B[|p.parents|] \leftarrow B[|p.parents|] \cup \{p\}$
29: **end if**
30: **end for**
31: **return** G

3.1 Definition of MDG

In this section, we first deduce the following corollary from Theorem 1, which is the basis of MDG. Then, we present the definition of DMG.

Corollary 1. *Let P be a set of d-dimensional data points and A be an l-point group of P. If A is a g-skyline group, then for each point $a \in A$, there exist less than l points which dominate a in P.*

Proof. According to Theorem 1, for each point $a \in A$, the points which dominate it must be in A. Since A has l points and $a \not\prec a$, there exist less than l points which dominate a. □

Based on Corollary 1, we introduce a data structure, called minimum dominance graph (MDG), to help find the g-skyline groups in a given data point set.

Definition 7 (Minimum dominance graph). *Let P be a set of d-dimensional data points. The minimum dominance graph (MDG) of P is a directed acyclic graph, where each node represents a point in P and each edge represents a dominance relationship between two points. Only the points which are dominated by less than l points in P and the dominance relationships between them are in the MDG.*

Fig. 3b illustrates the MDG of the data set in Fig. 1. In an MDG, if there is an edge from a to b, i.e., a dominates b, we consider a as b's parent and b as a's child. In the MDG shown in Fig. 3b, for example, P1 and P2 are the parents of P3, and P3 is the child of these two points. Notice that the MDG is not based on the skyline layers.

Our searching algorithms for g-skyline groups are based on the MDG, i.e., we only consider the points in MDG when constructing a g-skyline group. The effectiveness and efficiency of MDG are discussed in Section 3.3.

3.2 Construction of MDG

We first introduce a simple but effective way to construct MDG, called the basic MDG construction algorithm. Then, we present how to improve this algorithm with the R-tree index.

3.2.1 Basic MDG Construction Algorithm

The basic MDG construction algorithm is designed based on Definition 7, i.e., the algorithm tries to find all the points in P that are dominated by less than l points. For each point in P, this algorithm runs dominance checks between it and all the points in the current MDG to determine whether it should be added into the MDG. Its details are shown in Algorithm 1.

First, we initialize the MDG with an empty graph $G = (V, E)$ (Line 1), and sort the points in P in the ascending order of the sum on all dimensions (Line 2). After sorting, it is guaranteed that each point cannot be dominated by the points after it. Thus, for any point, we only need to find out how many points before the point can dominate it to judge whether it should be in the MDG.

Next, we initialize l buckets denoted as $B[i]$, and $B[i]$ consists of the points in V (the node set corresponding to MDG) with i parents (Line 4). With these buckets, we can easily keep points in V in a descending order of the number of their parents.

Then, for each point p, we need to find how many points in V dominate it. For all the buckets (in a reverse order), we traverse all the points in it, and if we find a point $p' \prec p$, we add not only p' but also its parents to the parent set (without duplication) of p according to the transitivity of dominance (Line 15). If $p.parents$ has more than $l-1$ points, the traversal should be stopped (Lines 18 and 11).

Last, if $p.parents$ has less than l points after the traversal, we add p into V, add edges from points in $p.parents$ to p into the edge set corresponding to the MDG (E), and add p to one bucket according to the size of $p.parents$ (Lines 24–28).

Example 1. We take the point set in Fig. 1 and the corresponding MDG in Fig. 3b as an example to show how Algorithm 1 works ($l = 3$). At first, we sort these points with the ascending order of the sum on all dimensions, and the sorted result is (P4, P2, P9, P1, P7, P6, P11, P3, P10, P5, P8, P12). Next, points P4, P2, P9, P1 and P7 are added into both the MDG and the bucket $B[0]$, since they are all skyline points. P6 and P11 are added into the MDG and the bucket $B[1]$, because they are both dominated by one point. We add P3 into the MDG as well as the bucket $B[2]$ for the similar reason. Then, when considering the point P10, we find that it cannot be in the MDG since it is dominated by three points (P4, P7 and P9). P5 is dominated by P3 which is in $B[2]$, i.e., P5 is dominated by at least three points (P3 and its parents P1 and P2), and thus

we do not add P5 into the MDG. P8 is abandoned for the similar reason, since it has four parents. At last, we add P12 into the MDG and the bucket $B[2]$.

3.2.2 MDG Construction with R-Tree Algorithm

Through analyzing Algorithm 1, we can find that the most time-consuming part is to test the dominance relationships between the current point and the points in the MDG already. Thus, in this part we propose a special variant of R-tree, with a heuristic node splitting strategy, to improve the basic MDG construction algorithm.

The R-tree, proposed by Antonin Guttman in 1984 [6], is a tree data structure used for indexing multi-dimensional data. Similar to the B-tree, the R-tree is a balanced search tree. The spatial objects, which are data points in our problem, are stored in or linked to the leaf nodes. Each intermediate node is represented as a minimum bounding rectangle (MBR), which is the minimum rectangle where the MBRs of its child nodes lie.

Based on the R-tree, we first propose an improved algorithm (Algorithm 2) for MDG construction, and then present the heuristic node splitting strategy.

The Improved Algorithm with R-Tree. There are two major differences between Algorithms 2 and 1. The first difference is that, the set $parents$ is obtained through searching the R-tree with the function $SearchForParents(p, R, l)$ (Lines 13–15) in Algorithm 2, while Algorithm 1 gets this set by traversing all the data points in $V - L$. The function $SearchForParents(p, R, l)$ is based on the search operation on R-tree, and it actually searches the points in the area, denoted as $Area_p$. For each dimension, the range of $Area_p$ is from the minimum of this dimension to the value of p on this dimension. As we know, for a search operation on R-tree, we always have to search more than one branch. However, in the function $SearchForParents(p, R, l)$, if the results in first branches already have more than $l - 1$ points, we can stop searching the following branches, because it is easy to judge p should not be in the MDG in this situation.

The second difference lies in that points whose parents are less than l are inserted into the R-tree index in Algorithm 2 (Line 24). Notice that this is the only situation where we modify the R-tree index.

Heuristic Node Splitting. The strategy of node splitting is very important for the efficiency of R-tree, and there exist various node splitting strategies for R-tree and its variants. Our R-tree variant uses a heuristic node splitting strategy, which is inspired by the Manhattan-norm based sorting in [7].

For a leaf node to split, we calculate the sum on all dimensions for each data point in it, and half of the points with lower sum form a new node while the rest points form the other new node. For an internal node, we use the centroid of its MBR to represent it and follow the steps for leaf nodes. This heuristic node splitting strategy has two advantages. First, it is much faster to determine how a node splits with this strategy than other existing strategies. Second, this strategy is very friendly to the queries used in Algorithm 2. For a given query point p, a point in MDG with very low sum of values on all dimensions is likely to be in $Area_p$, and vice versa. Thus, our heuristic node splitting strategy could make the search operations used in Algorithm 2 more efficient.

Algorithm 2. MDG Construction with R-Tree Algorithm

Require:
The data point set P, the size of each g-skyline group l.
Ensure:
G /* The minimum dominance graph of P. */
1: Initialize $G = (V, E)$ with an empty graph.
2: Initialize R with an empty R-tree.
3: Sort the data points in P in the ascending order of the sum on all dimensions.
4: $L \leftarrow \emptyset$ /*L consists of all the points which have $l - 1$ parents*/
5: **for all** $p \in P$ **do**
6: $flag \leftarrow$ TRUE
7: **for all** $p' \in L$ **do**
8: **if** $p' \prec p$ **then**
9: $flag \leftarrow$ FALSE
10: break
11: **end if**
12: **end for**
13: **if** $flag$ is True **then**
14: $parents = SearchForParents(p, R, l)$
15: **end if**
16: **if** $|parents| < l$ **then**
17: $V \leftarrow V \cup \{p\}$
18: **for all** $p' \in parents$ **do**
19: Add an edge into E from p' to p
20: **end for**
21: **if** $|parents| = l - 1$ **then**
22: $L = L \cup \{p\}$
23: **end if**
24: $InsertPoint(p, R)$
25: **end if**
26: **end for**
27: **return** G

Example 2. Fig. 4 shows an example of Algorithm 2 with the first six points (P1, P2, P3, P4, P5, P6) in Fig. 1. Supposing the points from P1 to P5 are already checked, we find that P1, P2, P3 and P4 are in MDG and index them through an R-tree with one node (the blue solid box in Fig. 4a). When considering P6, we search the R-tree with the searching area of P6 (the red dashed box in Fig. 4a), and find only P4 in this area, i.e., P4 $\prec$ P6. Thus, P6 should be added into MDG and indexed through the R-tree. Supposing the maximum number of entries hosted in one node is four, we need to split this node which has five points, i.e., (P1, P2, P3, P4, P6). With our heuristic node splitting strategy, the points with lower sum of attribute values, i.e., P1, P2 and P4, form one new node, and the rest points, P3 and P6, form the other new node (blue solid boxes in Fig. 4b).

