Knowledge and Information Systems (2021) 63:1471–1496
https://doi.org/10.1007/s10115-021-01562-8

REGULAR PAPER

Progressive approaches to flexible group skyline queries

Zhibang Yang[1] · Xu Zhou[2] · Kenli Li[2] · Yunjun Gao[4] · Keqin Li[2,3]

Received: 14 August 2019 / Revised: 6 March 2021 / Accepted: 13 March 2021 / Published online: 9 April 2021

Abstract
The G-Skyline (GSky) query is formulated to report optimal groups that are not dominated by any other group of the same size. Particularly, a given group G_1 dominates another group G_2 if for any point $p \in G_1$, p dominates or equals to points $p' \in G_2$; at the same time, there is at least one point p dominating p'. Most existing group skyline queries need to calculate an aggregate point for each group. Compared to these queries, the GSky query is more practical because it avoids specifying an aggregate function which leads to miss important results containing non-skyline points. This means the GSky query can get much more comprehensive query results which not only contain the G-Skylines consisting of skyline points but also the G-Skylines including non-skyline points. Here, a non-skyline point is dominated by another point in a given data set. However, the GSky query usually returns too many results, making it a big burden for users to pick out their expected results. To address these issues, we investigate a flexible group skyline query, namely Flexible G-Skyline (FGSky) query, which is flexible and practical for directly computing the optimal groups on the basis of user preferences. In this paper, we formulate the FGSky query, identify its properties, and present effective pruning strategies. Besides, we propose progressive algorithms for the FGSky query where a grouping strategy and a layered strategy are utilized to get better query performance. Through

✉ Xu Zhou
zhxu@hnu.edu.cn

Zhibang Yang
yangzb@ccsu.edu.cn

Kenli Li
lkl@hnu.edu.cn

Yunjun Gao
gaoyj@zju.edu.cn

Keqin Li
lik@newpaltz.edu

1 Hunan Province Key Laboratory of Industrial Internet Technology and Security, Changsha University, Changsha 410003, Hunan, China

2 School of Information Science and Engineering, Hunan University, Changsha 410082, Hunan, China

3 Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

4 College of Computer Science, Zhejiang University, Hangzhou 310027, Zhejiang, China

extensive experiments on both synthetic and real data sets, we demonstrate the efficiency, effectiveness, and progressiveness of the proposed algorithms.

Keywords Data management · Group skyline · Query processing

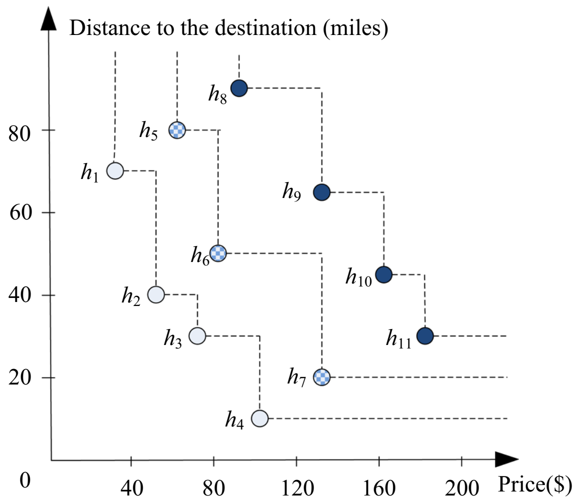
1 Introduction

The skyline query is an important tool in decision making applications [1]. It aims at capturing superior points (skylines) which are not dominated by another in a given data set. For two given multi-dimensional points p and p', p dominates p' denoted as $p \prec p'$ if for all the dimensions p is not worse than p' and there is at least on one dimension p is better than p'. However, there are many real-life applications requiring to get groups of prominent points. In these applications, the skyline query cannot be applied directly for it can only get individual points but not point groups. In order to achieve this goal, many group skyline queries [2–5] have been formulated. Most of them need to compute aggregate points of candidate groups of points and identify the optimal groups due to the dominance relationship of the aggregate points. These group skyline queries face the difficulty of specifying an appropriate aggregate function. Besides, they may miss some significant results that users are interested in.

To overcome the difficulty of the group skyline queries in [2–5] and get much more comprehensive results, the G-Skyline (GSky) query is proposed by Liu et al. [6] for the first time. The GSky query aims to pick out optimal groups based on a new operator of group dominance. Consider two given groups G_1 and G_2 of size k. The group G_1 dominates another group G_2 if for any point $p \in G_1$, p dominates or equals points $p' \in G_2$; at the same time, there is at least one point p dominating p'. A group including k points is called G-Skyline if it is not dominated by any other group of the same size. From this definition, the groups only consisting of by skyline points are G-Skylines. Besides, a G-Skyline G may contain non-skyline points that are only dominated by another point in the G-Skyline G but are not dominated by any point outside G.

Figure 1 illustrates an example of the GSky query. As shown, the hotel data set $H = \{h_1, h_2, \ldots, h_{11}\}$ contains eleven hotels (data points). For simplicity, we take two types of dimensions in hotels into account, which are distance and price. Without loss of generality, a cheap hotel close to the destination is considered preferred. The hotels h_1, h_2, h_3, and h_4 are skylines as there is no hotel in the given hotel set dominating them. For a travel agency, it is common to pick out more than one hotel for cooperation. Assume that it needs to select 4 hotels. According to dominance relationships between different hotel groups of size 4, we get all the optimal hotel groups (G-Skylines) depicted in Fig. 1b. Other groups consisting of 4 hotels are not G-Skylines, and they are not shown in the figure. Take the hotel group $\{h_3, h_4, h_{10}, h_{11}\}$ as an example, it cannot be a G-Skyline because it is dominated by the group $\{h_3, h_4, h_7, h_{11}\}$ for $h_7 \prec h_{10}$. Compared to the hotel h_{10}, the hotel h_7 is both cheaper and closer to the destination. As well as the hotel group $\{h_1, h_2, h_3, h_4\}$ consisting of hotels which are skylines, the left G-Skylines, such as $\{h_1, h_2, h_3, h_5\}$, $\{h_1, h_2, h_4, h_5\}$, $\{h_1, h_2, h_3, h_6\}$, just to name a few, contain non-skyline hotels that are only dominated by other hotels in the same G-Skyline. For instance, the G-Skyline $G = \{h_1, h_2, h_3, h_5\}$ contains the non-skyline h_5 dominated by two hotels h_1 and h_2; however, the hotels outside G cannot dominate h_5. Figure 1b illustrates the non-skyline hotels included in each G-Skyline.

