Progressive Approaches for Pareto Optimal Groups Computation

Xu Zhou, Kenli Li, *Senior Member, IEEE*, Zhibang Yang, Guoqing Xiao, and Keqin Li, *Fellow, IEEE*

Abstract—Group skyline query is a powerful tool for optimal group analysis. Most of the existing group skyline queries select optimal groups by comparing the dominance relationship between aggregate-based points; such feature creates difficulties for users to specify an appropriate aggregate function. Besides, many significant groups that have great attractions to users in practice may be overlooked. To address these issues, the group skyline (GSky) query is formulated on the basis of a general definition of group dominance operator. While the existing GSky query algorithms are effective, there is still room for improvement in terms of progressiveness and efficiency. In this paper, we propose some new lemmas which facilitate direct generation of the GSky query results. Consecutively, we design a layered unit-based (LU) algorithm that applies a layered optimum strategy. Additionally, for the GSky query over the data that are dynamically produced and cannot be indexed, we propose a novel index-independent algorithm, called sorted-based progressive (SP) algorithm. The experimental results demonstrate the effectiveness, efficiency, and progressiveness of the proposed algorithms. By comparing with the state-of-the-art algorithm for the GSky query, our LU algorithm is more scalable and two orders of magnitude faster.

Index Terms—Data management, group skyline query, progressive algorithms

1 Introduction

Skyline query, as a useful tool in decision making applications [1], [2], receives growing attention in the database community. Given a set of data points P in a d-dimensional space, the skyline query retrieves the points that are not dominated by any other point in P [3]. Here, a point p dominates another point p', if and only if p is not worse than p' in all the dimensions and p is better than p' in at least one dimension. A point that is not dominated by any other point is called skyline.

There are abundant works about skyline queries [2], [3], [4], [5]. In addition to the traditional skyline query, many skyline query variants have also been studied in the literature. These variants include reverse skyline queries [6], [7], skyline queries under distributed environments [8], [9], [10], probabilistic skyline queries [11], [12], [13], [14], skyline queries with constraints [15], [16], [17], [18], [19], why-not range-based skyline queries [20], and optimizing quality for probabilistic skyline computation [21], to name just a few.

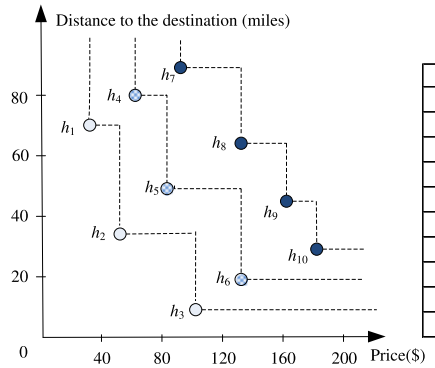
The skyline query and its variants facilitate to analyze individual points, however, it is inadequate for many applications that need to compute optimal groups, such as a travel agency selects some hotels and the coaches of NBA teams select players.

As an example, a travel agency requires to select some hotels for cooperation. Fig. 1 shows a hotel set H that contains ten candidate hotels $h_1, h_2, \ldots, h_{10}$ with two attributes, distance to the destination and price. Without loss of the generality, lower values of the two attributes are preferable. Based on the dominance relationship between hotels, we get the skyline set $\{h_1, h_2, h_3\}$ where each hotel is not dominated by any other one in H. Assume that three hotels is desired. We could select hotels from the skyline set $\{h_1, h_2, h_3\}$. The hotel group $\{h_1, h_2, h_3\}$ that only contains skylines could be returned to the travel agency. However, as analyzed in [22], some other hotel groups such as $\{h_1, h_2, h_4\}$ consists of hotels with low prices and $\{h_3, h_6, h_{10}\}$ including hotels close to the destination may be appeal to the travel agency. Here, the hotel group $\{h_1, h_2, h_4\}$ includes the hotel h_4 that is dominated by h_1 and h_2. In the hotel group $\{h_3, h_6, h_{10}\}$, it contains the hotels h_6 and h_{10} that are not skylines. In essence, h_6 is only dominated by $h_3 \in \{h_3, h_6, h_{10}\}$. Similarly, h_{10} is dominated by h_3 and h_6 which are both included in the group $\{h_3, h_6, h_{10}\}$.

To compute optimal groups of points, there are mainly two approaches. One approach is to pick out optimal groups by the dominance relationship between different aggregate points. The functions, such as SUM, MAX, and MIN, are commonly used to calculate the aggregate points. However, as mentioned in [22], it is difficult to select an appropriate function, and it overlooks many significant

- *X. Zhou, K. Li, and G. Xiao are with the College of Information Science and Engineering, Hunan University, and the National Supercomputing Center in Changsha, Hunan 410082, China. E-mail: zhouxu2006@126.com, {lkl, xiaoguoqing}@hnu.edu.cn.*
- *Z. Yang is with the Department of Mathematics and Computer Science, Changsha University, Changsha, Hunan 410082, China. E-mail: yangzhibang2006@126.com.*
- *K. Li is with the College of Information Science and Engineering, Hunan University, the National Supercomputing Center in Changsha, Hunan 410082, China, and the Department of Computer Science, State University of New York, New Paltz, New York, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 7 Jan. 2018; accepted 1 May 2018. Date of publication 16 May 2018; date of current version 4 Feb. 2019.
(Corresponding author: Kenli Li and Keqin Li.)
Recommended for acceptance by K. Selcuk Candan.
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.2018.2837117



Hotel	Price	Distance
h_1	30	70
h_2	50	35
h_3	110	10
h_4	60	80
h_5	80	50
h_6	130	20
h_7	90	90
h_8	130	65
h_9	160	45
h_{10}	180	30

Fig. 1. The skyline query example of a hotel set.

groups which have great attractions to users in practice. Liu et al. [22] developed another approach, namely group skyline (GSky) query, that aims to identify optimal groups not dominated by any other one of the same size. They formulated a novel group dominance operator based on the dominance relationship between different points of the groups. Given two groups G and G', G dominates G' if and only if each point $p{\in}G$ dominates or is equal to some point $p'{\in}G'$, and for at least one point p, p dominates p'. The group of points is called G-Skyline if it is dominated by no other group of equal size. As well as some G-Skylines only consisting of skyline points, the GSky query reports important groups that include non-skyline points.

Considering the example in Fig. 1, the GSky query in [22] returns the groups $\{h_1, h_2, h_4\}, \{h_1, h_2, h_5\}$, $\{h_2, h_3, h_5\}, \{h_1, h_3, h_6\}, \{h_2, h_3, h_6\}$, and $\{h_3, h_6, h_{10}\}$ that include some non-skylines as well as the group $\{h_1, h_2, h_3\}$ consisting of skylines.

In [22], it organizes the first k skyline layers with a directed skyline graph (DSG) where k is the group size. Based on the DSG, the point-wise and unit-wise algorithms for the GSky query are proposed. Although the two algorithms can process the GSky query effectively, there is still room for improvement from the following two aspects.

- Progressiveness. The point-wise algorithm lacks progressiveness since it cannot get any result until the end of the algorithm. While the unit-wise algorithm can provide the G-Skylines in a progressive manner, its progressiveness could also been greatly improved.
- Efficiency. There are a larger number of redundant dominance tests in the point-wise algorithm. Moreover, in the two algorithms, numerous unqualified candidate groups are generated and the search space can be further reduced.

Furthermore, in many applications the data are dynamically produced (e.g., they arrive from a stream), so they cannot be indexed [2], [23]. In this case, the point-wise and unit-wise algorithms proposed in [22] cannot be utilized to process the GSky query directly, since they are all based on a pre-computed index, namely DSG. Besides, similar to the index-based algorithms for skyline queries, they suffer from the well-known curse of dimensionality and face memory management problems [23], [24].

Skyline computation has received considerable attention in the database community, especially for progressive methods that can quickly return the initial results without reading the entire database. There has been some works about progressive algorithms for the skyline query [12], [13]. The GSky query is much more complex than the skyline queries. The query time may be very long especially when processing a big dataset or with a large group size k. In the experiment, for an anti-correlated dataset with $k = 5$, $d = 3$, and $N = 40,000$, the query time of the state-of-the-art algorithm for the GSky query is larger than 24h. Therefore, progressiveness is also a desirable property for the GSky query algorithms.

In this paper, we develop two progressive algorithms for the GSky query. To begin with, we propose some new lemmas and introduce the layered optimum strategy to boost the GSky query performance, and present the LU algorithm. Different from the UWise+ algorithm in [22], our LU algorithm could generate the G-Skylines directly due to the proposed lemmas. Then, to process the GSky query over the data that are dynamically produced and cannot be indexed, we present the index-independent algorithm, called SP, as an important supplement of the index-based algorithms.

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

- We exploit some new properties of the GSky query and present novel lemmas which facilitate direct generation of the GSky query results.
- We propose a new index-based algorithm, called LU, by integrating the new lemmas and adopting the layered optimum strategy. We analyze correctness and complexity of the LU algorithm.
- We investigate, for the first time, the GSky query over the data that cannot be indexed. We present the index-independent algorithm, namely SP.
- We offer an approach of continuously maintaining the G-Skylines when data updates happen. We also discuss an important variant of the GSky query with a size constraint based on a novel ranking criterion.
- We perform an extensive experimental study with both synthetic and real datasets to demonstrate the performance of our proposed algorithms in terms of effectiveness, efficiency, and progressiveness.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we introduce the GSky query and the DSG. In Section 4, we propose the LU algorithm. In Section 5, we design the SP algorithm. In Section 6, we evaluate the performance of the proposed algorithms by extensive experiments. In Section 7, we offer the approach of continuously maintaining the G-Skylines and formulate the top l GSky query. In Section 8, we conclude the paper and also expatiate the directions for future work.

2 Related Work

In this section, we review the related research about group skyline queries and the skyline queries under constraints.