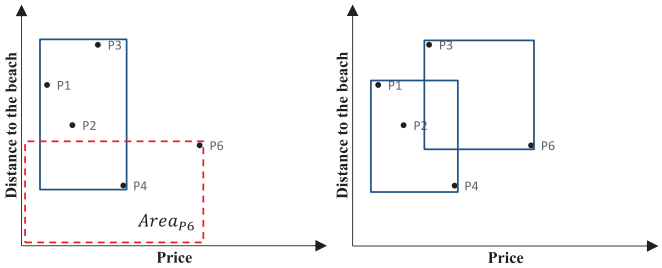
Comparing with Algorithm 1, Algorithm 2 improves the time cost of searching for parents with the help of R-tree index, and we will discuss the time complexity of these two algorithms in the next section.

3.3 Comparisons between MDG and DSG

In this section, we compare our MDG with DSG, and present the advantages of MDG.

3.3.1 The Size of Graph

We first discuss the sizes of these two graph structures.



(a) Before the splitting (b) After the splitting

Fig. 4. An example of Algorithm 2. Blue solid boxes represent nodes of R-tree, and the red dashed box is the searching area for P6.

Lemma 1. *Let P be a set of d-dimensional data points. Each point in the ith skyline layer w.r.t. P must be dominated by no less than $i-1$ points.*

Proof. According to Definition 2, for each point in the ith skyline layer, there exists at least one point dominating it in each of the first $i-1$ skyline layers, and thus it must be dominated by no less than $i-1$ points. □

Theorem 2. *Let P be a set of d-dimensional data points. Given the g-skyline group size l, the size of the MDG is not larger than that of the DSG.*

Proof. For any point x in the MDG, it has no more than $l-1$ parents on the basis of Definition 7. According to Lemma 1, x must be in the first l skyline layers of P, otherwise it should be dominated by more than $l-1$ points. Thus, x is in the DSG of P. As a result, each point in the MDG is also in the DSG, and then the size of the MDG is not larger than that of the DSG. □

Fig. 3 shows an example of the difference of the size of MDG and DSG corresponding to the data point set in Fig. 1. In this example, we can see that there are 12 points in the DSG and only 9 points in the MDG. Points P10, P5 and P8 are in the DSG but not in the MDG. The reason is that they are all dominated by more than two points, i.e., they cannot be in a g-skyline group of three points. From this example, we can intuitively find that the MDG is smaller than the DSG through pruning those points that are in the first l layers but are dominated by more than $l-1$ points.

We have proved that the size of MDG is not larger than that of DSG, but we still have a question whether the size of MDG could be further reduced.

To answer this question, we first introduce the definition of g-skyline support structure.

Definition 8 (G-skyline support structure). *Let P be a set of d-dimensional data points and l be the size of each g-skyline group. A g-skyline support structure is a data structure consisting of a subset of P, denoted as V, and the relationships between points in V, such that V must contain all the points that are in l-point g-skyline groups. If V contains no such point that cannot be in any l-point g-skyline groups, we call the corresponding structure the minimum g-skyline support structure.*

It is obvious that both MDG and DSG are g-skyline support structures, and thus they can be used to find g-skyline groups. Clearly, P and the relationships between points in it form a maximum g-skyline support structure, and we have $V_{MDG} \subseteq V_{DSG} \subseteq P$. Moreover, we find that the MDG is a minimum g-skyline support structure. Before proving it, we need to introduce a related definition and a lemma.

Definition 9 (Parent group). *Let P be a set of d-dimensional data points and l be the size of each g-skyline group. In the corresponding MDG, a point p and its parents form the parent group of p, denoted as $g(p)$.*

Lemma 2. *Let P be a set of d-dimensional data points and l be the size of each g-skyline group. Given a point p in the corresponding MDG and its parent group $g(p)$, for each point $p' \in g(p)$, we have $g(p') \subseteq g(p)$.*

Proof. For any point $p'' \in g(p')$, we have $p'' \preceq p'$ according to Definition 9. In the same way, $p' \preceq p$. Then $p'' \preceq p$, and thus $p'' \in g(p)$. Therefore, $g(p') \subseteq g(p)$. □

Theorem 3. *Let P be a set of d-dimensional data points and l be the size of each g-skyline group. The MDG corresponding to P is a minimum g-skyline support structure.*

Proof. We first prove that the MDG is a g-skyline support structure. For each point that is in an l-point g-skyline group, there exist less than l points that dominate it according to Corollary 1, which means it should be in the MDG.

Next, we prove that the MDG does not contain any point that is not in any l-point g-skyline groups. By contradiction, we assume there exists one point p in the MDG that is not in any l-point g-skyline groups. We select an arbitrary l-point g-skyline group gr. Note that there is at least one l-point g-skyline group, since we can repeat picking one skyline point and removing it from the data point set for l times to construct an l-point g-skyline group.

It is easy to verify that $gr \cup g(p)$ is a g-skyline group according to Theorem 1 and Lemma 2. Moreover, we have $|gr \cup g(p)| > |gr| = l$, since $p \in g(p)$ and $p \notin gr$.

Because the MDG is a directed acyclic graph, it is obvious that for the group $gr - g(p)$, there exists at least one point $p'' \in gr - g(p)$ such that none of p'''s children is in the group $gr - g(p)$ (otherwise there exists a cycle in the MDG). Also, none of p'''s children is in $g(p)$, otherwise we have $p'' \in g(p)$ according to Lemma 2, which conflicts with that $p'' \in gr - g(p)$. We remove the point p'' from gr, and the group $gr \cup g(p)$ is still a g-skyline group since p'' is not the parent of any points remained in $gr \cup g(p)$. By repeating this removing procedure enough times, we can get an l-point g-skyline group $gr \cup g(p)$. Thus, p is in an l-point g-skyline group now, which contradicts our assumption. □

Theorem 3 guarantees that searching for g-skyline groups on MDG is not only effective but also the most efficient.

3.3.2 The Time Complexity

Here, we analyze the time complexity of constructing MDG and DSG. Suppose the size of P is n, and the sizes of MDG and DSG corresponding to P are n_m and n_d, respectively.

The construction of MDG can be divided into two parts. In the first part, sorting all points takes $O(n \times \log n)$ time. In the second part, for each point $p \in P$, we find out the points in the MDG which dominate p. For the basic algorithm shown in Algorithm 1, the time complexity of this part is $O(n \times n_m)$, since there are almost $n \times n_m$ dominance checks. When the R-tree variant is utilized, this part needs only $O(n \times \log n_m)$ time. Thus, the total time complexity of Algorithm 1 is $O(n \times \log n + n \times n_m)$, and that of Algorithm 2 is $O(n \times \log n + n \times \log n_m)$.

Calculating in a similar way, the total time complexity of constructing DSG is $O(n \times \log n + n_d \times \log l + n_d^2)$ for two-

dimensional data and $O(n \times \log n + n \times n_d)$ for higher dimensional data.

Theorem 2 indicates that $n_m \leq n_d$. We can find that for higher dimensional data, Algorithms 1 and 2 both have a lower time complexity than the DSG construction algorithm. When the dimension of data becomes two, the time complexity of DSG construction algorithm decreases significantly. However, Algorithm 2 is still comparable in this case, and the experimental results in Section 5 show that it is faster than the DSG construction algorithm when the data points are two-dimensional.

The construction of DSG heavily depends on the skyline layers. However, it seems that the skyline layers are the shackles of DSG. On the one hand, computing skyline layers takes much time on higher-dimensional data. It is efficient to compute skyline layers for two-dimensional data because of the monotonic property of skyline points in two-dimensional space, i.e., if the skyline points are sorted with increasing x-coordinate, their y-coordinates must decrease monotonically, since they cannot dominate each other. This applies to the points in each skyline layer. Therefore, if we refer the point with minimum y-coordinate of a skyline layer as the tail point of this layer, we can easily judge whether a point can be dominated by the points in a specific skyline layer through running dominance check between this point and the tail point of the layer. But the monotonic property does not hold for higher-dimensional data, and thus computing skyline layers on these data costs much more time than that on two-dimensional data. On the other hand, skyline layers bring a lot of points which are not necessary for g-skyline groups, and thus lots of efforts are wasted when constructing DSG based on skyline layers. Our proposed MDG abandons the skyline layers, and becomes a minimum g-skyline support structure with efficient construction algorithms.

4 Finding G-Skyline Groups Based on MDG

In the previous section, we have proved that given the data set P and the size of each g-skyline group l, the corresponding MDG contains all the points which are in l-point g-skyline groups. In this section, we discuss how to find all the g-skyline groups based on the MDG, which is the second step of MDS. We first present two searching algorithms which aim to find all the solutions. The difference between the two algorithms is the strategy for generating new candidate groups. Then, we present a practical optimization based on skyline-combination for these two algorithms.