In comparison with the group skyline queries presented in [2–5], the GSky query exempts users from selecting the aggregate function and can get much more comprehensive query



(a)

G-Skylines	non-Skylines
$\{h_1, h_2, h_3, h_4\}$	
$\{h_1, h_2, h_3, h_5\}$	h_5
$\{h_1, h_2, h_4, h_5\}$	h_5
$\{h_1, h_2, h_3, h_6\}$	h_6
$\{h_2, h_3, h_4, h_6\}$	h_6
$\{h_1, h_2, h_4, h_7\}$	h_7
$\{h_1, h_3, h_4, h_7\}$	h_7
$\{h_2, h_3, h_4, h_7\}$	h_7
$\{h_3, h_4, h_7, h_{11}\}$	h_7, h_{11}

(b)

Fig. 1 Example of the GSky query

results. After Liu et al. presented the GSky query, there are many follow-up studies [6–8]. However, from the experimental results in [6–8], the GSky query faces the problem of combinatorial explosion and the size of the G-Skylines may become excessively large. For a given data set D containing 100 skyline points with $k = 3$, the number of G-Skylines is $\binom{100}{3} = 161,700$ at least. Obviously, it is a huge burden for users to process all the G-Skylines manually and then make a rational decision.

To help the users make a quick and rational decision, we could adjust the algorithms proposed in [6–8] to process the G-Skyline query and pick out the results due to user preferences. However, this method may generate many redundant G-Skylines and cost much to identify the final query results. Moreover, it cannot get any result until the end of the query procedure, which leaves plenty of room for improving the progressiveness.

In recent years, to improve the databases' usability, why-not questions have received growing attention [9–16]. A why-not query is to help users know why some expected results are missed and then refine corresponding queries to ensure reporting these results [9]. Inspired by but different from the why-not queries, we investigate a new GSky query, called Flexible G-Skyline (FGSky) query to compute the expected optimal groups satisfying user preferences. The GSky query in [6] is a special case of the FGSky query without taking into account user preferences.

Different from the GSky query, the FGSky query can get refined results due to user specified preferences. This is significant to lighten the burden of the user when making a final decision. Besides, it is much more appropriate to the applications as follows.

Application 1 In some real-life applications, the user may expect certain points contained in the GSky query results. Consider the hotel example in Fig. 1. A travel agency may expect long-term cooperative hotels appearing in the selected hotel group. In Fig. 1, let the hotel h_5 be the expected hotel. This means that it requires to compute the optimal hotel groups of size 4 which contain the hotel h_5. Therefore, the hotel groups $\{h_1, h_2, h_3, h_5\}$ and $\{h_1, h_2, h_4, h_5\}$ including the hotel h_5 are expected results of the travel agency.

Application 2 The data set is usually updated in practical applications. After an update operator, we need to perform the GSky query over the new data set. However, the new GSky query always retrieves a large number of query results identical with the GSky query over the old data set before updating. This brings a lot of redundant computation cost. To improve usability of the query results, the FGSky query could be applied to compute new G-Skylines that contain the updated points. As an example, in Fig. 1, assume the travel agency selects the hotel set $\{h_1, h_2, h_3, h_6\}$ at first. In the process of cooperation, the hotel h_6 is closed down suddenly. To reuse the old GSky query results, the G-Skylines containing the hotel h_6 and the hotels h_8, h_9 dominated by h_6 are removed. In Fig. 1b, the old G-Skylines except the G-Skylines $\{h_1, h_2, h_3, h_6\}$ and $\{h_2, h_3, h_4, h_6\}$ could be returned directly. Besides, we need to generate new G-Skylines that include the hotels h_8 or h_9.

For a given user-specified preference set P and group size k, to get the FGSky query results, the intuitive method is to compute the G-Skylines G of size $k - |P|$ and then generate candidate groups by merging them with the set P. The groups returned by this method are the FGSky query results as they meet the user requirement. However, this method may miss some results.

In the above example, for a given user expected hotel set $\{h_4\}$ and the group size $k = 3$, the G-Skylines of size $k - 1 = 2$ need to be computed and merged with $\{h_4\}$ to get the final FGSky query results. In [6], the authors introduced that the G-Skylines with size k only contain points belonging to the first k skyline layers. Accordingly, the G-Skylines with size 2 only include the points of the 1st and 2nd skyline layer, and the hotel set $\{h_4, h_7, h_{11}\}$ which is a FGSky query result is overlooked. This is because the hotel $h_{11} \in \{h_4, h_7, h_{11}\}$ is in the 3rd skyline layer.

To the best of our knowledge, we investigate the FGSky query for the first time in the literature. The FGSky query is more practical to the GSky query proposed in [6]. In the FGSky query, user preferences are taken into account to compute the optimal groups that the user expects. The GSky query in [6] is a special case of the FGSky query when the user preference set is empty. In addition, the FGSky query is applicable to many applications where data sets are frequently updated.

The FGSky query has a different goal with the GSky query. The state-of-the-art algorithm, G-MDS [8], for the GSky query cannot be utilized to process the FGSky query directly. In this paper, we first extend G-MDS and propose the extended G-MDS algorithm, called EGM, for processing the FGSky query. The EGM algorithm takes the minimum dominance graph (MDG) [8] to organize the given data set. Next, we propose a layered minimum dominance graph (LMDG) to refine MDG by introducing the layered strategy to MDG to remove redundant edges. Based on LMDG, we apply a grouping strategy to the FGSky query and develop the GCQ algorithm to further improve the effectiveness and progressiveness of the FGSky query. The GCQ algorithm generates abundant candidate groups at first, and then identifies the final FGSky query results. Last, different from the GCQ algorithm, we propose the layered FGSky query (LCQ) algorithm which only generates the candidate groups with high possibilities to be the final results. From experimental results, the LCQ algorithm can get best performance among the proposed algorithms.

Specially, our work has the following contributions.

- We propose the FGSky query to get the optimal results satisfying user preferences.
- We investigate the properties of the FGSky query and present effective pruning strategies to cut down the search space.
- We develop effective and progressive algorithms to process the FGSky query where the grouping strategy and the layered strategy are utilized.

- We use both synthetic and real data sets in the experiments to verify the performance of the proposed algorithms.

The rest of the paper is organized as follows. Section 2 reviews related work on the FGSky query. Section 3 introduces the FGSky query and its properties. Section 4 proposes three algorithms for the FGSky query. Section 5 evaluates the performance of the proposed algorithms. Section 6 concludes this paper.

2 Related work

In this section, we summarize the work related to group skyline queries and why-not queries.

2.1 Group skyline queries

The group skyline query is a useful tool to analyze optimal groups of points. There has been abundant work about the group skyline query. In [17], Wu et al. studied how to create competitive products that are not dominated by any exiting product in the market. In [18], Su et al. focused on the top k combinatorial skyline query that aims to compute the combinatorial skyline tuples with highest aggregate values on a certain attribute. In [4], Magnani et al. proposed a new aggregate skyline query by merging the skyline query and the group by operator. In [5], Zhang et al. focused the query that retrieves k tuple groups not being dominated by another group of the same size.