2.1 Group Skyline Queries

Group skyline queries are very important in many applications that need to compute optimal groups of points. In most of the group skyline queries, optimal groups are computed by the dominance relationship between corresponding aggregate-based points. Su et al. [25] formulated top k

combinatorial skyline query (k-CSQ) which returns the combinatorial skyline tuples whose aggregate values for a certain attribute are the highest. Chung et al. [26] extended the traditional skyline query and formulated a combinatorial skyline query, namely CSQ, to find the outstanding skyline combinations. Im et al. [1] studied the group skyline query based on the dominance relationship that is checked according to the aggregate values of attributes. Magnani et al. [27] introduced the aggregate skyline query that merges two basic database operators, skyline and group by. Zhang et al. [28] focused on the novel problem that aims to report k tuple groups dominated by no other group of equal size. The dominance test is also on the basis of aggregate-based group dominance relationship. Wu et al. [29] investigated a problem of creating competitive products which are not dominated by the products in the existing market. Here each new product is generated by combing its corresponding attributes from different source tables. Jiang et al. [30] defined a top-k combinatorial metric skyline (kCMS) query to find k combinations of data points according to a monotonic preference function. Each combination returned by the kCMS query has the query object in its metric skyline. Recently, Zhou et al. [31] were concerned about product selection under price promotion and formulated a constrained optimal product combination (COPC) problem.

In the group skyline queries aforementioned, the aggregate values of corresponding attributes are taken into account when checking the dominance relationship between different groups. However, it is difficult for users to specify an appropriate aggregate function. Moreover, it overlooks many significant groups that may contain non-skyline points [22].

Liu et al. [22] formulated the Pareto group-based skyline (GSky) query which retrieves G-Skyline groups not g-dominated by any other group of the same size. They presented the directed skyline graph to capture the dominant relationship between the points within the first k skyline layers. Furthermore, two heuristic algorithms, point-wise and unit group-wise algorithms, were designed to process the GSky query. After that, Wang et al. [32] developed the minimum g-skyline support structure, called MDG, and proposed two efficient algorithms. Yu et al. [33] defined the multiple skyline layers, and presented two fast algorithms to compute the G-skylines due to the observation that skyline points contribute more to skyline groups compared to nonskyline points. In this paper, we focus on the GSky query formulated in [22]. We develop the LU algorithm that has better progressiveness and efficiency than the state of the art algorithm for the GSky query. Moreover, we investigate the GSky query over the data that cannot be indexed for the first time and propose the index-independent algorithm.

2.2 Skyline Queries under Constraints

Since the traditional skyline queries always return a large number of query results [19], many approaches are proposed to identify k representative skylines having the highest dominant capacity or the maximum diversification.

Papadias et al. [3] proposed a top k dominating query which aims to find points having the largest dominance capacity. This query has the advantages of both ranking queries and skyline queries, which are with the control on the size of the answer set and without users' efforts to specify ranking functions. Lin et al. [18] studied the problem of selecting k skyline points such that the number of points, which are dominated by at least one of these k skyline points, is maximized. Tao et al. [17] proposed a distance-based skyline query that minimizes the distance between a non-representative skyline point and its nearest representative one. Huang et al. [16] presented an $l-$SkyDiv query to retrieve l skylines having the maximum diversity. Lu et al. [15] concerned the case when the actual cardinality of skyline results is less than the desired cardinality k. They proposed a new approach, namely skyline ordering, to form a skyline-based partitioning of a given data set. Then they applied the set-wide maximization technique, which is to find an object set dominated the largest number of points, to process each partition. Zhou et al. [34] formulated a top k favorite probabilistic products (TFPP) query to select k products which can meet the needs of different customers at the maximum level.

The above representative skyline queries only highlight some features which are stability, scale invariance, diversification of the results, and partial knowledge of the record scoring function [19]. Conversely, Magnani et al. [19] focused on the representative skyline queries in terms of both the significance and diversity of results to satisfy all the features aforementioned.

In general, the above approaches to the skyline queries under constraints cannot be utilized to the GSky query in this paper. The skyline queries with constraints fucus on selecting representative skylines based on some ranking criterions, however, the GSky query pays more attention on groups of points that are dominated by no other group of equal size.

The frequently used symbols are summarized in Table 1.

TABLE 1
The Summary of Frequently Used Notations

Notation	Definition
P	The dataset
N	The size of the dataset
p	A point
G	The group of points
k	The group size
u_p	The unit group of point p
SL_i	The ith skyline layer
$ParSet(p)$	The set of all the parents of p
$TailSet(p)$	The tail set of p

3 Preliminaries

In this section, we introduce the GSky query and the direct skyline graph.

3.1 The Group Skyline (GSky) Query Problem

In this subsection, we introduce the skyline query and the GSky query, respectively.

Definition 3.1 (Skyline Query [3]). *Given a dataset P with cardinality N, the skyline query returns all the points $p \in P$ that are not dominated by any other point $p' \in P - \{p\}$. Without loss of the generality, assume that smaller value is preferred in*

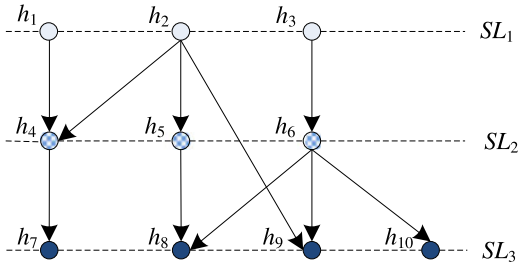


Fig. 2. The DSG over the hotel set in Fig. 1.

each dimension. Then, point p' dominates point p, denoted as $p' \prec p$, if it holds that for all i, $p'[i] \leq p[i]$ and for at least one i, $p'[i] < p[i]$ for $1 \leq i \leq d$. The points $p \in P$ which are not dominated by any other point $p' \in P - \{p\}$ are the skylines.

Definition 3.2 (Skyline Layer). *Given a dataset P with cardinality N and a parameter k, the first skyline layer SL_1 includes all the skylines of P, and the ith skyline layer SL_i for $1 < i \leq k$ consists of the skylines of $P - \bigcup_{j=1}^{i-1} SL_j$.*

Due to Definition 3.2, the points within SL_i are not dominated by any point in $P - \bigcup_{j=1}^{i-1} SL_j$. Assume that the objects within P are uniformly distributed in each dimension, there are no objects within P sharing the same value along any dimension, and all dimensions are reciprocally independent. Let $\hbar_i$ represent the cardinality of SL_i and as introduced in [35], its expected value could be computed as

$$\hbar_i \approx \frac{\left(\ln N_i + \gamma\right)^{d-1}}{(d-1)!}. \tag{1}$$

Here $N_1 = N$, $N_i = N - \sum_{j=1}^{i-1} \hbar_j$ for $2 \leq i \leq k$, and γ is the Euler-Mascheroni constant approximately equal to 0.577 [35].

Definition 3.3 (Parent Set, ParSet). *Given a point $p \in P$, the parent set of p is defined as*

$$ParSet(p) = \{p' \in P | p' \prec p\}.$$

Moreover, for a set $P' \subseteq P$, we have

$$ParSet(P') = \cup_{p \in P'} ParSet(p).$$

Definition 3.4 (Group Dominance [22]). *Given two k-point groups $G \subseteq P$ and $G' \subseteq P$, it holds that G g-dominates G', denoted as $G \prec_g G'$, if there are two permutations of G and G', $G = \{p_1, p_2, \ldots, p_k\}$ and $G' = \{p'_1, p'_2, \ldots, p'_k\}$, satisfying $p_i \preceq p'_i$ for $1 \leq i \leq k$ and $p_i \prec p'_i$ for at least one i. Here $p_i \preceq p'_i$ means that $p_i \prec p'_i$ or p_i is equal to p'_i.*

Definition 3.5 (Group Skyline Query, GSky). *Given a dataset P and a parameter k, the GSky query returns k-point groups $G \subseteq P$ that are g-dominated by no other group of the same size, formally,*

$$GSky(P, k) = \{G \subseteq P | \nexists G' \subseteq P, G' \prec_g G, |G| = |G'| = k\}.$$

The k-point group G that is g-dominated by no other group of the same size is said to be G-Skyline.

Consider the dataset in Fig. 1. For two groups $G = \{h_1, h_2, h_3\}$ and $G' = \{h_4, h_5, h_6\}$, we have $G \prec_g G'$ because $h_1 \prec h_4$, $h_2 \prec h_5$, and $h_3 \prec h_6$. Therefore, G' is not a G-Skyline.

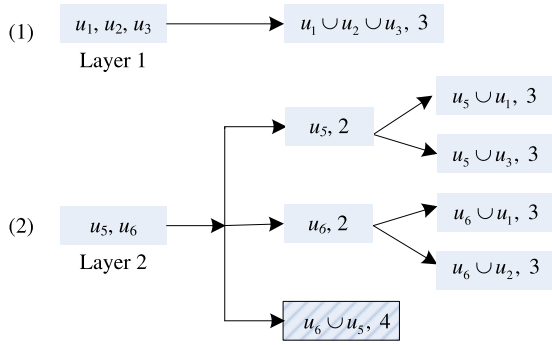


Fig. 3. The LU algorithm over the hotel dataset in Fig. 1 with k=3.

The group $G = \{h_1, h_2, h_3\}$ is a G-Skyline since it is not g-dominated by any other group of equal size.

3.2 The Directed Skyline Graph (DSG)

In [22], it organizes the skyline layers by a directed skyline graph where a node represents a point and an edge represents the dominance relationship between two different points. Besides, it also proves that each G-Skyline with size k only contains the points within the first k skyline layers due to Theorem 3.1. The structure of each node is [*layer index*, *point index*, *parents*, *children*]. Here the layer index indicates the skyline layer that the point lies on and the point index uniquely identifies the point. Additionally, the parents include all the points that dominate this point, and the children contain all the points that are dominated by the point.

Theorem 3.1. *For a given G-Skyline of size k $G = \{p_1, p_2, \ldots, p_k\}$, points $p_i \in G$ are all included in the first k skyline layers [22].*

Going back to the example in Fig. 1, the skyline layers are $SL_1 = \{h_1, h_2, h_3\}$, $SL_2 = \{h_4, h_5, h_6\}$, and $SL_3 = \{h_7, h_8, h_9, h_{10}\}$. Each hotel in SL_1 is dominated by no other hotel in H, while SL_2 includes all the skylines over $H - SL_1$, and SL_3 contains all the skylines over $H - SL_1 - SL_2$. Fig. 2 shows the DSG over the hotel set H in Fig. 1. The edges of the DSG represent the dominance relationship between different hotels. For instance, the edge $h_1 \rightarrow h_4$ means that h_4 is dominated by h_1. In the node representing h_4, it stores its parent set $\{h_1, h_2\}$ and its child set $\{h_7\}$ as well as its layer index 2 and point index 4. Due to Theorem 3.1, the G-Skylines only contain the points in the first 3 skyline layers in this example.