4.1 Searching Algorithm Based on Single Points

The first searching algorithm, called P-MDS, is shown in Algorithm 3. In this algorithm, we generate new candidate groups by adding proper single points in the current candidate group.

First, we initialize the result set R and the current group gr with empty sets (Lines 1-2 of Algorithm 3). Next, we sort all the points in MDG with the ascending order of the number of their parents (Line 3). Therefore, we can avoid generating duplicated candidate groups if we always add the points that are after the last added point in the current group. Moreover, the skyline points are in the front of sorted points. Then, we start a depth-first-search (DFS) through adding each skyline point into the empty current group (Lines 4-9). Last, the result set R can be obtained after the DFS ends.

The DFS is implemented based on $SearchSinglePoint()$, a recursive function shown in Algorithm 4. If the current group gr contains l points, we add it to the result set as an l-point g-skyline group without verification and the function returns (Lines 1-4 of Algorithm 4). If gr has less than l points, we will try to add a candidate point p to gr and recursively call the function $SearchSinglePoint()$ (Lines 5-10). More precisely, for each p after p_{last} ($p \in$ MDG), we append p at the end of gr if all p' parents have already been in gr (Line 6). This is our pruning strategy for P-MDS.

Algorithm 3. Searching for G-Skyline Groups with MDG Based on Single Points (P-MDS)

Require:
The data point set P, the size of each g-skyline group l, the corresponding MDG.
Ensure:
R /* The result set of l-point g-skyline groups*/
1: $R \leftarrow \emptyset$
2: $gr \leftarrow \emptyset$
3: Sort points in the MDG with the ascending order of the number of their parents.
4: **for all** $p \in$ MDG **do**
5: **if** p is a skyline point of P **then**
6: $gr \leftarrow \{p\}$
7: $SearchSinglePoint(gr, p, R)$
8: **end if**
9: **end for**
10: **return** R

The pruning strategy, called the Parent Pruning, guarantees that the current group gr is always a g-skyline group according to Theorem 1, and that is why we do not check whether gr is an l-point g-skyline group when it contains l points. Moreover, the Parent Pruning is in harmony with our sorting strategy. In the sorted points, a parent node must be in front of its children. Therefore, when we add points in the ascending order of the number of their parents and search with our Parent Pruning strategy, there will be no missing or duplicated results.

Algorithm 4. SearchSinglePoint

Require:
The current group gr, the last point p_{last} in gr, the result set R.
1: **if** $|gr| = l$ **then**
2: $R \leftarrow R \cup \{gr\}$
3: **return**
4: **end if**
5: **for all** $p \in$ MDG & p is after p_{last} **do**
6: **if** $(g(p) \backslash p) \subseteq gr$ **then**
7: $gr \leftarrow gr \cup \{p\}$
8: $SearchSinglePoint(gr, p, R)$
9: **end if**
10: **end for**

Our P-MDS shares a similar basic idea with the PWise algorithm when generating new candidate groups. However, there are two major differences between them. First, we propose the Parent Pruning to replace all the pruning strategies in PWise. The discussion above indicates that the Parent Pruning achieves the highest pruning ability, i.e., there is no need to verify candidate results. Second, P-MDS employs

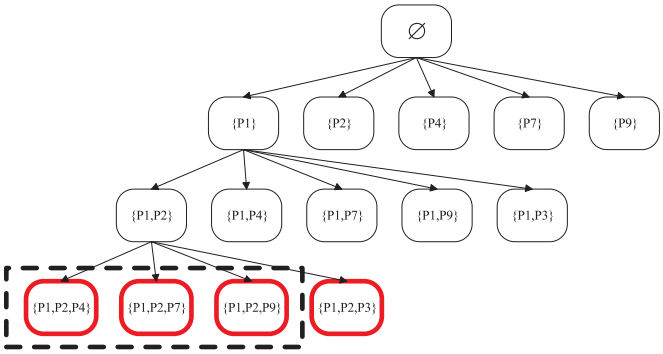


Fig. 5. An example of P-MDS, $l = 3$. The groups with red bold borders are parts of the result. Those in the dashed box can be skipped if the skyline-combination optimization strategy is used.

depth-first-search while PWise uses breadth-first-search. This difference keeps P-MDS from being bothered by the space cost when there are too many temporal candidate groups.

Example 3. Fig. 5 shows how the P-MDS algorithm works w.r.t. the MDG in Fig. 3b. We first sort the points with the ascending order of the number of their parents. Without loss of generality, the result is (P1, P2, P4, P7, P9, P6, P11, P3, P12). The first candidate group is $\emptyset$, and we can add one of the 5 skyline points into it. Suppose we add point P1 and the current group becomes {P1}. For this group, we can add P2, P4, P7, and P9 into it to generate new groups, because they are all skyline points after P1. Besides the skyline points P4, P7 and P9, we can also add the point P3 into the group {P1, P2}, for P3's parents are P1 and P2, which are both already in the group. During this DFS process, all the candidate groups with three points are the g-skyline groups we want to find.

4.2 Searching Algorithm Based on Parent Groups

In this section, we present the other algorithm to find g-skyline groups based on MDG, called G-MDS (Algorithms 5 and 6). It has the following three differences compared with the P-MDS algorithm.

- The most important difference is that their strategies to generate new candidate groups are different. G-MDS generates new candidate groups by adding the parent group of candidate points (Line 5 of Algorithm 5 and Line 10 of Algorithm 6), rather than the single points. According to Theorem 1, this strategy ensures that the current group is always a g-skyline group, which is the guarantee of the correctness of this algorithm. Thus, it is the same as P-MDS that candidate groups with l points must be l-point g-skyline groups (Lines 1-4 of Algorithm 5).
- In G-MDS, the current group may have more than l points, since we add multiple points at one time. Thus, these groups can be pruned (Lines 5-7 of Algorithm 6).
- In the G-MDS algorithm, points in MDG are sorted in the descending order of the number of their parents (Line 3 of Algorithm 5), while the opposite order is used in P-MDS. This order tends to generate larger current groups at first, and thus, with the condition that the current group must have no more than l points, more candidate groups can be pruned.

Notice that G-MDS also employs DFS for the same reason as P-MDS, which is different from UWise+ (a similar searching algorithm based on parent groups).

Algorithm 5. Searching for G-Skyline Groups with MDG Based on Parent Groups (G-MDS)

Require:
The data point set P, the size of each g-skyline group l, the corresponding MDG.
Ensure:
R /* The result set of l-point g-skyline groups*/
1: $R \leftarrow \emptyset$
2: $gr \leftarrow \emptyset$
3: Sort points in the MDG with the descending order of the number of their parents.
4: **for all** $p \in$ MDG **do**
5: $gr \leftarrow g(p)$ /*$g(p)$ is the parent group of p*/
6: $SearchParentGroup(gr, p, R)$
7: **end for**
8: **return** R

Example 4. Fig. 6 is an example of the G-MDS algorithm w.r.t. the MDG in Fig. 3b. After we sort the points in the MDG with the descending order of the number of their parents, the order of these points is (P12, P3, P11, P6, P9, P7, P4, P2, P1). It is the same as P-MDS that the first candidate group is $\emptyset$. The difference is that we add the parent group of each point to generate new groups. For the groups g(P12) and g(P3), they both contain 3 points, and then we get two result groups. However, for g(P11), the current group has only two points, which means we still need to add parent groups into it. We can add the parent groups of P6, P7, P4, P2 and P1, since these points are not contained in the current group now. If we add P6, the current group will contain 4 points, so that we should abandon this group. If we add the other 4 points, we can obtain four 3-point g-skyline groups that we are looking for.

Algorithm 6. SearchParentGroup

Require:
The current group gr, the last point p_{last} in gr, the result set R.
1: **if** $|gr| = l$ **then**
2: $R \leftarrow R \cup \{gr\}$
3: **return**
4: **end if**
5: **if** $|gr| > l$ **then**
6: **return**
7: **end if**
8: **for all** $p \in$ MDG & p is after p_{last} **do**
9: **if** $p \notin gr$ **then**
10: $gr \leftarrow gr \cup g(p)$ /*$g(p)$ is the parent group of p*/
11: $SearchParentGroup(gr, p, R)$
12: **end if**
13: **end for**

We can find that the core ideas of P-MDS and G-MDS both come from Theorem 1. P-MDS only adds the points which cannot be dominated by the points outside the current group, and G-MDS adds the candidate point (p is after p_{last} and $p \notin gr$) as well as its parents. They both ensure that the condition of Theorem 1 holds in current groups. Therefore, we do not need to check whether a current group with l points is a g-skyline group in both algorithms, which makes them much efficient.