As pointed out in [6], most of the traditional group skyline queries may miss some results. This is because the dominance relationship between groups depends on the traditional dominance relationship between corresponding aggregate points. Here, the attributes of aggregate points are computed by the aggregate values of attributes of all points within the groups. To resolve this problem, Liu et al. [6] developed the pareto group-based skyline (GSky) query through introducing a new operator named group dominance. The GSky query is more practical with comparing to traditional group skyline queries, which need to calculate an aggregate point for each group. It is because the GSky query is exempt from specifying an aggregate function that leads to miss important results containing non-skyline points. Therefore, it can get much more comprehensive query results which not only contain the G-Skylines consisting of skyline points but also the G-Skylines including non-skyline points.

To efficiently process the GSky query, Liu et al. [6] proposed a directed skyline graph (DSG) to organize the first k skyline layers where k is the group size. Based on DSG, the point-wise algorithm and the unit-wise algorithm for the GSky query were presented. Although the two algorithms can process the GSky query effectively, they lack progressiveness and cannot get any result until the end of the algorithm. To further improve the progressiveness and efficiency of the algorithms in [6], we developed the layered unit-based (LU) algorithm [19] by introducing the layered optimum strategy. Different from the algorithms in [6], the LU algorithm can get better performance which benefits from only generating candidate groups that have large possibly to be the final results. Recently, Wang et al. [8] further refined DSG in [6] and introduced the minimum dominance graph (MDG). It organizes the points that are dominated by less than k points in two parts , the left part and the right part. The MDG is considered as the minimum G-Skyline support structure. In addition, they improved the G-Skyline query performance by employing a skyline combination-based optimization strategy. In [20], Zhu et al. designed a parallel approach to the GSky query.

Apart from the work about the GSky query, there are also some work on important variants of the GSky query. The GSky query always retrieves too numerous results to make users overwhelmed. To address this concern, top k GSky queries are investigated in [21,22]; these queries compute the k best groups with the maximum dominated points. Furthermore, in [23], Yu et al. formulated a representative G-Skyline that represents the contour of the G-skyline. Moreover, they presented a group-based clustering algorithm to compute these representative G-Skylines.

Different from the above research on the GSky query and the variants in [21–23], in this paper we mainly investigate a novel variant of the GSky query, named the FGSky query, for the first time. The FGSky query aims to obtain optimal groups that users expect based on their preferences.

2.2 Why-not queries

Why-not queries are utilized to analyze the reason of excluding certain expected answers from the final query results. Besides, it could guide users to reformulate the expected results for satisfying their practical demand. The why-not queries are usually classified into categories including manipulation identification, database modification, and query refinement [13,24]. In the following, we summarize the why-not queries using the query refinement method which are close related to our work.

There is abundant work about query refinement such as the why-not question on SPJA query [16], reverse skyline query [10], top k query [9,25], top k dominating query [25], reverse top k query [11,13], similar graph [14], spatial keyword query [15], spatial keyword top k query [12], metric probabilistic range query [26], and range-based skyline query in road network [24]. Recently, in [27,28], Chen et al. were concerned about the direction-aware why-not spatial keyword top k query problem. In [13], Liu et al. also researched the why questions on the reverse top k query that aims to eliminate the unexpected points from the final query results. The focuses of the why-not queries aforementioned are to compute appropriate parameters for corresponding queries to ensure returning all the user expected results. For example, in the why-not reverse skyline query, it helps users get their expected results by modifying the why-not point or the query point.

It is worth noticing that our work in this paper is related to the why-not queries, but they have significant differences. In our work, we aim to retrieve the G-Skylines satisfying user preferences directly. To achieve this goal, it is unnecessary to modify the parameters in the proposed FGSky query. Accordingly, the exiting approaches to the above why-not queries are not applicable to the FGSky query.

Table 1 illustrates the frequently used symbols in this paper.

3 The flexible group skyline query

In this section, we formulate the flexible group skyline (FGSky) query, where the user preferences are introduced into the GSky query. The GSky query can be considered as a special case of the FGSky query with an empty user preference set.

Definition 3.1 (*Skyline Query* [1]) For a given data set D in d-dimensional space, the skyline query aims to pick out points $p \in D$ that are not dominated by points $p' \in D - \{p\}$. Assume that for each dimension of the points within D, small value is preferred. The point p' dominates another point p, denoted as $p' \prec p$, if it holds that for all i, $p'[i] \leq p[i]$ and

Table 1 Summary of frequently used notations

Notation	Definition
D	Data set
N	Cardinality of data set
k	Group size
G	Group of points
SL_i	The ith skyline layer
P	Preference set containing all the user expected points
u_p	Unit group of p
$AncSet(p)$	Ancestor set of p
$PreG$	Preference group consists of the points within P and $AncSet(P)$
$DChiSet(p, i)$	Direct child set of p over the ith skyline layer

there is at least one i, $p'[i] < p[i]$. Here, $1 \leq i \leq d$. The points $p \in D$ not dominated by any other point p' are called skylines.

Consider the example in Fig. 1, the hotel h_1 is a skyline since it is not dominated by any other hotel in the given hotel data set. Similarly, the hotels h_2, h_3, and h_4 are also returned as skylines. However, the other hotels h_i for $5 \leq i \leq 11$ are not skylines since they are dominated by at least one hotel. For instance, the hotel h_8 is dominated by the hotel h_1 and h_5 because its price and distance are both larger than those of h_1 or h_5.

Definition 3.2 (*Skyline layer*) For a given data set D and a parameter k, the first skyline layer SL_1 contains skylines of D. The skyline layer SL_i includes points that are dominated by at least one point within SL_j but are not dominated by any point within $D - \cup_{1 \leq x \leq j-1} SL_x$ for $1 < i \leq k$ and $1 \leq j \leq i - 1$.

In Fig. 1, the hotel data set is organized in three skyline layers $SL_1 = \{h_1, h_2, h_3, h_4\}$, $SL_2 = \{h_5, h_6, h_7\}$, and $SL_3 = \{h_8, h_9, h_{10}, h_{11}\}$. Consider two attributes, the price and the distance to destination, of the hotels. Each hotel within SL_2 is dominated by at least one hotel within SL_1 and is not dominated by any hotel within SL_2 and SL_3.

Definition 3.3 (*Ancestor Set, AncSet*) For a given point $p \in D$, the ancestor set of p, denoted as $AncSet(p)$, contains all the points dominating p. Moreover, for a set $P' \subseteq D$, its ancestor set is

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

Definition 3.4 (*Group Dominance* [6]) For two given k-point groups $G, G' \subseteq D$, G g-dominates G', denoted as $G \prec_g G'$, if there are 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. The k-point group G that is g-dominated by no other group of the same size is called G-Skyline.

Based on Definition 3.4, the G-Skyline is defined as follows.

Definition 3.5 (*G-Skyline* [6]) The groups G that are not g-dominated by another group of equal size.

For two groups $G = \{h_1, h_2, h_3, h_4\}$ and $G' = \{h_1, h_5, h_6, h_7\}$, we have $G \prec_g G'$ because $h_2 \prec h_5$, $h_3 \prec h_6$, and $h_4 \prec h_7$. The hotel group G is a G-Skyline because there is not any group of size 4 dominating it.