It requires a lot of storage cost and a long time to create the DSG especially when processing large datasets. When creating a DSG of a dataset, it needs to find out all the parents and children of each point in the dataset. This also makes the DSG difficult to be managed. Besides, if adding a new point to the dataset or deleting a point from the dataset, the DSG should be refreshed and abundant nodes of the points dominated by p and dominating p are updated in turn.

As an important input of the index-based algorithms for the GSky query, the DSG in [22] could be adjusted due to the corresponding algorithms. Similar to the UWise+ algorithm in [22], our LU algorithm in Section 4.2 generates candidate groups by combing unit groups. The UWise+ and LU algorithm are only closely related to parents of each point. Thus, the structure of each node in the DSG could be refined as [*layer index*, *point index*, *parents*]. Here for a point

p, its parents include the points which dominate p. By this adjustment, when creating the DSG, it saves the cost of computing the children of each point, and when adding or deleting a point p, only the nodes of the points dominated by p need to be updated.

In addition, to process the GSky query with an arbitrary size, we create the DSG over the dataset. Consequently, when processing the DSG query with size larger than k, it is unnecessary to update the DSG by adding more skyline layers in advance.

4 The Index-Based Algorithm for the GSky Query

In this section, we introduce and analyze the UWise+ algorithm in [22]. Afterward, we present the new index-based algorithm, called LU.

4.1 The UWise+ Algorithm

Liu et al. [22] formulated the GSky query and proposed two effective algorithms, the point-wise and unit-wise algorithms, for processing the GSky query. To the best of our knowledge, the unit-wise algorithm, called UWise+, has better performance and is considered the state of the art algorithm for the GSky query.

Definition 4.1 (Unit Group [22]). *Given a point $p\in P$, the unit group of p is denoted as*

$$u_p = \{p\}\cup ParSet(p).$$

Definition 4.2 (Tail set [22]). *Given a point p, the tail set $TailSet(p)$ consists of all the points p' whose point indexes are larger than that of p.*

In Fig. 2, $TailSet(h_1) = \{h_2, h_3, h_4, h_5,\ h_6, h_7, h_8, h_9, h_{10}\}$ and $TailSet(h_5) = \{h_6, h_7, h_8, h_9, h_{10}\}$.

The UWise+ algorithm contains three stages that are unit group reordering, subset pruning, and G-Skyline computation stages. In the unit group reordering stage, it first builds 1-unit group of each point within the DSG and reorders these unit groups in the reverse order of their point index. By this, the unit groups are likely in the decreasing order of their unit group sizes. Then, in the subset pruning stage, for each candidate group G in 1-unit groups with $|G| < k$, a new candidate group G^{last} is generated by combing G with its tail set. The subtree of G can be pruned directly if $|G^{last}|\leq k$ and G^{last} of size k is reported as a G-Skyline. Moreover, the candidate groups in the subtree of G's right siblings are pruned also. The G-Skyline computation stage is an iteration procedure. For each candidate group G' of size less than k, it computes the parents of each unit group in G' and removes unit groups of these parents from $TailSet(G')$. By merging u_p and G' for each left unit group $u_p\in TailSet(G')$, new candidate groups G'' are generated. The candidate groups G'' of size k could be reported as G-Skylines. Besides, the candidate groups with size no less than k are pruned, and the ones with size less than k are remained for generating new G-Skylines in the next iteration.

UWise+ contains k times iterations at most. In each iteration, for a candidate group G, the UWise+ algorithm creates new candidate groups G' by adding one unit within the tail set of G at a time. It cannot get the G-Skylines containing G until multiple iterations. This results in a large number of unqualified candidate groups which can be pruned in the left iterations, and it takes a lot of time to check them.

4.2 The Layered Unit-Based Algorithm

In the UWise+ algorithm, it generates a large number of candidate groups, prunes the unqualified groups, and picks out the G-Skylines. This always produces redundant computation for generating and pruning unqualified candidate groups. To boost the query performance in terms of progressiveness and efficiency, we develop the layered unit-based (LU) algorithm by adopting a layered optimum strategy and based on the following lemmas.

Lemma 4.1. *Given a G-Skyline G and a group $G'\subseteq SL_i\cap G$ where $1\leq i\leq k$, it holds that $|G'|\leq k-i+1$.*

Proof. Since $G'\subseteq SL_i\cap G$, it holds that G' consists of points within the ith skyline layer and

$$|G'\cup ParSet(G')| = |G'|+|ParSet(G')|\leq k.$$

Due to Definition 3.2, any point $p\in SL_i$ is dominated by at least one point $p'\in SL_j$ for $1\leq j<i$. Therefore $|ParSet(G')|\geq i-1$. Since $|G'|+|ParSet(G')|\leq k$ and $|ParSet(G')|\geq i-1$, it is easy to get $|G'|\leq k-|ParSet(G')|\leq k-i+1$. Therefore, this lemma holds. □

Lemma 4.1 means that each G-Skyline G of size k contains at most $k-i+1$ points within SL_i for $1\leq i\leq k$. It provides an upper bound of the size of $G\cap SL_i$.

Lemma 4.2. *Each k-point group $G\subseteq \bigcup_{j=1}^{i} SL_j$, which contains at least one point within SL_i, only has probability to be g-dominated by a group $G'\subseteq \bigcup_{j=1}^{i-1} SL_j$ with $|G'| = k$.*

Proof. Assume that there is a k-point group $G''\subseteq \bigcup_{j=1}^{t} SL_j$ and G'' contains at least one point within SL_t for $t\geq i$. Additionally, G'' g-dominates $G\subseteq \bigcup_{j=1}^{i} SL_j$. Due to Definition 3.4, $G''\prec_g G$ if for each point $p''\in G''$, it holds that p'' dominates or is just equal to some point $p\in G$, and for at least one point p'', $p''\prec p$. However, the points $p\in G''\cap SL_t$ are dominated by or incomparable to some points in G for $t\geq i$. This contradicts to the assumption. Therefore, this lemma holds. □

In Fig. 1, for $\{h_1, h_3, h_5\}$ where $h_1\in SL_1,\ h_3\in SL_1$, and $h_5\in SL_2$, it is g-dominated by $\{h_1, h_2, h_3\}$ where each hotel is within SL_1. The hotel groups of size 3, which contain hotels within SL_i for $2\leq i\leq 3$, do not g-dominate $\{h_1, h_3, h_5\}$. For an instance, $\{h_1, h_3, h_4\}$ is incomparable to $\{h_1, h_3, h_5\}$ since h_5 is incomparable with h_4. For a unit group $u_4 = \{h_1, h_4\}$ of $h_4\in SL_2$, according to Lemma 4.2, we could generate two G-Skylines $u_4\cup u_2 = \{h_1, h_2, h_4\}$ and $u_4\cup u_3 = \{h_1, h_3, h_4\}$ by merging u_4 with the unit groups u_2 and u_3 which are the unit groups of the points within SL_1-u_4.

Due to Lemma 4.2, any group G, which contains at least one point within SL_i, may be only g-dominated by the groups $G'\subseteq \bigcup_{k=1}^{i-1} SL_k$. Besides, the groups of k points within the first skyline layer SL_1 could be returned as G-Skylines, directly, since they are dominated by no groups of equal size.

Theorem 4.3 (Verification of G-Skyline [22]). *Given a group $G = \{p_1, p_2, \ldots, p_k\}$, it is a G-Skyline group, if its corresponding unit group set $S = \cup_{i=1}^{k} u_i$ contains k points.*

Lemma 4.4. *Given a group $G \subseteq SL_i$ for $|G| \leq k$ and $1 \leq i \leq k$, we have $G' = G \cup ParSet(G)$ is a G-Skyline if $|G'| = k$. Moreover, G' with $|G'| > k$ is not a G-Skyline, and the points within G are not contained in any G-Skyline at the same time.*

Proof. This lemma holds due to Theorem 4.3. □

Continuing the example in Fig. 1 with $k = 3$, for $\{h_{10}\} \subseteq SL_3$, we have $\{h_{10}\} \cup ParSet(\{h_{10}\}) = \{h_3, h_6, h_{10}\}$ is a G-Skyline. Considering $\{h_4, h_5\} \subseteq SL_2$, it holds that $\{h_4, h_5\} \cup ParSet(\{h_4, h_5\}) = \{h_1, h_2, h_4, h_5\}$ is not a G-Skyline. And the hotels h_4 and h_5 are not contained in a G-Skyline simultaneously.

Lemma 4.5. *Given a point $p \in P$, the unit group u_p is a G-Skyline if $|u_p| = k$, and u_p could be pruned if $|u_p| > k$. Moreover, given a unit group set U with $1 \leq |U| \leq k$, $G = \bigcup_{u \in U} u$ is a G-Skyline if $|G| = k$, and G could be pruned if $|G| > k$.*

Proof. This lemma holds due to Theorem 4.3. □

According to Lemmas 4.1 and 4.2, we introduce the layered optimum strategy to the GSky query, and propose the following LU algorithm. In the LU algorithm, we generate the G-Skylines contain j points within SL_i for $1 \leq i \leq k$ and $1 \leq j \leq k-i+1$ respectively and directly.