4.3 Optimizations Based on Skyline-Combination

In this section, we first present a property of g-skyline groups, and then show how it could help optimize the algorithms we proposed.

Property 1 (Skyline-combination). *Let P be a set of d-dimensional data points and $skyline(P) = \{s_1, s_2, \ldots, s_n\}$. Each combination of k $(k \leq n)$ points in $skyline(P)$ is a k-point g-skyline group.*

The property of *skyline-combination* is easy to be proved, and it can help us make P-MDS and G-MDS more efficient. We can compute the combinations of skyline points to generate candidate groups directly, without the DFS process. Thus, the computation costs caused by the recursion and conditional statements during searching could be decreased.

For P-MDS, we generate l-point g-skyline groups through computing the combinations of l skyline points and avoid the corresponding cost during the DFS process. Specifically, we generate these combinations and add them to the result set R before starting searching (Line 4 of Algorithm 3). Also, we add another pruning strategy, i.e.,$\neg$ $(p \in skyline(P)$ & $|gr| = l - 1)$, in Line 6 of Algorithm 4 to form a conjunctive expression, so that if p is a skyline point and gr already contains $l - 1$ points, p would not be added into gr. The new pruning condition avoids generating the groups which are the combinations of l skyline points. Due to the space limitation, the pseudo code of the optimized P-MDS is not presented, and the same for the following optimization on G-MDS.

Example 5. In Fig. 5, the groups in the dashed box are some examples of groups made up of only skyline points. We can intuitively find that these groups have a high proportion of the result groups, i.e., 3/4 in this figure. Fortunately, these groups will not be generated through searching in the optimized P-MDS algorithm.

For G-MDS, we can employ a further optimization. Besides generating the groups made up of only skyline points directly, we also consider a more general scenario. Suppose the current group gr contains $l - m$ $(m > 0)$ points and the next point p to be considered adding into gr is a skyline point, and we know that the points after p are all skyline points due to the sorting process. Then, we can select any m points which are not already in gr from these skyline points to generate l-point g-skyline groups directly.

Example 6. We take Fig. 6 as an example to show how this further optimization works. Suppose the current group gr is {P9, P11}, and the next point to be considered adding into gr is P9. We find that P9 is a skyline point. Considering the points after P9, i.e., {P7, P4, P2, P1}, are all skyline points, we can select any $m = l - |gr| = 1$ point which is not in gr, and directly generate g-skyline groups through adding them into gr separately. Thus, we obtain the g-skyline groups in the dashed box without the DFS process.

5 Experiments

This section reports the experimental results of MDS. For a complete evaluation, we employ three synthetic data sets [5], i.e., **Indep**, **Corr** and **Anti**, as well as a real-world data set **NBA** [4].

Actually, each of the synthetic data sets is a family of data sets. We generate each data set of a given family by following the steps in [5] with the given number of points n, number of dimensions d and some hyper parameters. If all the parameters are the same, the **Anti** data set always has the most skyline points and **Corr** has the least. Moreover, the sizes of g-skyline support structures on **Anti** are the largest and those on **Corr** are the smallest.

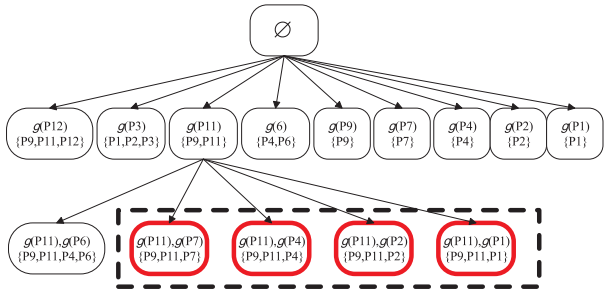


Fig. 6. An example of G-MDS, $l = 3$. The groups with red bold borders are parts of the result. Those in the dashed box can be skipped if the skyline-combination optimization strategy is used.

The real-world data set is extracted from the NBA website.[1] It contains 1,190 league leaders of playoffs. We use the five attributes, Points (PTS), Rebounds (REB), Assists (AST), Steals (STL), and Blocks (BLK), to measure all the players. We preprocess all the values of these attributes to satisfy our assumption, that is "the less the better".

On the one hand, we conduct comprehensive experiments on the synthetic data sets (in Sections 5.1 and 5.2), because they can provide us the data point sets with large sizes and different characteristics. On the other hand, some experiments are conducted on the real-world data set (in Section 5.3) to verify the conclusions drawn from the experimental results on the synthetic data sets.

We use the algorithms in [4] as baselines with the source code obtained from the authors. Also, we have implemented the proposed algorithms in Java. All experiments are conducted on a PC with Intel i7 CPU and 16 GB main memory.

5.1 Constructing G-Skyline Support Structures

In this section, we focus on the g-skyline support structures through comparing our MDG with DSG from the aspect of construction time and structure size on the three synthetic data sets.

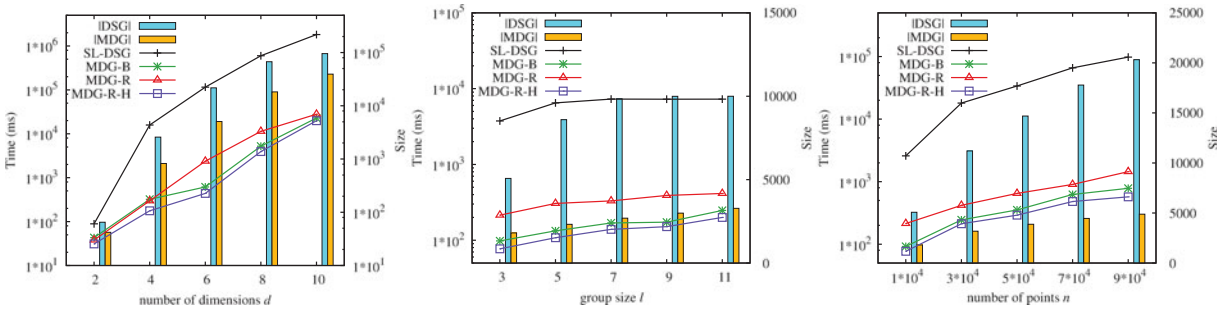
We mark the DSG construction algorithm as SL-DSG. Meanwhile, we construct MDG with three algorithms proposed in Section 3.2:

- MDG-B, the basic MDG Construction Algorithm shown in Algorithm 1.
- MDG-R, the algorithm shown in Algorithm 2 with a normal R-tree.
- MDG-R-H, the improved variant of MDG-R, which utilizes the heuristic node splitting strategy.

The experimental results on the three synthetic data sets are shown in Figs. 7, 8, and 9. In each sub-figure, four lines represent the running time of the four algorithms, and two bars record the sizes of DSG and MDG, denoted as |DSG| and |MDG|.

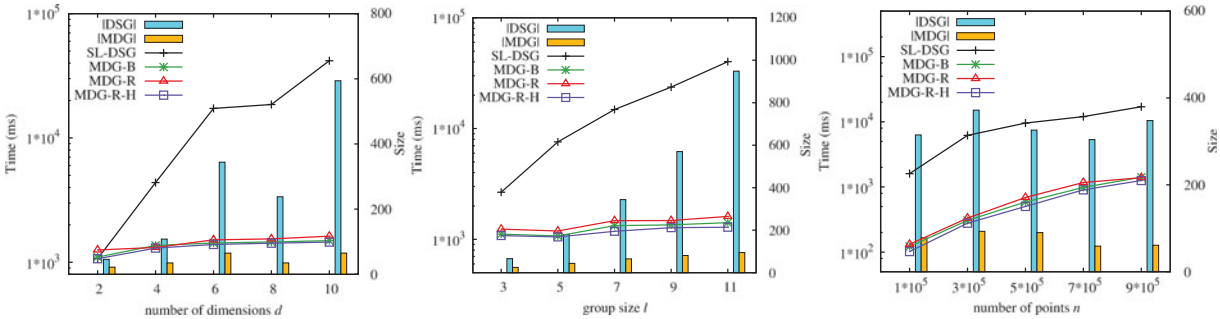
We investigate the impact of the three important parameters, i.e., the number of dimensions d, the group size l and the number of points n, on our algorithms. In each experiment, we fix two parameters and study how the variety of the third parameter affects the algorithms.