Definition 3.6 (*User preference set*) A user preference set is defined as a point set P, which includes all the points the user expects to be contained in the optimal groups.

The preference set in Definition 3.6 is used to represent user preferences.

To get the optimal groups that a user expects, a naive approach consists of two steps: computing all the G-Skylines and picking out the ones meeting user preferences. Although the naive algorithm is efficient, it has a lot of room to be improved in both efficiency and progressiveness. This is because it needs to generate a large number of candidate groups, most of which are not the final results. Besides, it cannot get any result until the end of the query procedure.

Consider the example in Fig. 1 with $P = \{h_1\}$. It first generates all the 9 G-Skylines and then identifies the ones containing the hotel h_1. For the FGSky query in this paper, only 6 G-Skylines are generated with the EGM algorithm in Sect. 4.2. Besides, the user expected results could be computed and outputted progressively.

Definition 3.7 (*Flexible Group Skyline Query, FGSky*): Given a data set D, a parameter k, and a user preference set $P \subseteq D$ including all the points expected, the FGSky query returns point groups $G \subseteq D$ of size k such that G are not dominated by another group of size k and G contain all the points within P, formally,

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

Going back to the example in Fig. 1 with $k = 4$ and the user preference hotel set $P = \{h_4\}$, the FGSky query retrieves the hotel groups $\{h_1, h_2, h_3, h_4\}$, $\{h_1, h_2, h_4, h_5\}$, $\{h_2, h_3, h_4, h_6\}$, $\{h_1, h_2, h_4, h_7\}$, $\{h_1, h_3, h_4, h_7\}$, $\{h_2, h_3, h_4, h_7\}$, and $\{h_3, h_4, h_7, h_{11}\}$, each of which contains the hotel h_4 and is dominated by no other hotel group of the same size.

4 Approaches to the FGSky query

The naive algorithm for the FGSky query is to compute the G-Skylines directly and then find out the ones satisfying the user preferences. Although this algorithm is efficient, it needs to generate a large number of unqualified candidate groups which are not the final results. Additionally, it cannot get any result until the end of the query procedure.

In this section, we first adjust the state-of-the-art algorithm, G-MDS in [8], and develop the extended G-MDS algorithm for the FGSky query. After that, we present the group FGSky query algorithm and the layered FGSky query algorithm which boost the query performance by applying the grouping and layered strategies.

4.1 The extended G-MDS (EGM) algorithm

In this subsection, we first introduce the minimum dominance graph (MDG) proposed in [8]. Then, we modify the state-of-the-art algorithm, G-MDS [8], and develop the extended G-MDS (EGM) algorithm for the FGSky query.

The G-MDS algorithm takes MDG that organizes all the points only dominated by less than k points as an input. Based on MDG, it sorts all the points in MDG in descending order

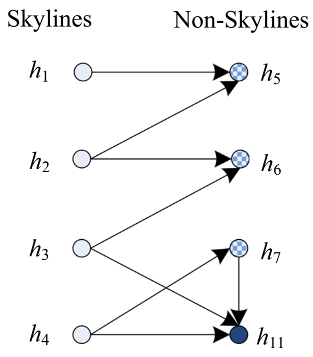


Fig. 2 MDG of the hotel data set in Fig. 1

of the sizes of their ancestors. Then, for each point $p \in$MDG, new candidate groups are generated recursively in a depth-first manner by adding p and its ancestors.

4.1.1 The minimum dominance graph

Recently, Wang et al. [8] introduced a minimum dominance graph (MDG) which is considered as the minimum g-skyline support structure without redundancy. Similar to DSG in [6], the structure of each node in MDG includes four components, including layer index, point index, ancestors, and descendants. The layer index is the skyline layer, and the point index is a unique identifier assigned to each point. Besides, the ancestors and descendants are points dominating a given point and points dominated by the same point, respectively. The edges represent the dominance relationship between points.

Theorem 4.1 (Verification theorem [8]) *Let D be a set of d-dimensional data points and G be a k-point group of D. G is a G-skyline if and only if for each point $p \in G$ there exists no point outside G that dominates p.*

In a G-Skyline G of size k, each point $p \in G$ and all the points dominating p are contained in G due to Theorem 4.1. Accordingly, each point $p \in D$ dominated by more than $k - 1$ points cannot be contained in a G-Skyline. For example, with $k = 4$ the hotel h_{10} does not exist in MDG shown in Fig. 2. This is because for any G-Skyline including h_{10}, the hotels h_2, h_3, h_4, h_7 dominating h_{10} are all contained in G. Therefore, the size of G is at least 5 which contradicts the assumption that $k = 4$. Inspired by this observation, MDG only takes into account the points dominated by at most $k - 1$ points. As depicted in Fig. 2 with $k = 4$, MDG only includes the hotels dominated by less than four hotels, and the hotels h_8 , h_9, and h_{10} are excluded from MDG. In addition, the hotels are divided into two parts, the left part contains all the hotels h_1 to h_4 that are skylines and non-skylines h_5, h_6, h_7, and h_{11} are included in the right part.

4.1.2 The extended G-MDS (EGM) algorithm

Due to Definition 3.7, the FGSky query is much more complicated than the GSky query, and the state-of-the-art algorithm, the G-MDS algorithm in [8], cannot process the FGSky query directly. We could generate all the G-Skylines by G-MDS and pick out the ones that the user expects. However, this approach generates numerous G-Skylines that are not the final results. As a result, it costs much time to identify the final result. To get better query performance, we extend the G-MDS algorithm and develop the EGM algorithm based on new properties of the FGSky query.

Lemma 4.2 *For a given preference set P and a FGSky query result G, it holds that* $\text{AncSet}(P) \subseteq G$.

Proof Due to Definition 3.7, if a point $p \in P$ is contained in the FGSky query result G which is a G-Skyline, then $AncSet(p) \subseteq G$. Therefore, $AncSet(P) \subseteq G$ and this lemma holds. □

Definition 4.1 (*Preference Group, PreG*): Given a preference set P, the preference group is denoted as

$$PreG(P) = P \cup AncSet(P).$$

In Fig. 1, for a hotel preference set $\{h_5\}$, we have $PreG(\{h_5\}) = \{h_5\} \cup AncSet(\{h_5\}) = \{h_1, h_2, h_5\}$.