Algorithm. As depicted in Algorithm 1, it takes the DSG DS and the group size k as the inputs. Line 1 first reports the unit groups whose sizes are equal to k as G-Skylines, and preprocesses DS by removing the nodes of points $p \in P$ and its children if $|u_p| \geq k$ due to Lemma 4.5. Later, Lines 3 to 5 compute the $start\_layer$. The G-Skylines contain at least one point within SL_{start_layer}. If $k \leq |SL_1|$, we have $start\_layer = 1$. Otherwise when $\sum_{t=1}^{i-1} |SL_t| < k \leq \sum_{t=1}^{i} |SL_t|$, it holds $start\_layer = i$. Line 6 checks whether $start\_layer$ is equal to 1. If it returns "yes", Lines 7 to 10 are executed to generate the G-Skylines that only contain points within the first skyline layer SL_1, and $start\_layer$ is updated as $start\_layer+1$. Lines 11 to 24 are an iteration procedure that generates the remaining G-Skylines. In an iteration, it builds the G-Skylines that contain just j points within SL_i for $1 \leq j \leq k-i+1$. Line 12 computes the unit set U_i that contains all the unit groups over the ith skyline layer SL_i. In Fig. 2, $SL_2 = \{h_4, h_5, h_6\}$, $u_4 = \{h_1, h_2, h_4\}$, $u_5 = \{h_2, h_5\}$, and $u_6 = \{h_3, h_6\}$. Therefore, we have $U_2 = \{u_4, u_5, u_6\} = \{\{h_1, h_2, h_4\}, \{h_2, h_5\}, \{h_3, h_6\}\}$. Line 13 builds unit group sets $U' \subseteq U_i$ with $|U'| = j$ for $1 \leq j \leq k-i+1$ due to Lemma 4.1. For each unit group set U', a new candidate group G is generated in Line 15. Lines 16 and 17 identify and report the groups G with $|G| = k$ as G-Skylines, and the ones of sizes larger than k are pruned based on Definition 3.5. To further improve the progressiveness, Line 18 sorts the left candidate groups G in non-increasing order of their sizes. Thereafter, Lines 19-24 aim to compute the G-Skylines based on each candidate group G. If there are some G-Skylines G' satisfying $G \subseteq G'$, then the groups $G'' = G' - G$ could be computed by invoking LU that takes the DSG DS' and $k-|G|$ as the inputs. Here DS' contains the points within $\bigcup_{t=1}^{i-1} SL_t - G$. Line 23 returns the groups G' of size k as G-Skylines. The groups G' with $|G'| > k$ are pruned due to Lemma 4.4.

To generate the G-Skylines that contain just j points within SL_i for $1 \leq j \leq k-i+1$ and $1 \leq i \leq k$, Line 13 of the LU algorithm generates unit group sets $U' \subseteq U_i$ of size j. It is noticeable that the G-Skylines may contain at most t points within SL_j where t is far less than $k-i+1$.

Algorithm 1. Layered Unit-Based (LU) Algorithm for the GSky Query

Input: A DSG DS and group size k
Output: G-Skylines
1: Report the unit groups u_p with $|u_p| = k$ as G-Skylines and preprocess DS by removing each node of point p with $|u_p| \geq k$ due to Lemma 4.5
2: Initialize $start\_layer \leftarrow 1$
3: **for** $i \leftarrow 1$ to k **do**
4: **if** $\sum_{t=1}^{i} |SL_t| \geq k$ **then**
5: $start\_layer \leftarrow i$ and break
6: **if** $start\_layer == 1$ **then**
7: Compute the unit group set $U_1 \leftarrow \{u_p | p \in SL_1\}$ that consists of unit groups of all the points within SL_1
8: Generate unit group sets $U' \subseteq U_1$ with $|U'| = k$
9: Report $G \leftarrow \bigcup_{u \in U'} u$ with $|G'| = k$ as G-Skylines due to Lemma 4.4
10: $start\_layer \leftarrow start\_layer+1$
11: **for** $i \leftarrow start\_layer$ to k **do**
12: Compute the unit group set $U_i \leftarrow \{u_p | p \in SL_i\}$ that consists of unit groups of all the points within SL_i
13: Generate unit group sets $U' \subseteq U_i$ of size j for $1 \leq j \leq k-i+1$ due to Lemma 4.1
14: **for** each unit group set U' **do**
15: Generate a new candidate group $G \leftarrow \bigcup_{u \in U'} u$
16: Report the groups G as G-Skylines if $|G| = k$
17: Delete the groups G if $|G| > k$
18: Sort the left candidate groups G of sizes less than k in non-increasing order of $|G|$
19: **for** Each candidate group G **do**
20: Compute a DSG DS' that consists of the points within $\bigcup_{t=1}^{i-1} SL_t - G$
21: Generate candidate groups $G'' \leftarrow$ LU(DS', $k-|G|$) due to Lemma 4.2
22: Build new candidate groups $G' \leftarrow G \cup G''$ for each G''
23: Report the groups G' as G-Skylines if $|G'| = k$ on basis of Lemma 4.4
24: Delete the groups G' if $|G'| > k$

Example. Going back to the example in Fig. 1 with $k = 3$, in Algorithm 1, it organizes the dataset by the DSG DS shown in Fig. 2. We first preprocess the DS by removing nodes of the points h_4, h_7, h_8, h_9, and h_{10} whose sizes of unit groups exceed 3. Additionally, the hotel groups $u_4 = \{h_1, h_2, h_4\}$ and $u_{10} = \{h_3, h_6, h_{10}\}$ are returned as G-Skylines due to Lemma 4.5. Afterward, by executing Lines 3 to 5 of the LU algorithm, it holds that $start\_layer = 1$. The hotel group $u_1 \cup u_2 \cup u_3 = \{h_1, h_2, h_3\}$ over SL_1 is generated as a G-Skyline, directly (Lines 7-9). Here $start\_layer$ is refreshed as 2. Then, Line 13 generates unit group sets $\{u_5\}$, $\{u_6\}$, and $\{u_5, u_6\}$ which include at most two units of the points within SL_2 due to Lemma 4.1. Next, new hotel groups $\{h_2, h_5\}$, $\{h_3, h_6\}$ and $\{h_2, h_3, h_5, h_6\}$ are computed by merging the hotels within the above unit group sets, respectively. Since the size of $\{h_2, h_3, h_5, h_6\}$ is larger than 3, it is pruned directly. Based on the unit $u_5 = \{h_2, h_5\}$, it generates $u_5 \cup u_1 = \{h_1, h_2, h_5\}$ and $u_5 \cup u_3 = \{h_2, h_3, h_5\}$ as G-Skylines by invoking the

LU algorithm recursively. Similarly, for the unit $u_6 = \{h_3, h_6\}$, it gets new G-Skylines $u_6 \cup u_1 = \{h_1, h_3, h_6\}$ and $u_6 \cup u_2 = \{h_2, h_3, h_6\}$.

Theorem 4.6. *The LU algorithm can return exactly GSky query results.*

Proof. This theorem means that there is no G-Skyline missed as an unqualified group (i.e., no false negative) and there is no unqualified group returned as a G-Skyline (i.e., no false positive). □

Suppose that there is a G-Skyline G of size k missed as an unqualified group. On this supposition, it holds that G is g-dominated by other G-Skyline G' or $|G| > k$ since in the LU algorithm, only the candidate groups of sizes larger than k are deleted as unqualified groups. The group G is generated by merging unit groups in the LU algorithm, and it is dominated by no other group of equal size due to Theorem 4.3. Therefore, if G is identified as an unqualified group, we have $|G| > k$. This contradicts the assumption that G is a G-Skyline of size k.

In the LU algorithm, it may generate some unqualified groups G whose sizes are larger than k. These unqualified groups are identified and pruned in Lines 17 or 24, and there is not any unqualified group G returned as a G-Skyline. This ensures no false positive.

As analyzed aforementioned, this theorem holds.

Theorem 4.7. *The time complexity of the LU algorithm is* $O(\binom{\hbar}{k} - \sum_{i=2}^{k}\binom{\hbar_i}{k})$, *where* $\hbar_i = |SL_i|$ *and* $\hbar = \sum_{i=1}^{k}\hbar_i$.

Proof. The G-Skylines of size k consist of the points within the first k skyline layers, and the maximum cardinality of the G-Skylines is $\binom{\hbar}{k}$. Moreover, each G-Skyline includes at most $k-i+1$ points of the ith skyline layer due to Lemma 4.1. The groups which include more than $k-i+2$ points within the ith skyline layer are not G-Sky lines for $1 \le i \le k$. The size of these groups could be computed as $\sum_{j=k-i+2}^{k}\binom{\hbar_i}{j} \times \binom{\hbar-\hbar_i}{k-j}$. Therefore, the maximum size of the G-Skylines is $\binom{\hbar}{k} - \sum_{i=2}^{k}\sum_{j=k-i+2}^{k}\binom{\hbar_i}{j} \times \binom{\hbar-\hbar_i}{k-j}$ where $\hbar = \sum_{i=1}^{k}\hbar_i$. □

The LU algorithm generates the G-Skylines directly. Since $\binom{\hbar}{k} - \sum_{i=2}^{k}\sum_{j=k-i+2}^{k}\binom{\hbar_i}{j} \times \binom{\hbar-\hbar_i}{k-j} < \binom{\hbar}{k} - \sum_{i=2}^{k}\binom{\hbar_i}{k}$, it holds that the time complexity of the LU algorithm is $O(\binom{\hbar}{k} - \sum_{i=2}^{k}\binom{\hbar_i}{k})$, and this theorem holds.

To improve the query performance of the GSky query, the LU algorithm is designed by the use of the layered optimum strategy. Besides, the LU algorithm generates the G-Skylines directly due to Lemmas 4.1 and 4.2. This brings significant reduction of redundant computation for generating and checking unqualified candidate groups.

5 The Index-Independent Algorithm for the GSky Query

For the skyline query and its variants, the index-based algorithms intuitively have superior performance than the index-independent algorithms [2], [23]. This is because they avoid accessing the entire dataset. However, the index-based algorithms are significantly limited by the indexing requirement. They are inappropriate to process the data that are dynamically produced (e.g., they arrive from a stream), and cannot be indexed [23], [24]. In this case, as well as the algorithms in [22], [32], [33], our LU algorithm cannot be utilized to process the GSky query directly. This is since they are all based on the pre-computed index, DSG. Moreover, the index-based algorithms for the GSky query suffer from the well-known curse of dimensionality and face memory management problems.

In this section, we investigate the GSky query over the data that cannot be indexed for the first time, and propose an index-independent algorithm, called SP, as an important supplement of the index-based algorithms.

Lemma 5.1. *Given two points* $p, p' \in P$, p' *is not contained in any G-Skyline if* $p \prec p'$ *and* $|u_p| \ge k$.

Proof. Since $p \prec p'$, for each $p'' \in u_p$, it holds that $p'' \prec p$ and $p'' \prec p'$. Therefore, each point within u_p dominates p'. Since $|u_p| \ge k$, it holds that $|u_{p'}| > k$, and p' will not be contained in any G-Skyline due to Lemma 4.5. As analysed above, this lemma holds. □

In the SP algorithm, we maintain a group pruning set $GPruSet \subseteq P$ with the following property.