1. http://stats.nba.com/



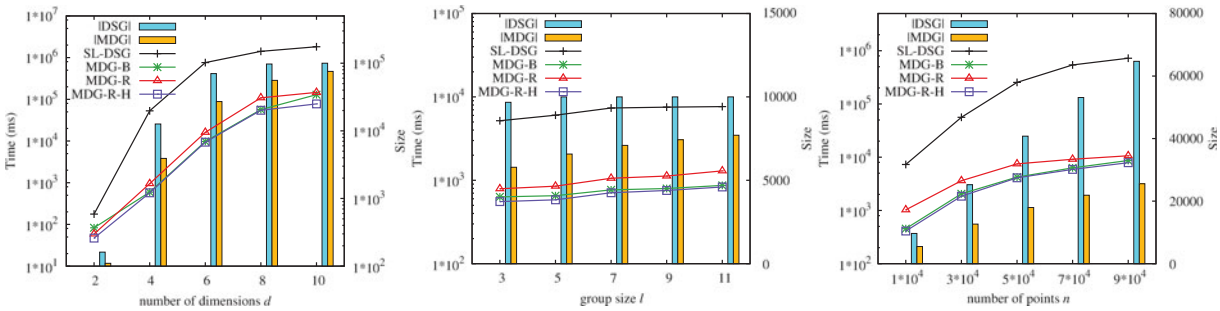
(a) **Indep**, with l=3 and n=100,000 (b) **Indep**, with d=6 and n=10,000 (c) **Indep**, with $d = 6$ and $l = 3$

Fig. 7. Time cost of constructing DSG and MDG and the sizes of them on Indep.



(a) **Corr**, with l=7 and n=1,000,000 (b) **Corr**, with d=6 and n=1,000,000 (c) **Corr**, with $d = 6$ and $l = 7$

Fig. 8. Time cost of constructing DSG and MDG and the sizes of them on Corr.



(a) **Anti**, with l=3 and n=100,000 (b) **Anti**, with d=6 and n=10,000 (c) **Anti**, with $d = 6$ and $l = 3$

Fig. 9. Time cost of constructing DSG and MDG and the sizes of them on Anti.

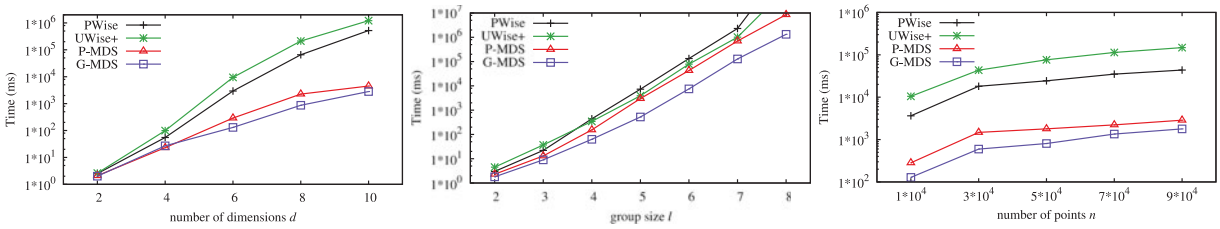
Note that the fixed values of a parameter for different data sets are not always the same. For instance, $n = 10,000$ in Figs. 7b and 9b and $n = 1,000,000$ in Fig. 8b. This different setting is decided by the running time. In our experiments w.r.t. Figs. 7b, 8b, and 9b, algorithms run much faster on Corr than on the other two data sets. If we set n in Fig. 8b with the same value in Figs. 7b and 9b, the running time of the algorithms will be so small that we cannot compare their performance. If we use the same data size in Figs. 7b and 9b as that in Fig. 8b, some algorithm will cost unacceptably long time. This parameter setting strategy applies to the experiments in the next section, i.e., experiments corresponding to Figs. 10, 11, and 12.

In general, MDG-R-H is always the fastest algorithm, and its time cost is surprisingly one or two orders of magnitude smaller than that of SL-DSG in almost all the cases. The other two MDG construction algorithms are a little slower than MDG-R-H. The size of MDG is 10-70 percent of the size of DSG under the same conditions.

From Figs. 7a, 8a, and 9a, we can find that MDG-R-H is only two times faster than SL-DSG when $d = 2$. However, the gap becomes larger when d increases, and reaches more than 10 times when $d = 10$. This observation is consistent with our discussion about the time complexity in Section 3.3.2 and confirms that our g-skyline support structure, MDG, is very efficient for higher-dimensional data.

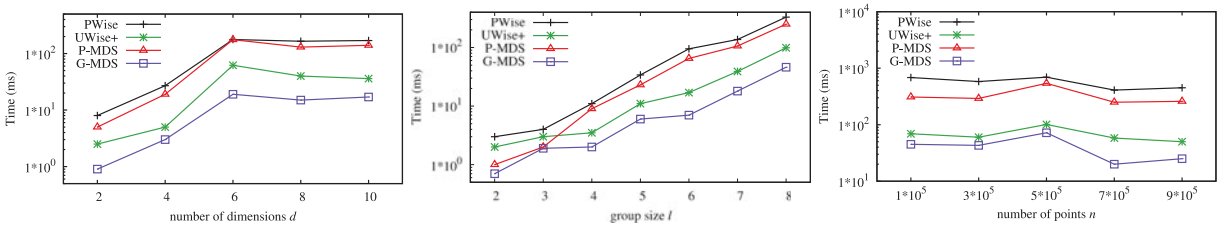
In Fig. 8b, the running time of four algorithms and the sizes of DSG and MDG all increase significantly when the group size l gets larger. But the running time and the sizes in Figs. 7b and 9b, corresponding to **Indep** and **Anti**, change slightly when $l \geq 7$. On these two data sets, the size of DSG almost equals to the number of point n when $l = 7$, so that the sizes of DSG and MDG and the time cost to construct them hardly increase when l gets further larger.

Fig. 8c shows that the time cost of the four algorithms grow almost linearly with the increase of the number of points n on **Corr**. However, on **Indep** and **Anti**, as shown in Figs. 7c and 9c respectively, the growth of the running time of four algorithms is faster than linear growth. From the discussion in Section 3.3.2, we can find that the time complexity is $O(n \times \log n + n \times n_d)$ for SL-DSG, $O(n \times \log n + n \times n_m)$ for MDG-B and $O(n \times \log n + n \times \log n_m)$ for both MDG-R and MDG-R-H, where $n_d = |\text{DSG}|$ and $n_m = |\text{MDG}|$.



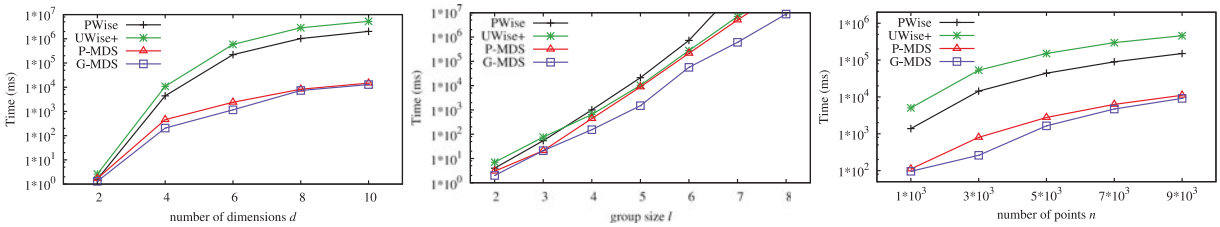
(a) **Indep**, with l=3 and n=10,000 (b) **Indep**, with d=4 and n=1,000 (c) **Indep**, with $d = 6$ and $l = 3$

Fig. 10. Time cost of searching for g-skyline groups based on g-skyline support structures on Indep.



(a) **Corr**, with l=7 and n=1,000,000 (b) **Corr**, with d=6 and n=1,000,000 (c) **Corr**, with $d = 6$ and $l = 7$

Fig. 11. Time cost of searching for g-skyline groups based on g-skyline support structures on Corr.



(a) **Anti**, with l=3 and n=10,000 (b) **Anti**, with d=3 and n=1,000 (c) **Anti**, with $d = 6$ and $l = 3$

Fig. 12. Time cost of searching for g-skyline groups based on g-skyline support structures on Anti.

Empirically, the $O(n \times \log n)$ term is approximately linear to n. Therefore, when n_d and n_m change slightly, which only holds in Fig. 8c, the relationship between time cost of the four algorithms and n is approximately linear.

In each experiment, MDG-R-H is significantly faster than MDG-R, which means our heuristic node splitting strategy obtains the expected result. Since MDG-R-H is always faster than MDG-B, we believe it is a good choice to index the points in MDG with our improved R-tree.

5.2 Searching for G-Skyline Groups

In this section, we present the experiments on searching for g-skyline groups based on the g-skyline support structures. We compare the following four algorithms on the three synthetic data sets.