Algorithm 1 Extend G-MDS (EGM) Algorithm

```
Require: A MDG $MD$, group size $k$, and a preference set $P$
Ensure: All the FGSky query results
1: Compute the preference group $PreG \leftarrow P \cup AncSet(P)$
2: Sort points $p \in MD - PreG$ with $|AncSet(p) - PreG| \leq k - |PreG| - 1$ in non-increasing order of
   $|AncSet(p)|$ and store the points $p$ within a candidate point set $CandP$
3: for Each point $p \in CandP$ do
4:     Compute a candidate group $G \leftarrow \{p\} \cup AncSet(p)$
5:     if $|G \cup PreG| = k$ then
6:         Report the group $G$
7:     else
8:         if $|G \cup PreG| < k$ then
9:             FGSkyGenerator($CandP$, $G$, $PreG$, $k$)
10:        end if
11:    end if
12: end for
```

The EGM algorithm depicted in Algorithm 1 first computes the preference group *PreG* by merging P and its ancestor set $AncSet(P)$. Next, the points in $MD - PreG$ are ranked in non-increasing order of the sizes of their ancestor sets. The candidate point set *CandP* is utilized to store these ordered points p whose sizes of ancestors outside *PreG* are no more than $k - |PreG| - 1$. This can ensure that the sizes of the candidate groups G are no more than k, where the candidate groups are generated in lines 3 to 12 by merging $\{p\}$, the ancestor set $AncSet(p)$, and the preference group *PreG*. Last, the candidate group G with size k is reported as a final result. If the size of G is less than k, the FGSkyGenerator algorithm as shown in Algorithm 2 is invoked to generate new candidate groups.

The FGSkyGenerator algorithm depicted in Algorithm 2 takes the candidate point set *CandP*, the candidate group G, the preference group *PreG*, and the group size k as inputs. It first identifies the point p_{last} which is the last point within the group G. Then, points $p' \in CandP$ ranked after the point p_{last} are considered, and new groups G are generated by merging G, p' and their ancestor sets. Finally, the groups $G \cup PreG$ of size k are reported as the final results. Besides, if the sizes of $G \cup PreG$ are less than k, FGSkyGenerator is invoked recursively.

Example Continuing the example in Fig. 1 with $k = 4$ and $P = \{h_1\}$, the EGM algorithm first computes the preference hotel group $PreG = P \cup AncSet(P) = \{h_1\}$. Then, we get

Algorithm 2 FGSkyGenerator Algorithm

```
Require: A candidate point set $CandP$, a candidate group $G$, a preference group $PreG$, and group size $k$
Ensure: FGSky query results
1: $p_{last}$ is the last point within $G$
2: for Each point $p' \in CandP$ ranked after $p_{last}$ do
3:     if $p' \notin G$ then
4:         Compute a new group $G \leftarrow G \cup \{p'\} \cup AncSet(p')$
5:         if $|G \cup PreG| = k$ then
6:             Report $G$ as a final result
7:         else
8:             if $|G \cup PreG| < k$ then
9:                 FGSkyGenerator($CandP$, $G$, $PreG$, $k$)
10:            end if
11:        end if
12:    end if
13: end for
```

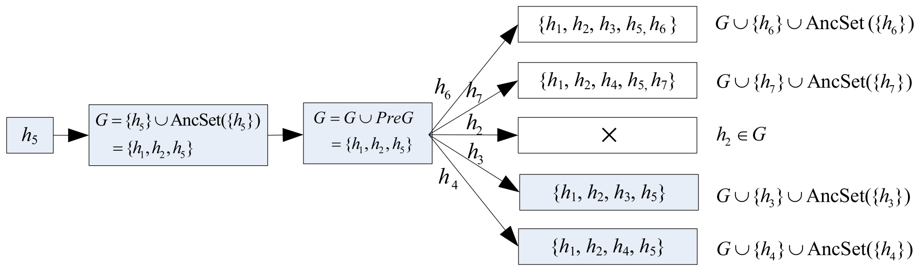

h_5
$G = \{h_5\} \cup \text{AncSet}(\{h_5\}) = \{h_1, h_2, h_5\}$
$G = G \cup PreG = \{h_1, h_2, h_5\}$
h_6
h_7
h_2
h_3
h_4
$\{h_1, h_2, h_3, h_5, h_6\}$
$\{h_1, h_2, h_4, h_5, h_7\}$
$\times$
$\{h_1, h_2, h_3, h_5\}$
$\{h_1, h_2, h_4, h_5\}$
$G \cup \{h_6\} \cup \text{AncSet}(\{h_6\})$
$G \cup \{h_7\} \cup \text{AncSet}(\{h_7\})$
$h_2 \in G$
$G \cup \{h_3\} \cup \text{AncSet}(\{h_3\})$
$G \cup \{h_4\} \cup \text{AncSet}(\{h_4\})$

Fig. 3 Example of the EGM algorithm

hotels $h \in MD - PreG$ satisfying $|AncSet(h) - PreG| \leq k - |PreG| - 1 = 2$. This means that outside the preference set $PreG$ there are at most 2 hotels dominating h. For instance, the hotel h_{11} is pruned for it is dominated by three hotels that are h_3, h_4 and h_7. The candidate groups containing h_{11} and the preference group $PreG = \{h_1\}$ must include at least 5 hotels that are $h_1, h_3, h_4, h_7, h_{11}$. Accordingly, h_{11} is not included in any G-Skyline with size 4 and can be pruned safely. After ranking these hotels h by their sizes of ancestors, we gain a candidate hotel set $CanP = \{h_5, h_6, h_7, h_2, h_3, h_4\}$. The hotels within $CanP$ are visited in sequence. Figure 3 shows the steps of the EGM algorithm when visiting the first hotel h_5. For the hotel h_5, we have a candidate group $G = \{h_5\} \cup AncSet(h_5) = \{h_1, h_2, h_5\}$. Through merging the preference group $\{h_1\}$ and the candidate group $\{h_1, h_2, h_5\}$, we can get a group $G = \{h_1, h_2, h_5\}$ with size 3. Since $|G| = 3 < k = 4$, new candidate groups are generated by adding one of the unvisited hotels $h \in CanP - \{h_5\} = \{h_6, h_7, h_2, h_3, h_4\}$ and its ancestors to G at a time. By merging $\{h_1, h_2, h_5\}$ with $\{h_6\} \cup AncSet(\{h_6\}) = \{h_2, h_3, h_6\}$, we get a new hotel group $\{h_1, h_2, h_3, h_5, h_6\}$ which is pruned as an unqualified group directly. This is since its size is larger than $k = 4$. Similarly, the group $\{h_1, h_2, h_4, h_5, h_7\}$ that is generated by adding h_7 and its ancestors to $G = \{h_1, h_2, h_5\}$ is pruned. Consider the hotel h_2. Because $h_2 \in G$, $\{h_2\}$ could be pruned safely. This is because merging $\{h_2\}$ and its ancestors to G cannot bring new candidate groups. Next, based on the hotels h_3 and h_4, we get two hotel groups $\{h_1, h_2, h_3, h_5\}$ and $\{h_1, h_2, h_4, h_5\}$ which could be returned as the FGSky query results. Finally, based on the remaining hotels within $CanP$, the other generated and reported query results are $\{h_1, h_2, h_3, h_6\}$, $\{h_1, h_2, h_4, h_7\}$, $\{h_1, h_3, h_4, h_7\}$, and $\{h_1, h_2, h_3, h_4\}$.