1) For each $p \in GPruSet$, the size of u_p exceeds k.
2) If a point p' is dominated by someone within $GPruSet$, it is not a G-Skyline due to Lemma 5.1.
3) Given two points $p, p' \in GPruSet$, it holds $p \not\prec p'$.

The first property ensures that $GPruSet$ consists of unqualified points that are not contained in any G-Skyline. The second property means that any point dominated by someone within $GPruSet$ is not a G-Skyline, and could be pruned safely. The last property of $GPruSet$ plays a significant role in refining the points within $GPruSet$ due to the following lemma.

Lemma 5.2. *Given two points* $p \in GPruSet$ *and* $p' \in P$, *there is no need to add* p' *to* $GPruSet$ *and* p' *could be deleted directly if* $p \prec p'$.

Proof. This is since $p \prec p'$, for any points p'' dominated by p', we have $p \prec p''$ due to the transitive property of dominance operator [4]. Due to Lemma 5.1, p'' could be pruned by p safely. Therefore, any point pruned by p could be pruned by p' instead, and there is no need to add p' to $GPruSet$. □

Algorithm. In the SP algorithm as depicted in Algorithm 2, it takes the dataset P sorted by a monotone function and the group size k as the inputs. Here, we use the function $\min_{1 \le i \le d}(t.a_i)$ to sort the points within P and use $\sum_{1 \le i \le d}(t.a_i)$ as the tie breaker which is also utilized in [14]. After sorting the points $p \in P$, any point p is only dominated by the points ranked before it.

Line 1 initializes a candidate group set CG and a G-Skyline set $GSky$ with $\emptyset$. Besides, a set $GPruSet$ includes some points which are pruned but could be utilized to identify other unqualified points. Then, for each point $p \in P$, it first checks whether there is some point $p' \in GPruSet$ that dominates p. If it return "yes", p could be pruned due to Lemma 5.2. Otherwise, Lines 6 and 7 compute the parent set $ParSet(p)$ and create a unit group u_p by merging $\{p\}$ and $ParSet(p)$. If the size of u_p is no less than k, then the point p is inserted into $GPruSet$ and $GPruSet$ is refreshed by

removing the points that are dominated by p. Moreover, if $|u_p| = k$, u_p is added to $GSky$ as a G-Skyline. Line 12 deletes p from P due to Lemma 4.5. In case u_p contains less than k points, for each group $G \in CG$, we add a new group $G \cup \{u_p\}$ of size k to $GSky$ (Lines 15-16). Besides, CG is refreshed by adding u_p and the groups $G \cup \{u_p\}$ whose sizes are less than k (Lines 18-20). After the for loop (Lines 2-20), we gain all the G-Skylines that are stored in $GSky$.

Algorithm 2. Sorted-Based Progressive (SP) Algorithm for the GSky Query

Input: A dataset P sorted using a monotone function and the group size k
Output: A GSky query result set $GSky$
1: Initialize $CG = GSky \leftarrow \emptyset$ and $GPruSet \leftarrow \emptyset$
2: **for** each $p \in P$ **do**
3: **if** p is dominated by some point $p' \in GPruSet$ **then**
4: Delete p from P due to Lemma 5.2
5: **else**
6: Compute $ParSet(p)$
7: Generate unit group $u_p \leftarrow \{p\} \cup ParSet(p)$
8: **if** $|u_p| \geq k$ **then**
9: Insert p to $GPruSet$ and refresh $GPruSet$ by removing the points that are dominated by p
10: **if** $|u_p| = k$ **then**
11: Add u_p to $GSky$ as a G-Skyline
12: Delete p from P due to Lemma 4.5
13: **else**
14: **for** each group $G \in CG$ **do**
15: **if** $|G \cup u_p| == k$ **then**
16: Add $G \cup u_p$ to $GSky$ as a G-Skyline
17: **else**
18: **if** $|G \cup u_p| < k$ **then**
19: Add $G \cup u_p$ to CG
20: Add u_p to CG
21: Return $GSky$

In the LU algorithm, the unit group of each point p could be computed by $u_p = \{p\} \cup ParSet(p)$ where $ParSet(p)$ has been computed in advance and stored in the DSG. However, in the SP algorithm, when visiting a point, it needs to compute its parents and get unit group of the point in turn. The cost of computing the parents of points plays an important impact on performance of the SP algorithm. Therefore, we apply the group pruning set $GPruSet$ to identify and prune unqualified groups as soon as possible. For a point p pruned by some one within $GPruSet$, the groups containing p are unqualified and unnecessary to be generated.

Example. We illustrate the SP algorithm for the GSky query on the hotel dataset in Fig. 1. The SP algorithm takes a hotel set $\{h_3, h_6, h_1, h_{10}, h_2, h_9, h_5, h_4, h_8, h_7\}$ where the hotels are ranked in non-decreasing order of their minimum attribute values as its input. The hotel h_3 is visited first and we have $CG = \{h_3\}$. For the hotel h_6, it holds that $u_6 = \{h_3, h_6\}$ and $CG = \{\{h_3\}, \{h_3, h_6\}\}$. After visiting h_1, we get the G-Skyline $\{h_1, h_3, h_6\}$ and $CG = \{\{h_1\}, \{h_3\}, \{h_1, h_3\}, \{h_3, h_6\}\}$. Next, $u_{10} = \{h_3, h_6, h_{10}\}$ is added to $GSky$ as a G-Skyline and h_{10} is inserted to $CPruSet$. In the same way, taking into account the hotels h_2, h_9, and h_5, we gain the G-Skylines $\{h_1, h_2, h_3\}$, $\{h_2, h_3, h_6\}$, $\{h_1, h_2, h_5\}$, and $\{h_2, h_3, h_5\}$, and h_9 is pruned and added to $GPruSet$. Then, since $|u_4| = |\{h_1, h_2, h_4\}| = 3$, u_4 is added to $GSky$ and h_4 is inserted to $GPruSet$. Now, we have $GPruSet = \{h_{10}, h_9, h_4\}$. Thereafter, h_8 is pruned and added to $GPruSet$ because $|u_8| = 5 > 3$ and h_8 is not dominated by any hotel within $GPruSet$. At last, for the hotel h_7 is dominated by $h_4 \in GPruSet$, it cannot be contained in any G-Skyline and is pruned directly due to Lemma 5.2.

Theorem 5.3. *The SP algorithm can return exactly GSky query results.*

Proof. The proof of this theorem is similar to that of Theorem 4.6. If there is a G-Skyline G missed as an unqualified group, it must be pruned by some point within $GPruSet$ due to Lemma 5.2 or its size is larger than k and pruned by Lemma 4.5. Hence G is not a G-Skyline which contradicts the assumption. Additionally, any unqualified group G of size larger than k will be pruned by Lemma 5.2 or Lemma 4.5, and there is not any unqualified group G returned as a G-Skyline. As analyzed above, this theorem holds. □

Theorem 5.4. *The time complexity of the SP algorithm is $O(N^2 + N \times |CG|)$ where N is the cardinality of the dataset P and CG consists of the candidate groups of sizes less than k.*

Proof. The SP algorithm needs to take all the points within P into account. Let N represent the cardinality of the dataset P. For the ith point $p \in P$ where $1 \leq i \leq N$, it first checks whether there is some point $p' \in GPruSet$ dominating p. This costs $O(|GPruSet|)$. If p is not pruned by any point in $GPruSet$, it computes its parent set by visiting all the points ranked before it. The time cost to compute the parents is $O(i-1-|GPruSet|)$. After that, the candidate group set CG is updated for each $G \in CG$. This costs $O(|CG|)$ where CG includes all the candidate groups whose sizes are less than k. Hence, the time cost of SP is

$$O\Big(\sum_{i=1}^{N}\Big(|GPruSet| + (i-1-|GPruSet|) + |CG|\Big)\Big) = O\Big(N^2 + N \times |CG|\Big).$$

Therefore, this theorem holds. □

6 Performance Evaluation

Our experiments use both synthetic datasets and real-life datasets. A number of experiments have been completed on synthetic datasets with two popular distributions: Independent (Ind) and Anti-correlated (Ant) following the work in [4]. Furthermore, we deploy the real dataset, NBA, which is also adapted in [22]. Specifically, NBA contains 17,265 points. We consider 3 attributes that are the numbers of points scored, rebounds, and assists. We also evaluate our proposed algorithms over the real dataset, Household (Hou) [12], [36]. It includes 127,000 tuples about the percentage of an American familys annual income. We consider 4 attributes, which are the expenditures of gas, electricity, water, and heating. We will consider the UWise+ algorithm [22], which is the state of the art algorithm for the GSky query to the best of our knowledge, as the baseline algorithm. Additionally, this is the first work for the GSky query

TABLE 2
Partial Running Information for SP over the Hotel Dataset in Fig. 1

Hotel	New groups in CG	New G-Skylines in $GSky$
h_3	$\{h_3\}$	
h_6	$\{h_3, h_6\}$	
h_1	$\{h_1\}, \{h_1, h_3\}$	$\{h_1, h_3, h_6\}$
h_{10}		$\{h_3, h_6, h_{10}\}$
h_2	$\{h_2\}, \{h_1, h_2\}, \{h_2, h_3\}$	$\{h_1, h_2, h_3\}, \{h_2, h_3, h_6\}$
h_5	$\{h_2, h_5\}$	$\{h_1, h_2, h_5\}, \{h_2, h_3, h_5\}$
h_4		$\{h_1, h_2, h_4\}$

TABLE 3
System Parameters

Parameter	Values
Data set dimensionality(d)	2, **3**, 4, 5
Group size (k)	2, **3**, 4, 5
Data set cardinality (N)	20000, **40000**, 60000, 80000, 100000

without an index, and we conduct the SP algorithm against a baseline algorithm, namely Base. In the Base algorithm, it computes all the unit groups of points within the dataset, sorts the unit groups in nondecreasing order of their sizes, and combines the unit groups to compute the G-Skylines. Moreover, we compare the SP algorithm with the one, namely SP-, without adopting Lemma 5.2 and the pruning set $GPruSet$. This can help to certify the pruning capacity of Lemma 5.2 and the set $GPruSet$.

In the following experiments, we evaluate the algorithms for the GSky query from the following two aspects.

- Query time (QT). The time between submitting a query request and returning all the answers.
- Progressiveness. It is evaluated by the query time as a function of the number of the GSky query results reported.