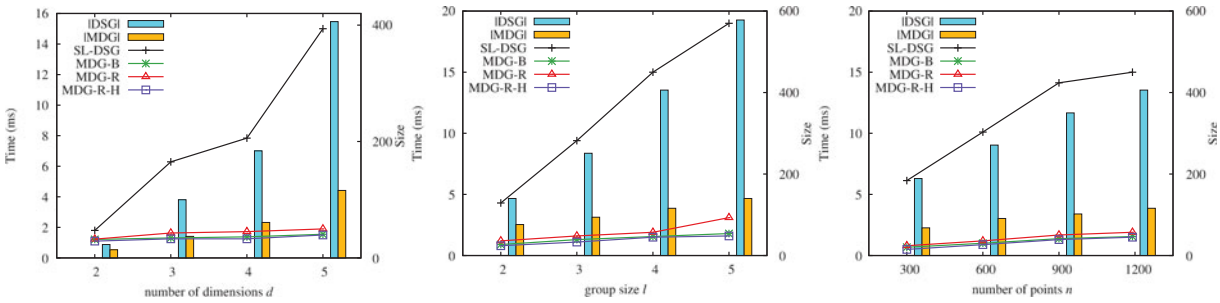
- PWise [4]. It generates new candidate groups through adding single points into the current group and searches for the results with sub-tree pruning and tail set pruning strategies.
- UWise+ [4]. This algorithm generates new candidate groups by adding parent groups (a. k. a. unit groups) and searches for the results with superset pruning, tail set pruning, subset pruning and unit group reordering strategies.
- P-MDS. This is the searching algorithm shown in Algorithm 3, and we utilize the skyline-combination optimization strategy in Section 4.3 on it.
- G-MDS. We present this searching algorithm in Algorithm 5. Also, the skyline-combination optimization is used in it.

The input for PWise and UWise+ is the DSG but not the MDG used in P-MDS and G-MDS. However, the first step in both PWise and UWise+ is to preprocess the DSG and obtain a structure which can be seen as a variant of MDG. Thus, the comparison among these four algorithms is fair and reasonable for we do not take the preprocessing time of PWise and UWise+ into account.

Figs. 10, 11, and 12 illustrate the experimental results. In general, G-MDS has the best performance, and is more than one order of magnitude faster than the better one between PWise and UWise+ in most cases. In all the figures, G-MDS is better than UWise+ and P-MDS is better than PWise, which means our different pruning strategies and skyline-combination optimization are really effective.

On the **Corr** data set (shown in Figs. 11a, 11b, and 11c), G-MDS is only 2–5 times faster than UWise+, which is better than PWise. It is because that on this data set the size of MDG is always small, seeing Figs. 8a, 8b, and 8c, and thus the searching based on MDG is simple but fast. Moreover, we can find that in the experiments on **Corr**, the time cost of constructing the g-skyline support structures takes a much higher proportion than that of searching for the results. Thus, our approach with MDG-R-H and G-MDS is still significantly better than the combination of SL-DSG and UWise+ on this data set.

Fig. 11c seems a bit strange that the running time of the four algorithms does not always increase with the number of points. The reason is that the running time of these algorithms is related to the number of points in MDG but not the points in the whole data set, and the size of MDG is not positively related to n on **Corr**, as shown in Fig. 8c. It is worth noting that the size of MDG is positively related to n



(a) **NBA**, with l=4 and n=1,190 (b) **NBA**, with d=5 and n=1,190 (c) **NBA**, with $d = 5$ and $l = 4$

Fig. 13. Time cost of constructing DSG and MDG and the sizes of them on NBA.

on **Indep** and **Anti**. We provide one explanation on why this unusual phenomenon only occurs on **Corr** as follows. We know that given a data point set and its MDG, new data points may trigger two types of actions which can affect the size of this MDG: (Act1) A new point is only dominated by such few points that it can be added into the MDG; (Act2) A new point dominates some points in the MDG so that a few points must be removed from the MDG. Notice that these two actions may occur at the same time. One of the key features of **Corr** is that there exists a great deal of dominance relationship, and thus the action Act2 is more likely to occur on the data set. Therefore, for **Corr**, with the increase of n, sometimes the influence of Act2 may be stronger than that of Act1, i.e., the size of MDG decreases.

Figs. 10a and 12a show the running time of four algorithms with different d, the number of dimensions. When $d = 2$, the four algorithms take similar time (about 1ms) to search for the result. However, with the increase of d, G-MDS and P-MDS have more and more advantages over PWise and UWise+. For example, G-MDS is about 200 times faster than PWise on **Anti** and **Indep** when $d = 10$. This experimental result confirms again that our algorithms are more friendly to higher-dimensional data.

The relationships between the running time of four algorithms and the group size l on the three data sets are shown in Figs. 10b, 11b, and 12b. We can find that G-MDS and UWise+ are more friendly than P-MDS and PWise to larger l. For instance, in Fig. 10b, the gap between G-MDS and P-MDS becomes larger with the increase of l, and UWise+ overtakes PWise when $l = 4$. Both G-MDS and UWise+ generate new candidate groups through adding parent groups, while P-MDS and PWise add single points to generate new candidate groups. Therefore, G-MDS and UWise+ have advantages of the cost of generating candidate groups. When l increases, the size of parent groups becomes larger, and then these two algorithms expand their advantages.

5.3 Experiments on the Real-World Data Set

In this section, we present our experimental results on the real-world data set, **NBA**.

We conduct the experiments in the previous two sections on the data set **NBA** to verify our conclusions drawn on the synthetic data sets. The experimental results are shown in Figs. 13 and 14.

The experiments corresponding to Fig. 13 are about constructing g-skyline support structures. We can obtain the same conclusions in Section 5.1, i.e., the size of MDG is much smaller than that of DSG and MDG-R-H is the best algorithm. Fig. 13a shows that MDG-R-H is about two times faster than SL-DSG when $d = 2$, but the gap increases and reaches one order of magnitude rapidly when $d > 2$, which verifies the inefficiency of DSG for high-dimensional data. Due to the small size of MDG, MDG-B performs better than MDG-R and very close to MDG-R-H, since the latter two suffer from the extra overhead of R-tree.

Fig. 14 illustrates the experiments comparing the four searching algorithms. The same as the results on synthetic data sets, G-MDS is the fastest algorithm, and G-MDS and P-MDS are better than UWise+ and PWise respectively. Similar to the experimental results on **Corr**, G-MDS is about two times faster than UWise+ due to the small size of MDG. We can also find that G-MDS and UWise+ are more friendly than P-MDS and PWise for larger l, which is an important conclusion in Section 5.2.

6 Discussion

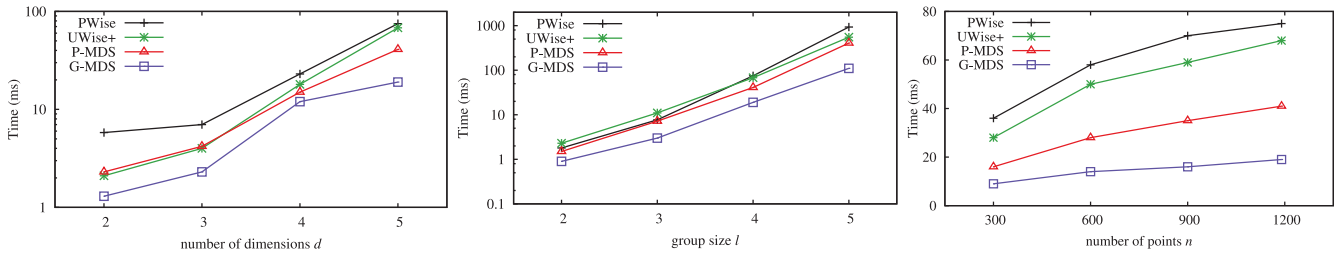
In this section, we mainly discuss two problems, i.e., how to support large-scale data and how to support multiple g-skyline queries with different values of l.

6.1 Large-Scale Data

Our proposed algorithms are all in-memory, i.e., we assume that the demand of memory space for input data and extra structures (e.g., R-tree index) is satisfied.

We first talk about the memory requirement for our algorithms. Suppose the number of data points is n, the number of points in MDG is m (the size of MDG is $O(m * l)$, since MDG is a graph w.r.t. m nodes and the maximal degree is $l - 1$), the number of dimensions is d, and the group size is l. For all the MDG constructing algorithms (MDG-B, MDG-R and MDG-R-H), the size of the input data is $O(n * d)$ and that of the output data is $O(m * l)$. However, these algorithms need different extra spaces. MDG-R and MDG-R-H need $O(m * d)$ space for R-tree, while MDG-B needs no extra space. For the searching algorithms (P-MDS and G-MDS), the space requirement w.r.t. the input of them is $O(m * l)$, i.e., the size of MDG, and the extra space is $O(l)$, which can be ignored since l is always small.