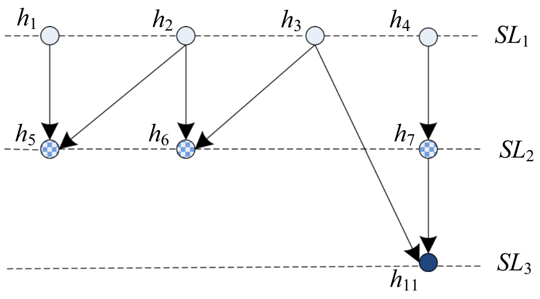


Fig. 4 LMDG of the hotel set in Fig. 1

Complexity The EGM algorithm first costs $O(N' \times \log N')$ where N' is the cardinality of the points within *MD* but not included in *PreG* to sort the points $p \in MD - PreG$. We use the set *CandP* to store all the points after sorting. Then, the points $p \in CandP$ are processed sequentially. For the ith point $p_i \in CandP$, it needs to select at most $k_i = k - |PreG| - |AncSet(p_i)| - 1$ points from $CandP - \cup_{j=1}^{i}\{p_j\}$. This costs $O\left(\binom{N'-i}{k_i}\right)$ for p_i. Therefore, taking into account all the points within *CandP*, it needs $O\left(\sum_{i=1}^{N'}\binom{N'-i}{k_i}\right)$. Therefore, the time complexity of the EGM algorithm is $O\left(N' \times \log N' + \sum_{i=1}^{N'}\binom{N'-i}{k_i}\right)$.

4.2 The group FGSky query algorithm

Although MDG contains no redundant points, it includes redundant edges. As shown in Fig. 2, as well as the edge $h_4 \rightarrow h_{11}$, it also contains two edges $h_4 \rightarrow h_7$ and $h_7 \rightarrow h_{11}$. To further refine MDG, we introduce a layered strategy where the points are organized in different skyline layers to MDG in [8] and produce a layered minimum dominance graph (LMDG). Then, to further improve the FGSky query performance, the Group FGSky Query algorithm is developed.

4.2.1 The layered minimum dominance graph

In this paper, we adjust MDG [8] and introduce a layered MDG, namely LMDG, to organize the data set. In LMDG, the points dominated by at most $k-1$ points are divided into different skyline layers. In addition, we refine the structure of each node as [*layer index*, *point index*, *ancestors*, *direct children*]. Here, each node stores the ancestors and direct children instead of all the descendants in the DSG. The direct child set is defined as follows.

Definition 4.2 (*Direct child set*) For two given points p and p', p' is a direct child of p if there is an edge between the nodes of p and p' in the DSG . The direct child set of a point p, $DChiSet(p, i)$, is composed of the direct children of p in the ith skyline layer. Besides, for a given point group G, its direct child set is $DChiSet(G, i) = \bigcup_{p \in G} DChiSet(p, i)$.

Figure 4 shows LMDG over the hotel data set H in Fig. 1. It contains 3 skyline layers $SL_1 = \{h_1, h_2, h_3, h_4\}$, $SL_2 = \{h_5, h_6, h_7\}$, and $SL_3 = \{h_{11}\}$. The edges in LMDG represent the dominance relationship between different hotels. For instance, the edge $h_3 \rightarrow h_6$ means that the hotel h_6 is dominated by the hotel h_3. In the node representing h_{11}, as well as the layer index 3 and point index 11, it stores the ancestor set $\{h_3, h_4, h_7\}$ and its direct child set is empty.

4.2.2 The group FGSky query algorithm

In this subsection, we propose the group FGSky query (GCQ) algorithm based on the following lemmas and theorems. In the GCQ algorithm, we employ LMDG to organize the data sets and introduce the grouping strategy for boosting the query performance.

Lemma 4.3 *For a given group G, it is a G-Skyline if* $\mathrm{AncSet}(p) \subseteq G$ *for each* $p \in G$ [6].

Definition 4.3 (*Unit Group* [6]) Given a point $p \in D$, the unit group of p is denoted as

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

Theorem 4.4 (Verification of the G-Skyline [6]) *For a given group $G = \{p_1, p_2, \ldots, p_k\}$, it is a G-Skyline, if and only if $| \cup_{i=1}^{k} u_i | = k$ where u_i is the unit group of the point p_i.*

Lemma 4.5 *For a preference set P and groups $G \subseteq \mathrm{SL}_1$, the group $G' = P \cup \mathrm{AncSet}(P) \cup G$ is a FGSky query result if and only if $|G'| = k$.*

Proof Due to Definition 3.7, if G' is a FGSky query result, then G' is a G-Skyline that is not dominated by any other G-Skyline of size k, and G' contains all the points within the preference set P. For each point $p \in G'$, we have $p \in P$, $p \in AncSet(P)$ or $p \in G$. If $p \in P$, it holds that $AncSet(p) \subseteq AncSet(P) \subseteq G'$. When $p \in AncSet(P)$, then p dominates some points within P, and each ancestor of p is also included in the set $AncSet(P)$ due to Lemma 4.3. On assumption that $p \in G$, there is no point dominating p since $G \subseteq SL_1$ where SL_1 contains all the points within the first skyline layer. As analyzed above, each point within G' and its ancestors are both contained in G', and G' is a G-Skyline based on Theorem 4.4. Since G' is a G-Skyline that contains the preference set P, G' is a FGSky query result. □

As listed in Algorithm 3, the GCQ algorithm first processes the points $p \in SL_1$ and generates groups $G \subseteq SL_1 - PreG$ of size i for $1 \le i \le k'$. Here, $PreG$ consists of the preference set P and its ancestor set. Besides, k' is equal to $k - |PreG|$ because the final FGSky query results contain at most $k - |PreG|$ points within SL_1. The groups G are divided into different groups according to their sizes (line 3). This means that the groups G containing i points within SL_1 are inserted to the set V_i. To improve the progressiveness of the FGSky query, the groups within $V_{k'}$ are considered at first (lines 4 to 7). We could get some FGSky query results by merging G and the set $PreG$ for each $G \subseteq V_{k'}$ due to Lemma 4.5, and k' is refreshed as $k' - 1$. Lines 8 to 26 are a for loop to compute the left FGSky query results based on candidate groups within V_i for $1 \le i \le k'$. To boost the progressiveness of the GSky query, the set V_i containing more points within SL_1 is given priority to be processed. If V_i is not empty, new candidate groups are built by merging each group $G' \in V_i$ with some direct children of $p \in G'$ (lines 10 to 24). In line 11, *maxlay* is initialized as the max skyline layer that the points $p \in G'$ belong to. After that, the candidate set *CanP* is computed as the direct child set which contains all direct children of points within $G' \cup PreG$ in the $(maxlay{+}1)^{\text{th}}$ skyline layer. For each subset DC' of *CanP* with size no more than $k' - |G'|$, if each ancestor of the points $p \in DC'$ is contained in G' or *PreG* (line 14), then a new group G'' is built by merging G' and DC' (line 15). The group G'' of size k' is merged with *PreG* to get a new result of the FGSky query (line 19), and the group of size less than k' is inserted to V_i as a