Moreover, the number of points within the first k skyline layers (NS) and the number of the GSky query results (NR) are also reported.

As well as theoretical proofs about the correctness of the proposed algorithms (Theorems 4.6 and 5.3), we also illustrate the query correctness with experimental demonstration. We design a programm to analyze the results returned by our proposed algorithms and the UWise+ algorithm. Many times of experiment show that, for the same datasets, the G-Skylines retrieved by the above algorithms are just the same.

TABLE 4
Experimental Results versus d

d	Ind			Ant		
	NS	LU (s)	UWise+(s)	NS	LU (s)	UWise+(s)
2	60	**0.0005**	**0.0003**	234	0.0021	0.0103
3	369	0.0019	0.0066	3138	0.2206	7.2249
4	1634	0.0212	0.3461	15575	26.4692	1078.3400
5	5571	0.5542	19.8756	30861	524.7570	21917.4000

All the experiments were conducted on a PC with Intel (R) Core(TM) I5-3330S 2.7 GHz CPU (contains 4 cores) and 4GB main memory running windows 7 operating system. All the algorithms were implemented in C++. In order to compare the results in different scenarios, we vary one of the parameters shown in Table 3 each time where the rest ones are fixed to their default values as highlighted in bold.

6.1 Performance of the Index-Based Algorithm

In this subsection, we analyze the performance of the index-based algorithms for the GSky query in terms of QT and progressiveness.

6.1.1 Performance on Synthetic Datasets

Experimental Results by Varying d. By varying d, NS and QT are shown in Table 4, and NR of the GSky query is depicted in Fig. 4a. The dimensionality d has a great impact on the performance of the LU and UWise+ algorithms. When d increases, NS, the number of points within the first $k = 3$ skyline layers, grows sharply. Due to Equation (1), the cardinality of the ith skyline layer SL_i for $1 \leq i \leq k$ increases exponentially with the growth of d, and NS grows in an exponential rate. The larger NS is, the larger NR is, and the more candidate groups need to be evaluated which in turn brings the growth of QT of the LU and UWise+ algorithms. Besides, the NS, NR, and QT of the LU and UWise+ algorithms over the Ant datasets are all much larger than those over the Ind datasets.

Table 4 shows that our LU algorithm is much better than the UWise+ algorithm in terms of QT. When processing the GSky query over the Ind datasets, it reduces 97.21% QT in the best case. Over the Ant datasets, it can cut down 97.61 percent QT at most. This is since LU adopts the layered optimum strategy and generates the query results

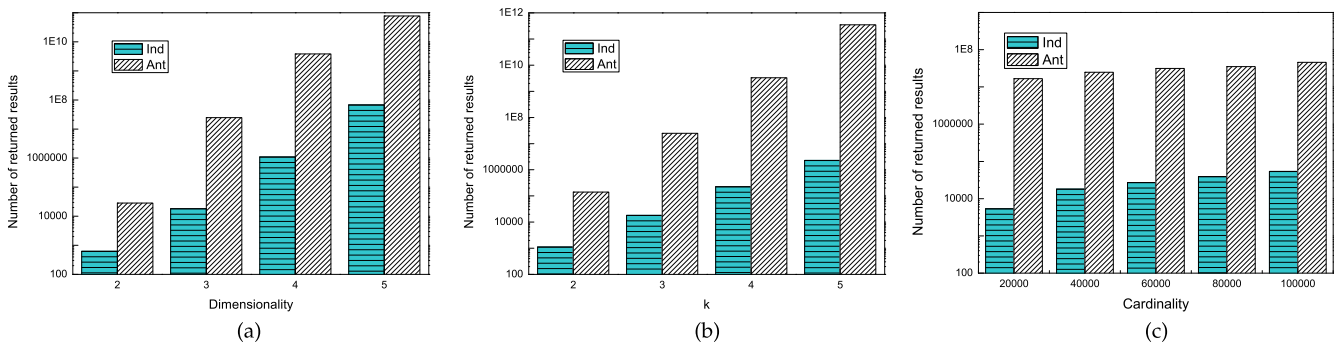


Fig. 4. Number of the GSky query results (NR). (a) d. (b) k. (c) N.

TABLE 5
Experimental Results versus k

k	Ind			Ant		
	NS	LU (s)	UWise+(s)	NS	LU (s)	UWise+(s)
2	177	0.0001	0.0006	1614	0.0013	0.0540
3	369	0.0017	0.0068	3138	0.2274	7.6742
4	634	0.0123	0.0835	5087	27.9916	1073.8900
5	943	0.1314	0.9458	7365	2911.7100	$> 24\mathbf{h}$

TABLE 6
Experimental Results versus N

N	Ind			Ant		
	NS	LU (s)	UWise+(s)	NS	LU (s)	UWise+(s)
20000	270	0.0012	0.0025	2575	0.1625	5.1836
40000	369	0.0019	0.0066	3138	0.2262	7.7928
60000	421	0.0022	0.0109	3431	0.2671	9.6701
80000	445	0.0025	0.0140	3711	0.3062	10.9869
100000	484	0.0031	0.0199	3916	0.3833	14.1110

directly. Therefore, it brings significant reduction of redundant computation for generating and identifying unqualified candidate groups comparing to the UWise+ algorithm.

Experimental Results by Varying k. The experimental results of NS and QT by varying k are shown in Table 5. Fig. 4b shows NR of the GSky query with increasing k. Similar to the dimensionality d, k affects the performance of the LU and UWise+ algorithms significantly. As k grows, NS, NR, and cardinality of candidate groups increase significantly. This makes QT of the LU and UWise+ algorithms grows sharply. Additionally, the Ant datasets have larger NS and NR, and need much more QT than the Ind datasets.

Again, our LU algorithm outperforms the UWise+ algorithm in terms of QT. For the GSky queries over the Ind datasets, it reduces 86.10% QT in the best case. Over the Ant datasets, our LU algorithm cuts down 97.52% QT at most. As depicted in Table 5, the QT of our LU algorithm is 2911.7100s ($<$1h) when processing the Ant dataset with $k = 5$. However, the QT of the UWise+ algorithm has exceeded 24h which becomes unacceptable.

Experimental Results by Varying N. Table 6 presents NS and QT of the LU and UWise+ algorithms, and Fig. 4c shows NR with varying N over the Ind and Ant datasets. With comparing to d or k, NS, NR, and QT of the LU and UWise+ algorithms have not been significant impacted by increasing N. As depicted in Table 6, NS increases as N grows. This makes a slight growth of the number of the GSky query results NR and QT of the LU and UWise+ algorithms. Besides, over the Ind datasets, comparing with the UWise+ algorithm, it reduces 84.37% QT at best by utilizing our LU algorithm. Over the Ant datasets, our LU algorithm cuts down 97.28% QT in the best case.

6.1.2 Progressiveness Performance

In this set of experiments, we evaluate progressiveness of the index-based algorithms, LU and UWise+, over the Ind and Ant datasets. We analyze the progressiveness of the algorithms through evaluating the QT with varying NR which is the number of results returned. The accumulated QT of the LU and UWise+ algorithms over the Ind and Ant datasets are reported as NR grows.

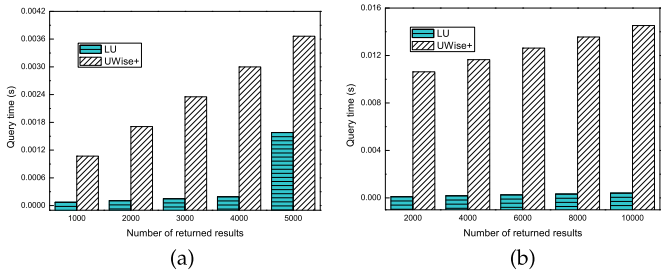


Fig. 5. Progressiveness comparison. (a) Ind. (b) Ant.

From Fig. 5, we note that when it returns the same number of results, our LU algorithm always needs much less QT than that of the UWise+ algorithm. Compared to the UWise+ algorithm, to get the GSky query results of the same size, our LU algorithm can reduce 93.81% QT at most for the Ind datasets. Besides, for the Ant datasets, the LU algorithm can reduce 99.05% QT at best. This indicates our LU algorithm has much better progressiveness than the UWise+ algorithm.

6.2 Performance of the Index-Independent Algorithms

In this subsection, we report the experimental results of the index-independent algorithms. Here, due to the experimental environment, the defaults of N and d are adjusted to 20000 and 2, respectively.

6.2.1 Performance on Synthetic Datasets

Experimental Results by Varying d. By varying d, Figs. 6a and 7a show QT of the SP, SP-, and Base algorithms. The dimensionality d is an important issue which has a significant impact on the performance of the GSky query. Similar to the analysis in Section 6.1.1, QT of the proposed algorithms increases as d grows. This is due to the exponential increase of candidate groups with increasing d. The Ant datasets need much more QT than that over the Ind datasets with the same dimensionality d.

Among the three index-independent algorithms, the SP algorithm has the best performance in most cases as shown in Figs. 6a and 7a. This is attributed to the pruned set $GPruSet$ and Lemmas 5.1 and 5.2. However, over the Ant dataset with $d = 5$, the SP- algorithm needs a little less QT than that of the SP algorithm. This is reasonable because a high-dimensional point is less likely to be dominated and pruned by the points within the set $GPruSet$, and many unqualified points are added to $GPruSet$. The larger $GPruSet$ is, the more cost of dominance test needs.

Experimental Results by Varying k. The experimental results of QT with varying k are shown in Figs. 6b and 7b. The QT of the algorithms increases when k grows. This is since the number of candidate groups grows rapidly as k grows. Besides, it always needs much more QT to process the Ant datasets with comparing to the Ind datasets. When dealing with the GSky query over the Ind dataset with $k = 2$, QT of the Base algorithm is a little less than that of the SP algorithm. However, with the growth of k, the SP algorithm has the best performance in terms of QT as shown in Figs. 6b and 7b. This also confirms the effectiveness of the pruned set $GPruSet$ and Lemmas 5.1 and 5.2.