Then, let us discuss the effect of the increasing scale of input data. Suppose the memory space we have is smaller than $O(n * d)$ but larger than $O(m * l)$. It is easy to find that P-MDS and G-MDS are not affected, because their space requirements are satisfied. For MDG constructing algorithms (MDG-B, MDG-R and MDG-R-H), the input data have to be stored in disk. The first thing of all of them is sorting the input data, and thus we need to change the in-memory sort into disk-based sort in them. In the rest of them, the data points are used in order for once, so we only need to add some code to read data points from disk and



(a) **NBA**, with l=4 and n=1,190 (b) **NBA**, with d=5 and n=1,190 (c) **NBA**, with $d = 5$ and $l = 4$

Fig. 14. Time cost of searching for g-skyline groups based on g-skyline support structures on NBA.

other parts of these algorithms could remain the same. Note that MDG-R and MDG-R-H may suffer from their needs of extra space ($O(m * d)$) for R-tree index and cannot ensure that the R-tree index fits into the memory. Therefore, MDG-B is more proper in this case.

If we go on increasing the scale of input data so that $O(m * l)$ is larger than the memory size we have, the MDG will not fit into the memory. In this situation, all of the proposed algorithms need to be re-designed as disk-based ones. The major challenge is to store the MDG on disks. MDG is a directed graph with three operations: 1. Traversing the nodes in the descending order of their in-degrees; 2. Inserting a new node with some in-edges; 3. Finding all the in-neighbors for a specific node. We need to design a storage strategy for MDG to efficiently support the above three operations. We leave this problem as a future work.

6.2 Multiple Queries with Different Values of *l*

In the original definition of g-skyline problem, the value of l is always fixed. In this section, we discuss how to support multiple g-skyline queries with different values of l.

Our basic idea is to make full use of existing MDG to deal with new queries. Suppose that we have constructed an MDG G w.r.t. $l = l_c$ and we need to process a new query with $l = l_n$. If $l_n \leq l_c$, we can directly leverage the existing MDG G to search for g-skyline groups with P-MDS and G-MDS by ignoring points which have more than $l_n - 1$ parents in G. If $l_n > l_c$, we should incrementally expand G with $l = l_n$, and then search for g-skyline groups on the resultant new MDG. Algorithm 7 shows the pseudo-code of the incremental algorithm for MDG-R (MDR-R-H), and the incremental algorithm for MDG-B is similar to it. The major differences between Algorithms 7 and 2 are as follows: Algorithm 2 needs to sort the data points while Algorithm 7 does not sort them since they are already ordered; Algorithm 2 traverses all the points in MDG (Line 5), however Algorithm 7 only considers the points not in the existing MDG (Line 2).

An experiment is conducted to verify the above incremental strategy. We use the **Indep** data set ($n = 100{,}000$ and $d = 6$) with four consecutive g-skyline queries ($l = 2, 4, 3, 5$). The incremental strategy is compared with the ordinary approach, i.e., re-construct a new MDG for each query. Since the searching algorithms (P-MDS and G-MDS) in these two strategies are almost the same, we only record the time cost for constructing the MDG w.r.t. the four consecutive queries in Table 1. We can see that the two strategies need similar time cost for the first query, but the incremental strategy can save much time for the subsequent queries. Incrementally constructing MDG in Q_2 and Q_4 saves much time compared with re-constructing new MDG, and reusing the existing MDG in Q_3 can avoid any computation on constructing MDG.

Algorithm 7. Incremental MDG Construction with R-Tree Algorithm

Require:
The data point set P, the size of each g-skyline group l, current MDG $G = (V, E)$, current R-tree R.
Ensure:
G
1: $L \leftarrow \emptyset$ /*L consists of all the points which have $l - 1$ parents*/
2: **for all** $p \in P \setminus V$ **do**
3: $flag \leftarrow$ TRUE
4: **for all** $p' \in L$ **do**
5: **if** $p' \prec p$ **then**
6: $flag \leftarrow$ FALSE
7: break
8: **end if**
9: **end for**
10: **if** $flag$ is True **then**
11: $parents = SearchForParents(p, R, l)$
12: **end if**
13: **if** $|parents| < l$ **then**
14: $V \leftarrow V \cup \{p\}$
15: **for all** $p' \in parents$ **do**
16: Add an edge into E from p' to p
17: **end for**
18: **if** $|parents| = l - 1$ **then**
19: $L = L \cup \{p\}$
20: **end if**
21: $InsertPoint(p, R)$
22: **end if**
23: **end for**
24: **return** G

7 Related Work

In this section, we survey some studies that are related to our work.

In recent years, the skyline problem has attracted much attention in the data management field ever since it was proposed in [5]. Two algorithms to compute the skyline, BNL and D&C, are presented in [5]. BNL uses transitivity and monotonicity to obtain efficiency over sorted data, while D&C halves the data space at an arbitrary dimension's median and solves each half recursively. Following work proposes two categories of algorithms based on BNL and D&C, i.e., sort-based skyline algorithms [7], [8], [9], [10] and partition-based skyline algorithms [11], [12], [13]. Several studies compute skylines based on GPUs [14] and the map-reduce framework [15], [16].

Many researchers study the variants of the skyline problem. Papadias et al. [17] first proposed the definition of

TABLE 1
Time Cost (ms) for Constructing MDG w.r.t. Four Consecutive Queries on the Same Data Set under Two Strategies

	$Q_1(l=2)$	$Q_2(l=4)$	$Q_3(l=3)$	$Q_4(l=5)$
Incremental	706	792	0	474
None-Incremental	696	1,331	873	1,450

For incremental strategy, we construct a new MDG in Q_1, incrementally construct MDG in Q_2 and Q_4, and reuse the existing MDG in Q_3. For non-incremental strategy, we build a new MDG in each query (we do not sort data points in Q_2, Q_3 and Q_4, since the points are already ordered after Q_1).

group-by skyline, i.e., grouping the data points by a group-by attribute and then computing the skyline for each group. The processing of group-by skyline queries in relational databases is studied in [18]. In the data warehousing environment, Yiu et al. [19] proposed to build data cubes using the skyline operation as the aggregation function, and data cubes built in this way are called group-by skyline cubes. The reverse skyline query, which retrieves data objects from the database considering the manufacturers point of view, is proposed in [20]. The parallelizing of reverse skyline is studied in [16] and [21] based on quad-tree. Pei et al. introduced the first probabilistic skyline problem, p-skyline in [22]. Since then, many researchers study about the skyline on uncertain data [23], [24], [25], [26]. Some of other popular variants are skyline in subspaces [27], [28], [29], [30], [31], [32], infra-skyline in social networks [33], metric skyline [34], [35], and continuous skyline [36], [37], [38].

The variant most related to our work is group-based skyline [1], [2], [3], [4], [39]. These studies propose the definition of group dominance, and then the skyline groups are defined as the groups which are not dominated by any other groups. Earlier work [1], [2], [3] represents each group with a virtual aggregation point. The values of the aggregation point on different dimensions are the aggregations over the points in the group on corresponding dimensions, and many aggregation functions can be used, such as SUM, MIN, and MAX. The dominance of groups is determined by the dominance of their aggregation points. There are some shortcomings of this definition, e.g., some optimal groups are not captured [4]. Liu et al. [4] give a novel definition of group dominance and the corresponding definition of group-based skyline, called g-skyline. In [4], the directed skyline graph (DSG) is designed and two searching algorithms, PWise and UWise+, based on DSG are proposed to compute g-skyline groups. Studies in [1], [2], [3], [4] concern about the deterministic dominance relationship between two groups. In addition to this, the work in [39] defines a group dominance notion based on the uncertain skyline definition.

Our study follows the definition of g-skyline [4] and proposes an efficient approach to finding out g-skyline groups. One of our contributions is proposing MDG, which is better than DSG [4] in both theory and practice. In theory, we point out the importance of MDG. Through Theorem 3, we prove that MDG is a minimum g-skyline support structure, which means it contains no such point that cannot be in any g-skyline groups. Therefore, searching for g-skyline groups based on MDG is the most efficient. The authors of [4] propose the concepts of skyline-layer as well as DSG, and prove searching for g-skyline group on DSG is sufficient. They realized that DSG has redundancy and proposed the preprocessing step to eliminate the redundancy, but they did not point out the reason and the correctness of the preprocessing step. In practice, our algorithms (MDG-B, MDG-R and MDG-R-H) show significant advantages over the SL-DSG algorithm in [4]. Our algorithms need smaller space (10-70 percent) and speed up orders of magnitudes.

8 Conclusions

This paper aims to address the problem of finding g-skyline groups. We present a new efficient method to overcome the shortcomings of the existing approaches. First, we bring forward a novel g-skyline support structure called *minimum dominance graph* (MDG). Different from the support structure DSG, MDG is proved to be non-redundant. Next, we give the algorithm about how to construct MDG with R-tree, and introduce a heuristic node splitting strategy for R-tree to improve the efficiency of the specified search in our algorithm. Then, we propose two searching algorithms, P-MDS and G-MDS, to find out g-skyline groups with different strategies generating new candidate groups. Both the algorithms use the pruning strategies based on our *Verification Theorem*, which are more effective than those used in P-Wise and U-Wise+, i.e., the state-of-the-art algorithms. Last, comprehensive experiments are conducted to evaluate the performance of the proposed algorithms. The experimental results show that our algorithms, both for constructing g-skyline support structures and for finding out g-skyline groups, are orders of magnitude faster than the state-of-the-art in most cases.