Algorithm 3 Group FGSky Query (GCQ) Algorithm

```
Require: A LMDG $LM$ over $D$, group size $k$, and a preference set $P$
Ensure: FGSky query results
 1: Compute the preference group $PreG \leftarrow P \cup AncSet(P)$
 2: $k' \leftarrow k - |PreG|$
 3: Generate candidate groups $G \subseteq SL_1 - PreG$ of size $i$ and insert them into a set $V_i$ for $1 \leq i \leq k'$
 4: if $V_{k'}$ is not empty then
 5:     Report groups $G \cup PreG$ for $G \in V_{k'}$ due to Lemma 4.5
 6:     $k' \leftarrow k' - 1$
 7: end if
 8: for $i \leftarrow k'$ to 1 do
 9:     if $V_i$ is not empty then
10:         for Each group $G'$ in $V_i$ do
11:             $maxlay \leftarrow \max_{p \in G'} p.layer$
12:             $CanP \leftarrow DChiSet(G' \cup PreG, maxlay+1)$
13:             for Each $DC' \subseteq CanP$ with $|DC'| \leq k' - |G'|$ do
14:                 if $AncSet(DC') \subseteq G' \cup PreG$ then
15:                     Generate a new candidate group $G'' \leftarrow G' \cup DC'$ due to Lemma 4.3
16:                     if $G'' < k'$ then
17:                         Insert $G''$ to the set $V_i$
18:                     else
19:                         Report $G'' \cup PreG$ if $|G''| = k'$
20:                     end if
21:                 end if
22:             end for
23:             $V_i \leftarrow V_i - \{G'\}$
24:         end for
25:     end if
26: end for
```

new candidate group (line 17). In line 23, the set V_i is updated by removing the group G' which has been processed.

Example Back to the example of Fig. 1 with $k = 4$ and $P = \{h_1\}$, the GCQ algorithm first computes the preference hotel group $PreG = \{h_1\}$ and $k' = 4 - |PreG| = 3$. Next, the hotel groups $G \subseteq SL_1 - PreG = \{h_2, h_3, h_4\}$ of size not more than $k' = 3$ are generated and divided into three hotel sets $V_1 = \{\{h_2\}, \{h_3\}, \{h_4\}\}$, $V_2 = \{\{h_2, h_3\}, \{h_2, h_4\}, \{h_3, h_4\}\}$, and $V_3 = \{\{h_2, h_3, h_4\}\}$. Here, the hotel group set V_i for $1 \leq i \leq 3$ consists of hotel groups that contain only i hotels in the first skyline layer SL_1. The set V_i containing hotel groups with large sizes is given priority to be processed. This contributes to improve the progressiveness of the FGSky query.

The hotel groups in V_3 are considered firstly. The hotel group $\{h_2, h_3, h_4\} \in V_3$ includes 3 hotels $h_2, h_3, h_4 \in SL_1$. By merging $\{h_2, h_3, h_4\}$ and the preference set $PreG = \{h_1\}$, we get a FGSky query result which is $\{h_1, h_2, h_3, h_4\}$. Then, the hotel groups within V_2 are taken into account. Figure 5 shows the steps of the GCQ algorithm when visiting the hotel groups within V_2. For the hotel group $G' = \{h_2, h_3\}$, $maxlay = \max_{h \in G'} h.layer = 1$ and the candidate hotel set $CanP = DChiSet(G' \cup PreG, 2) = DChiSet(\{h_1, h_2, h_3\}, 2) = \{h_5, h_6\}$. The hotel set $\{h_5, h_6\}$ contains all the direct children of hotels within $\{h_1, h_2, h_3\}$ in the 2nd skyline layer. There are two hotel groups, $\{h_5\}$ and $\{h_6\}$, which are subsets of $\{h_5, h_6\}$ with sizes not more than $k' - |G'| = 3 - 2 = 1$. For $AncSet(\{h_5\}) = \{h_1, h_2\} \subseteq G' \cup PreG = \{h_1, h_2, h_3\}$, we could add h_5 to $\{h_1, h_2, h_3\}$ directly to generate a new candidate hotel group $\{h_1, h_2, h_3, h_5\}$. Similarly, by merging $\{h_6\}$ and $\{h_1, h_2, h_3\}$, we get another candidate group $\{h_1, h_2, h_3, h_6\}$. The hotel groups $\{h_1, h_2, h_3, h_5\}$ and $\{h_1, h_2, h_3, h_6\}$ where each hotel is not

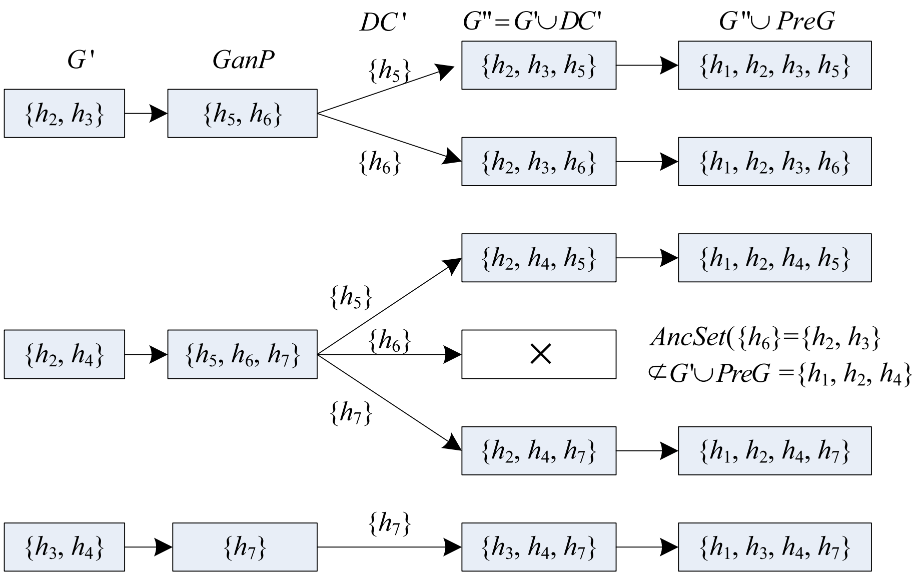


Fig. 5 Example of the GCQ algorithm

dominated by other hotels outside the groups could be reported as final FGSky query results directly. Taking into account the hotel group $\{h_2, h_4\} \in V_2$, we have the candidate set $CanP = DChiSet(\{h_1, h_2, h_4\}, 2) = \{h_5, h_6, h_7\}$. The hotel groups $\{h_5\}$, $\{h_6\}$, and $\{h_7\}$ are subsets of $\{h_5, h_6, h_7\}$, and their sizes are all less than $k' - |G'| = 1$. For the hotels h_5 and h_7 whose ancestors are all contained in $\{h_1, h_2, h_4\}$, they could be added to $\{h_1, h_2, h_4\}$ to generate two new hotel groups $\{h_1, h_2, h_4, h_5\}$ and $\{h_1, h_2, h_4, h_7\}$ which are final FGSky query results. However, since h_3, one of the ancestors of h_6, is not included in $\{h_1, h_2, h_4\}$, h_6 could be pruned safely. Based on the last hotel group $\{h_3, h_4\}$ with V_2, we could gain a new FGSky query result $\{h_1, h_3, h_4, h_7\}$. Finally, the hotel groups in V_1 are considered. By merging the hotel group $\{h_4\} \in V_1$ and its direct child set $\{h_7\}$, we have a new candidate group $\{h_4, h_7\}$ and $maxlay = \max_{h \in \{h_4, h_7\}} h.layer = 2$. Since $DChiSet(\{h_4, h_7\}, maxlay{+}1) = \{h_{11}\}$ and h_3 one of the ancestors of h_{11} is not contained in $\{h_4, h_7\}$, we could not get any new FGSky query result based on $\{h_4\}$. In addition, there is no FGSky query result generated based on other hotel groups within V_1.