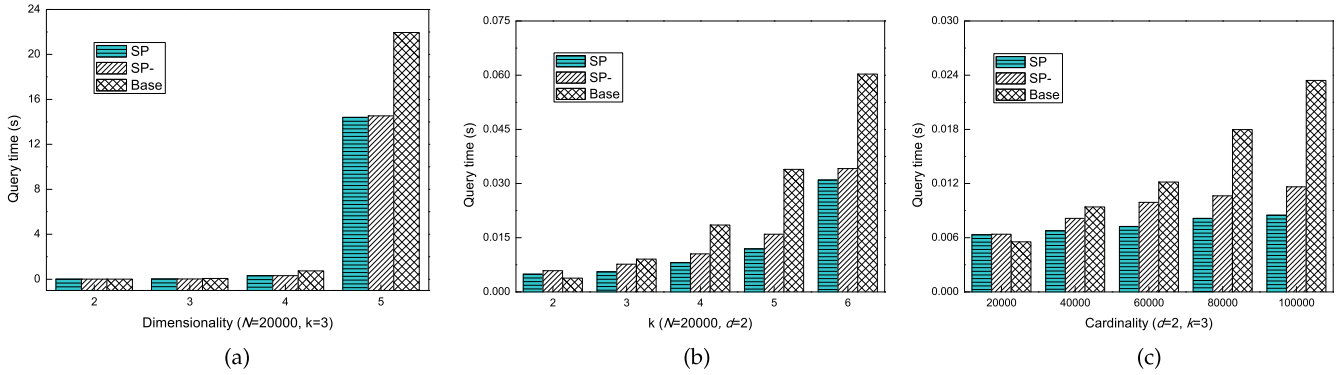


Fig. 6. Experimental results over the Ind datasets. (a) d. (b) k. (c) N.

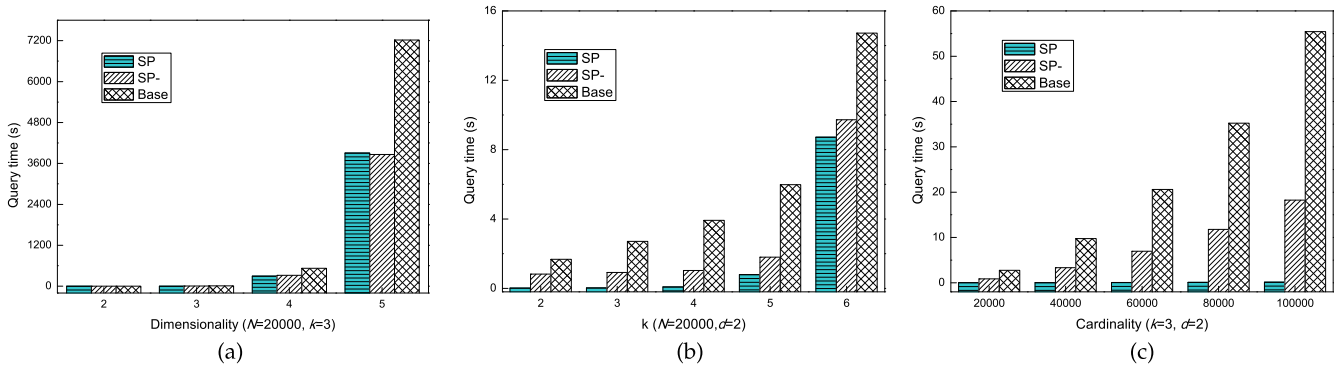


Fig. 7. Experimental results over the Ant datasets. (a) d. (b) k. (c) N.

Experimental Results by Varying N. Figs. 6c and 7c present QT of the SP, SP-, and Base algorithms with varying N. As depicted in Figs. 6c and 7c, QT increases with the increase of N. This is also as expected since the larger N is, the much more candidate groups need to be considered in the GSky query. It requires much more QT to process the Ant datasets with comparing to the Ind datasets. In terms of QT, the Base algorithm is the best when $N = 20000$ over the Ant dataset. However, with varying N, the SP algorithm requires the minimal QT as shown in Figs. 6c and 7c.

6.2.2 Progressiveness Performance

In this set of experiments, we evaluate the progressiveness of the index-independent algorithms.

Table 7 reports the QT of the SP, SP-, and Base algorithms over the Ind and Ant datasets, respectively. Here NR represents the number of returned results. From Table 7, we observe that when it returns the same number of the GSky query results, the SP algorithm needs much less QT, comparing to the Base algorithm. This indicates the SP algorithm has better progressiveness than the Base algorithm. Additionally, the Base algorithm has the worst progressiveness, this is also as expected. It needs to compute all the unit groups of points within the datasets at first, which has a significant impact on the progressiveness of the Base algorithm. The progressiveness of SP is better than that of SP- over Ant datasets, and for Ind datasets, QT of the two algorithms is comparable.

TABLE 7
Progressiveness Comparison over the Synthetic Datasets

NR	Ind (N=4000, d=3, k=3)			Ant (N=2000, d=2, k=3)		
	SP(s)	SP- (s)	Base(s)	SP(s)	SP- (s)	Base(s)
2000	0.0009	0.0008	0.0616	0.0018	0.0040	2.6665
4000	0.0015	0.0014	0.0622	0.0038	0.0186	2.6672
6000	0.0019	0.0018	0.0630	0.0063	0.0447	2.6679
8000	0.0024	0.0022	0.0637	0.0075	0.0619	2.6686
10000	0.0029	0.0027	0.0643	0.0092	0.0833	2.6693

6.3 Performance on the Real Datasets

In this subsection, we report the experimental results on the real datasets, NBA and Hou, with $k = 3$. Table 8 shows QT of the proposed algorithms for the GSky query over the NBA and Hou datasets, respectively. The results on the real datasets are consistent with the ones obtained from the experiments on the synthetic datasets. From Table 8, the LU algorithm requires slightly more QT than the UWise+ algorithm over the NBA dataset which is a small dataset, but it is better than the UWise+ algorithm over the Hou dataset. The SP algorithm again requires the minimal QT among the three index-independent algorithms.

The cardinality of the NBA dataset is too small to show the difference in the progressiveness of the algorithms for the GSky query. Therefore, we only analyze the

TABLE 8
Experimental Results over the Real Datasets

Dataset	Index-based Algs.		Index-independent Algs.		
	LU(s)	UWise+(s)	SP(s)	SP-(s)	Base(s)
NBA	0.0006	0.0002	0.0649	0.0094	0.0145
Hou	0.0657	1.2745	0.9148	0.9436	2.1761

TABLE 9
Progressiveness Comparison over the Hou Dataset

NR	Index-based Algs.		Index-independent Algs.		
	LU(s)	UWise+(s)	SP(s)	SP-(s)	Base(s)
2000	0.00006	0.02611	0.00079	0.00091	0.94281
4000	0.00012	0.02707	0.00113	0.00124	0.92874
6000	0.00019	0.02808	0.00162	0.00170	0.92975
8000	0.00027	0.02908	0.00227	0.00234	0.93075
10000	0.00032	0.03008	0.00277	0.00286	0.93175

progressiveness of the algorithms for the GSky query over the Hou dataset. From Table 9, we observe that our LU algorithm has much better progressiveness than the UWise+ algorithm, and the SP algorithm is the best index-independent algorithm for the GSky query in terms of progressiveness.

7 Extensions

In this section, we explore how to continuously maintain the GSky query results when data updates happen. Besides, we propose an important variant of the GSky query, namely TlGSky, which aims to return l G-Skylines based on a novel ranking criterion.

7.1 Maintain the GSky Query Results

In practical applications, the dataset is usually updated. Noticeably, in the literature, there is no work about the GSky query when data updates happen. In this section, we develop an approach of maintaining the GSky query results, namely MGSky.

Generally speaking, an update operator includes insert and delete operators. If there is a new point p, to get the accurate G-Skylines after the insert operator, the old G-Skylines containing p' dominated by the point p are removed from the GSky query result set, and new G-Skylines including p need to be rebuilt (The new G-Skylines G containing p' satisfy $p{\in}G$ because $p{\prec}p'$). Suppose that a point p is deleted from the dataset P. The old G-Skylines including the point p are deleted (For the old G-Skylines G containing the points p' that are dominated by p, it holds that p is also included in G), and it is necessary to regenerate the G-Skylines including p' afterwards.

In the above insert and delete operations, they need to generate G-Skylines containing the point p updated. Assume that $p{\in}SL_i$ for $1{\leq}i{\leq}k$. The MGSky algorithm generates unit group sets $U'{\subseteq}U_i$ with $u_p{\in}U'$ and $1{\leq}|U'|{\leq}k{-}i{+}1$. This ensures each G-Skyline gained contains the point p. Based on the unit group sets U', some G-Skylines G are generated in a bottom up fashion by invoking Lines 14 to 24 of the LU algorithm. The left part of the MGSky algorithm is an iteration procedure. In each iteration, we compute $ChiSet(p,j)$ that includes children of the point p in SL_j for $i{+}1{\leq}j{\leq}k$. Next, it generates unit group sets $U'{\subseteq}U_j$ that contain at least one of the units $u_{p'}$ of the points $p'{\in}ChiSet(p,j)$. Since p' is a child of p, G-Skylines that contain p' include p simultaneously due to Lemma 4.4. For each U', we could get new G-Skylines containing the point p by applying Lines 14 to 24 of the LU algorithm. Thereafter, for unit group sets $U'{\subseteq}U_j$ that do not include any $u_{p'}$ with $p'{\in}ChiSet(p,j)$, we generate candidate groups $G{\leftarrow}\bigcup_{u\in U'\cup\{u_p\}}u$. Similarly, by invoking Lines 16 to 24 of the LU algorithm, we could get the left G-Skylines that contain the point p.

7.2 A GSky Query with a Size Constraint

As introduced in [22], the GSky query faces a big limitation which is it usually reports too many results to draw a meaningful insight. In addition, a large number of G-Skylines could create emotional stress to users and act as powerful barriers to rational decisions [37]. In the experiments, for the anti-correlated dataset with $k=2$, $d=3$, and $N=40000$, it reports 140,078 G-Skylines. Besides, the GSky query over the anti-correlated dataset with $k=3$, $d=3$, and $N=40000$, returns 24,756,789 G-Skylines.

Liu et al. [22] developed an interesting variant of the GSky query, namely PG-Skyline. It gets a subset of the GSky query results by relaxing the dominance requirement. Although this approach is useful to limit the GSky query results, it is difficult for users to specify an appropriate parameter p and the query results are not controllable. In this subsection, we investigate the GSky query with a size constraint to gain G-Skylines of a manageable size.