In the future, considering the number of g-skyline groups is always huge, we will focus on further operations on the g-skyline, e.g., selecting more representative g-skyline groups from various viewpoints. Moreover, we will study the storage strategy for MDG to support large-scale input data.

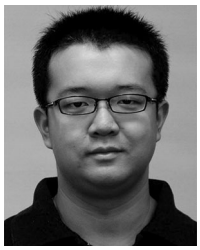
Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No. 61373023), the Intelligent Manufacturing Comprehensive Standardization and New Pattern Application Project of Ministry of Industry and Information Technology (Experimental validation of key technical standards for trusted services in industrial Internet), and the China National Arts Fund (No. 20164129).

References

[1] C. Li, N. Zhang, N. Hassan, S. Rajasekaran, and G. Das, "On skyline groups," in *Proc. ACM Int. Conf. Inf. Knowl. Manag.*, 2012, pp. 2119–2123.
[2] N. Zhang, C. Li, N. Hassan, S. Rajasekaran, and G. Das, "On skyline groups," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 942–956, Apr. 2014.
[3] H. Im and S. Park, "Group skyline computation," *Inf. Sci.*, vol. 188, pp. 151–169, 2012.
[4] J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang, "Finding pareto optimal groups: Group-based skyline," *Proc. VLDB Endowment*, vol. 8, no. 13, pp. 2086–2097, Sep. 2015.
[5] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. IEEE Int. Conf. Data Eng.*, 2001, pp. 421–430.
[6] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1984, pp. 47–57.
[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. IEEE Int. Conf. Data Eng.*, 2003, pp. 717–719.
[8] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Trans. Database Syst.*, vol. 33, no. 4, pp. 31:1–31:49, 2008.
[9] K. C. K. Lee, B. Zheng, H. Li, and W. Lee, "Approaching the skyline in Z order," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 279–290.

[10] P. Godfrey, R. Shipley, and J. Gryz, "Algorithms and analyses for maximal vector computation," *VLDB J.*, vol. 16, no. 1, pp. 5–28, 2007.
[11] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable skyline computation using object-based space partitioning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 483–494.
[12] S. Chester, D. Sidlauskas, I. Assent, and K. S. Bøgh, "Scalable parallelization of skyline computation for multi-core processors," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 1083–1094.
[13] A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-based space partitioning for efficient parallel skyline computation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 227–238.
[14] K. S. Bøgh, S. Chester, and I. Assent, "Skyalign: A portable, work-efficient skyline algorithm for multicore and GPU architectures," *VLDB J.*, vol. 25, no. 6, pp. 817–841, 2016.
[15] J. Zhang, X. Jiang, W. Ku, and X. Qin, "Efficient parallel skyline evaluation using MapReduce," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 7, pp. 1996–2009, Jul. 2016.
[16] Y. Park, J. Min, and K. Shim, "Parallel computation of skyline and reverse skyline queries using MapReduce," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 2002–2013, 2013.
[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[18] M. Luk, M. L. Yiu, and E. Lo, "Group-by skyline query processing in relational engines," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2009, pp. 1433–1436.
[19] M. L. Yiu, E. Lo, and D. Yung, "Measuring the sky: On computing data cubes via skylining the measures," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 3, pp. 492–505, Mar. 2012.
[20] E. Dellis and B. Seeger, "Efficient computation of reverse skyline queries," in *Proc. Int. Conf. Very Large Data Bases*, 2007, pp. 291–302.
[21] M. S. Islam, C. Liu, J. W. Rahayu, and T. Anwar, "Q+tree: An efficient quad tree based data indexing for parallelizing dynamic and reverse skylines," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2016, pp. 1291–1300.
[22] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Proc. Int. Conf. Very Large Data Bases*, 2007, pp. 15–26.
[23] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. X. Yu, "Probabilistic skyline operator over sliding windows," in *Proc. IEEE Int. Conf. Data Eng.*, 2009, pp. 1060–1071.
[24] X. Lian and L. Chen, "Reverse skyline search in uncertain databases," *ACM Trans. Database Syst.*, vol. 35, no. 1, pp. 3:1–3:49, 2010.
[25] W. Zhang, X. Lin, Y. Zhang, M. A. Cheema, and Q. Zhang, "Stochastic skylines," *ACM Trans. Database Syst.*, vol. 37, no. 2, pp. 14:1–14:34, 2012.
[26] X. Liu, D. Yang, M. Ye, and W. Lee, "U-skyline: A new skyline query for uncertain databases," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 945–960, Apr. 2013.
[27] C. Y. Chan, H. V. Jagadish, K. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 503–514.
[28] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the best views of skyline: A semantic approach based on decisive subspaces," in *Proc. Int. Conf. Very Large Data Bases*, 2005, pp. 253–264.
[29] J. Pei, et al., "Towards multidimensional subspace skyline analysis," *ACM Trans. Database Syst.*, vol. 31, no. 4, pp. 1335–1381, 2006.
[30] Y. Tao, X. Xiao, and J. Pei, "Efficient skyline and top-k retrieval in subspaces," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 8, pp. 1072–1088, Aug. 2007.
[31] J. Pei, A. W. Fu, X. Lin, and H. Wang, "Computing compressed multidimensional skyline cubes efficiently," in *Proc. IEEE Int. Conf. Data Eng.*, 2007, pp. 96–105.
[32] J. Lee and S. Hwang, "Toward efficient multidimensional subspace skyline computation," *VLDB J.*, vol. 23, no. 1, pp. 129–145, 2014.
[33] Z. Peng and C. Wang, "Member promotion in social networks via skyline," *World Wide Web J.*, vol. 17, no. 4, pp. 457–492, 2014.
[34] L. Chen and X. Lian, "Dynamic skyline queries in metric spaces," in *Proc. Int. Conf. Extending Database Technol.*, 2008, pp. 333–343.
[35] D. Fuhry, R. Jin, and D. Zhang, "Efficient skyline computation in metric space," in *Proc. Int. Conf. Extending Database Technol.*, 2009, pp. 1042–1051.
[36] Z. Huang, H. Lu, B. C. Ooi, and A. K. H. Tung, "Continuous skyline queries for moving objects," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 12, pp. 1645–1658, Dec. 2006.
[37] M. Lee and S. Hwang, "Continuous skylining on volatile moving data," in *Proc. IEEE Int. Conf. Data Eng.*, 2009, pp. 1568–1575.
[38] Z. Zhang, R. Cheng, D. Papadias, and A. K. H. Tung, "Minimizing the communication cost for continuous skyline maintenance," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 495–508.
[39] M. Magnani and I. Assent, "From stars to galaxies: Skyline queries on aggregate data," in *Proc. Int. Conf. Extending Database Technol.*, 2013, pp. 477–488.



Changping Wang received the BS degree in computer software from Tsinghua University, in 2012. He is currently working toward the PhD degree in the School of Software, Tsinghua University. His major research interests include social network analysis, data mining, and database applications.

Chaokun Wang received the BEng degree in computer science and technology, the MSc degree in computational mathematics, and the PhD degree in computer software and theory, in 1997, 2000, and 2005, respectively, from the Harbin Institute of Technology, China. He joined the faculty of the School of Software, Tsinghua University, in February 2006, where currently he is an associate professor. His current research interest includes social network analysis, graph data management, and music computing. He is a member of the IEEE.

Gaoyang Guo received the BE degree in software engineering from the Harbin Institute of Technology, Harbin, China, in 2016. He is currently working toward the PhD degree in software engineering in the School of Software, Tsinghua University. His research interests include social networks, natural language processing, and deep learning.

Xiaojun Ye received the BS degree in mechanical engineering from Northwest Polytechnical University, Xian, China, in 1987 and the PhD degree in information engineering from INSA Lyon, France, in 1994. Currently, he is a professor in the School of Software, Tsinghua University, Beijing, China. His research interests include cloud data management, data security and privacy, and database system testing.

Philip S. Yu received the PhD degree in electrical engineering from Stanford University. He is a distinguished professor in computer science with the University of Illinois at Chicago and is the Wexler chair in Information Technology. His research interests include big data, data mining, data stream, database, and privacy. He is the editor-in-chief of the *ACM Transactions on Knowledge Discovery from Data*. He was the editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering* (2001-2004). He received the ACM SIGKDD 2016 Innovation Award, the Research Contributions Award from IEEE International Conference on Data Mining (2003), and the Technical Achievement Award from the IEEE Computer Society (2013). He is a fellow of the IEEE and ACM.