Search result diversification has become important for improving user satisfactory. To meet as many requirements as possible for users, we investigate the GSky query with a size constraint based on the diversify of different G-Skylines. In [19], a diversity function δ was proposed to measure the diversity of a pair of skyline points. The function δ is scale invariant and monotone-robust. However, it cannot be utilized to measure the diversification of different G-Skylines.

In the following, we extend the function δ and propose a new ranking function $\delta(G,G')$, which is defined as

$$\delta(G,G')=\sum_{p\in G}\min_{p'\in G'}\delta(p,p'). \tag{2}$$

Here $\delta(p,p')=$

$$\frac{1}{d}\sum_{j\in[1,d]}\frac{\left|\{o\in\cup_{i\in[1,k]}SL_i|(p_j{\leq}o_j{\leq}p'_j){\cup}(p'_j{\leq}o_j{\leq}p_j)\}\right|-1}{|\cup_{i\in[1,k]}SL_i|-1},$$

SL_i is a set that includes all the points within the ith skyline layer.

Based on the new ranking function in Equation (2), we formulate the TlGSky query which returns representative G-Skylines of a manageable size.

Definition 7.1 (Top l Group-based Skyline Query, TlGSky). *Given a dataset P, two parameters k and l, the l-GSky query returns a set GS including l G-Skylines G that have the highest scores Score(GS) as given in*

$$Score(GS)=\sum_{G\in GSky}\min_{G'\in GSky-\{G\}}\delta(G,G').$$

Here $\delta(G,G')$ is given in Equation (2).

8 Conclusions

In this paper, we investigate the GSky query which aims to report the G-Skylines dominated by no other group of the same size. We first develop a new index-based algorithm, called LU, for the GSky query. The LU algorithm has much better progressiveness and efficiency compared to

the state-of-the-art algorithm for the GSky query. Additionally, we focus on the GSky query over datasets that cannot be indexed and develop an index-independent algorithm. Last but not least, we explore how to continuously maintain the GSky query results when data updates happen and discuss an interesting extension of the GSky query, called T*l*GSky, based on the new ranking criterion.

This work opens to some promising directions for future work. First, it is of interest to develop efficient algorithms for the GSky queries over sliding windows [5], [38]. Then, uncertain data analysis is significant in many important applications [13], [14], [39], and it is significant to research the GSky query over uncertain data. After that, to further improve the query performance, it is worth to research the GSky query under distributed or multi-core environments based on the previous work in [8], [9], [10], [40]. Last, it is a significant but challenging work to develop effective algorithms for the T*l*GSky query.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable and helpful comments on improving the manuscript. The research was partially funded by the National Natural Science Foundation of China (Grant Nos. 61772182, 61472126, 61602170), the Key Program of National Natural Science Foundation of China (Grant No. 61432005), the International (Regional) Cooperation and Exchange Program of National Natural Science Foundation of China (Grant No. 61661146006), the Emergency Special Project of National Natural Science Foundation of China (Grant No. 61751204), the National Key R&D Program of China (Grant No. SQ2018YFB020061), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), the National High-tech R&D Program of China (Grant No. 2015AA015303), China Postdoctoral Science Foundation (Grant No. 2016M602410), and the Science and Technology Plan of Changsha (K1705032).

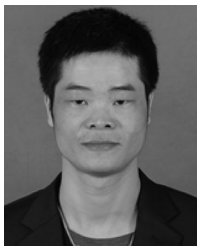
References

[1] H. Im and S. Park, "Group skyline computation," *Inf. Sci.*, vol. 188, pp. 151–169, 2012.
[2] 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.
[3] 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.
[4] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. Int. Conf. Data Eng.*, 2001, pp. 421–430.
[5] X. Lin, Y. Yuan, W. Wang, and H. Lu, "Stabbing the sky: Efficient skyline computation over sliding windows," in *Proc. 21th Int. Conf. Data Eng.*, 2005, pp. 502–513.
[6] Y. Gao, Q. Liu, B. Zheng, L. Mou, G. Chen, and Q. Li, "On processing reverse k-skyband and ranked reverse skyline queries," *Inf. Sci.*, vol. 293, pp. 11–34, 2015.
[7] M. S. Islam, R. Zhou, and C. Liu, "On answering why-not questions in reverse skyline queries," in *Proc. 29th Int. Conf. Data Eng.*, 2013, pp. 973–984.
[8] D. Pertesis and C. Doulkeridis, "Efficient skyline query processing in spatialhadoop," *Inf. Syst.*, 2014.
[9] X. Han, J. Li, D. Yang, and J. Wang, "Efficient skyline computation on big data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 11, pp. 2521–2535, 2013.
[10] Y. Park, J.-K. 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.
[11] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 15–26.
[12] X. Zhou, K. Li, Y. Zhou, and K. Li, "Adaptive processing for distributed skyline queries over uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 371–384, Feb. 2016.
[13] X. Ding and H. Jin, "Efficient and progressive algorithms for distributed skyline queries over uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 8, pp. 1448–1462, Aug. 2012.
[14] X. Liu, D. N. Yang, M. Ye, and W. C. Lee, "U-skyline: A new skyline query for uncertain databases," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 945–960, Apr. 2013.
[15] H. Lu, C. S. Jensen, and Z. Zhang, "Flexible and efficient resolution of skyline query size constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 7, pp. 991–1005, Jul. 2011.
[16] Z. Huang, Y. Xiang, and Z. Lin, "l-skydiv query: Effectively improve the usefulness of skylines," *Sci. China Inf. Sci.*, vol. 53, no. 9, pp. 1785–1799, 2010.
[17] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-based representative skyline," in *Proc. 25th Int. Conf. Data Eng.*, 2009, pp. 892–903.
[18] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in *Proc. 23th Int. Conf. Data Eng.*, 2007, pp. 86–95.
[19] M. Magnani, I. Assent, and M. L. Mortensen, "Taking the big picture: representative skylines based on significance and diversity," *VLDB J.*, vol. 23, no. 5, pp. 795–815, 2014.
[20] X. Miao, Y. Gao, S. Guo, and G. Chen, "On efficiently answering why-not range-based skyline queries in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1697–1711, Sep. 2018.
[21] X. Miao, Y. Gao, L. Zhou, W. Wang, and Q. Li, "Optimizing quality for probabilistic skyline computation and probabilistic similarity search," *IEEE Trans. Knowl. Data Eng*, vol. 30, no. 9, pp. 1741–1755, Sep. 2018.
[22] 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, 2015.
[23] J. Lee and S. W. Hwang, "Scalable skyline computation using a balanced pivot selection technique," *Inf. Syst.*, vol. 39, no. 1, pp. 1–21, 2014.
[24] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Trans. Database Syst.*, vol. 33, no. 4, pp. 1–49, 2008.
[25] I.-F. Su, Y.-C. Chung, and C. Lee, "Top-k combinatorial skyline queries," in *Database Systems for Advanced Applications*. New York, NY, USA: Springer, 2010, pp. 79–93.
[26] Y.-C. Chung, I.-F. Su, and C. Lee, "Efficient computation of combinatorial skyline queries," *Inf. Syst.*, vol. 38, no. 3, pp. 369–387, 2013.
[27] 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.
[28] 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.
[29] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng, "Creating competitive products," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 898–909, 2009.
[30] T. Jiang, B. Zhang, D. Lin, Y. Gao, and Q. Li, "Incremental evaluation of top-k combinatorial metric skyline query," *Knowl.-Based Syst.*, vol. 74, pp. 89–105, 2015.
[31] X. Zhou, K. Li, Z. Yang, and K. Li, "Finding optimal skyline product combinations under price promotion," *IEEE Trans. Knowl Data Eng*, vol. 31, no. 1, pp. 138–151, Jan. 2019.
[32] C. Wang, C. Wang, G. Guo, X. Ye, and P. Yu, "Efficient computation of g-skyline groups," *IEEE Trans. Knowl. Data Eng*, vol. 30, no. 4, pp. 674–688, Apr. 2018.
[33] W. Yu, Z. Qin, J. Liu, L. Xiong, X. Chen, and H. Zhang, "Fast algorithms for pareto optimal group-based skyline," in *Proc. 26th Int. Conf. Inf. Knowl. Manage.*, 2017, pp. 417–426.
[34] X. Zhou, K. Li, G. Xiao, Y. Zhou, and K. Li, "Top k favorite probabilistic products queries," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 10, pp. 2808–2821, Oct. 2016.
[35] L. Zhu, Y. Tao, and S. Zhou, "Distributed skyline retrieval with low bandwidth consumption," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 3, pp. 384–400, Mar. 2009.
[36] 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.

[37] X. Zhao, Y. Wu, W. Cui, X. Du, Y. Chen, Y. Wang, D. L. Lee, and H. Qu, "Skylens: Visual analysis of skyline on multi-dimensional data," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 246–255, Jan. 2018.
[38] 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.
[39] G. Xiao, K. Li, and K. Li, "Reporting l most influential objects in uncertain databases based on probabilistic reverse top- k queries," *Inf. Sci.*, vol. 405, pp. 207–226, 2017.
[40] L. Chen, B. Cui, and H. Lu, "Constrained skyline query processing against distributed data sites," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 2, pp. 204–217, Feb. 2011.

Xu Zhou received the master's degree from the Department of Information Science and Engineering, Hunan University, in 2009. She is currently a postdoctoral with the Department of Information Science and Engineering, Hunan University, Changsha, China. Her research interests include parallel computing and data management.

Kenli Li received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. His major research areas include parallel computing, high-performance computing, and grid and cloud computing. He has published more than 200 research papers in international conferences and journals such as the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, and ICPP. He is a senior member of the IEEE and serves on the editorial board of the *IEEE-TC*.



Zhibang Yang received the PhD degree in computer science from Hunan University, Changsha, China, in 2012. He is currently an associate professor with Changsha University. His major research contains data management and parallel computing.

Guoqing Xiao received the PhD degree from the College of Information Science and Engineering, Hunan University, Changsha, China, in 2017. He is currently a postdoctoral with the College of Information Science and Engineering, Hunan University. His research interests include data management and parallel and distributed processing.

Keqin Li is a SUNY distinguished professor of computer science with the State University of New York. His current research interests include parallel computing and distributed computing. He has published more than 560 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.