Complexity The GCQ algorithm generates candidate groups containing i points within SL_1 and it costs $O(\binom{\hbar_1}{i})$ where $\hbar_1 = |SL_1 - PreG|$ and $1 \leq i \leq k' = k - |PreG|$. Then, based on each group G' within V_i for $1 \leq i \leq k' - 1$, new candidate groups are generated by adding corresponding direct children. This costs $O\big(|V_i| \times |DC'|\big)$ where the groups DG' denote subsets of the direct child set of $G' \cup PreG$ over the $(maxlay + 1)^{\text{th}}$ skyline layer. Suppose that the maximum size of candidate groups generated on a group $G' \in V_i$ is ∂_i. Therefore, the time complexity of the GCQ algorithm is

$$O\Big(\sum_{i=1}^{k'} \Big(\binom{\hbar_1}{i} \times \partial_i\Big)\Big) < O\big(2^{\hbar_1} \times \partial\big)$$

for $\partial = \max \partial_i$.

4.3 The layered FGSky query algorithm

In the GCQ algorithm, it generates abundant candidate groups at first and then identifies the final FGSky query results. To further improve the query performance, we propose the layered FGSky query (LCQ) algorithm which generates the FGSky query results directly.

Lemma 4.6 *For a given G-Skyline G and a user preference set P satisfying $P \subseteq G$, it holds that $|G'| \leq k - |\mathrm{PreG}|$ where the group $G' \subseteq \mathrm{SL}_i \cap G$ for $1 \leq i \leq k$.*

Proof On the assumption that $G' \subseteq SL_i \cap G$, each point $p \in G'$ lies on the ith skyline layer. Besides, the following relation holds,

$$|G' \cup AncSet(G') \cup PreG| \leq |G'| + |AncSet(G') \cup PreG| \leq k.$$

The above equation can be rewritten as:

$$|G'| \leq k - |AncSet(G') \cup PreG| \leq k - |PreG|.$$

In case $AncSet(G') \subseteq PreG$, there are at most $k - |PreG|$ points $p \in G$ contained in SL_i and this lemma holds. □

Theorem 4.7 *Consider a given G-Skyline $G = \{p_1, p_2, \ldots, p_k\}$. The points $p_i \in G$ for $1 \leq i \leq k$ are all in the first k skyline layers* [6].

According to Lemma 4.6, each FGSky query result contains at most $k - |PreG|$ points within SL_i for $1 \leq i \leq k$. Here, $PreG$ consists of the user-specified set P and its ancestor set $AncSet(P)$.

From Theorem 4.4, a G-Skyline could be generated by combining different unit groups. Inspired by this, in the following LCQ algorithm, we generate the FGSky query results through computing unit group sets in a bottom-up manner. When taking into account the unit group u_p of points p within the ith skyline layer SL_i, it creates candidate FGSky query results by merging u_p with the unit groups u_q of points q belonging to the jth skyline layer SL_j. Here, it holds that $j \leq i$.

As shown in Algorithm 4, the LCQ algorithm first combines the set P and $AncSet(P)$ to get the preference group $PreG$. Lines 2 to 21 are a for loop that takes into account the unit groups within different skyline layers separately. Due to Lemma 4.6, the groups $G \cup PreG$ are generated and reported as FGSky query results for each $G \subseteq SL_1 - PreG$ of size $k - |PreG|$ (Lines 4 and 5). Thereafter, other FGSky query results are generated based on the unit groups within SL_i for $1 < i \leq k$ (lines 7 to 19). It computes the unit group set U_i including all the unit groups of the points $p \in SL_i - PreG$. Then, the unit group sets $U' \subseteq U_i$ are generated (line 8). According to Lemma 4.6, the size of U' is at most $k - |PreG|$. For each unit group set U', a candidate group G' is built by merging $PreG$ and all the unit groups within U'. Last, the group G' of size k is returned as a final FGSky query result due to Theorem 4.5 (Line 12). In addition, for the group G' with $|G'| < k$, new groups G'' are computed by invoking the LCQ algorithm recursively. The groups G'' whose sizes are equal to k are reported as the final FGSky query results.

Example Consider the example in Fig. 1 with $k = 4$ and $P = \{h_1\}$ again. The preference group $PreG$ is computed as $P \cup AncSet(P) = \{h_1\}$. Then, the hotels within $SL_1 - PreG = \{h_2, h_3, h_4\}$ are taken into account firstly. We generate hotel group $\{h_2, h_3, h_4\}$ of size $k - |PreG| = 3$ over the hotels in $SL_1 - PreG = \{h_2, h_3, h_4\}$. Based on Lemma 4.6, the hotel group $PreP \cup \{h_2, h_3, h_4\} = \{h_1, h_2, h_3, h_4\}$ only including $k = 4$ hotels within

Algorithm 4 Layered FGSky Query (LCQ) Algorithm

```
Require: A LMDG $LM$ over $D$, a group size $k$, and a user preference set $P$
Ensure: FGSky query results
1: Compute the preference group $PreG \leftarrow P \cup AncSet(P)$
2: for $i \leftarrow 1$ to $k$ do
3:     if $i = 1$ then
4:         Generate groups $G \subseteq SL_1 - PreG$ of size $k - |PreG|$
5:         Report groups $G \cup PreG$ as FGSky query results
6:     else
7:         Compute a unit group set $U_i \leftarrow \{u_p | p \in SL_i - PreG\}$
8:         Generate unit group sets $U' \subseteq U_i$ of sizes at most $k - |PreG|$ due to Lemma 4.6
9:         for each $U'$ do
10:            Generate a candidate group $G' \leftarrow PreG \cup \bigcup_{u \in U'} u$
11:            if $|G'| = k$ then
12:                Report the group $G'$ as a FGSky query result
13:            else
14:                if $|G'| < k$ then
15:                    Generate groups $G'' \leftarrow$ LCQ($LM, k, G'$)
16:                    Report the groups $G''$ as FGSky query results if $|G''| = k$
17:                end if
18:            end if
19:        end for
20:    end if
21: end for
```

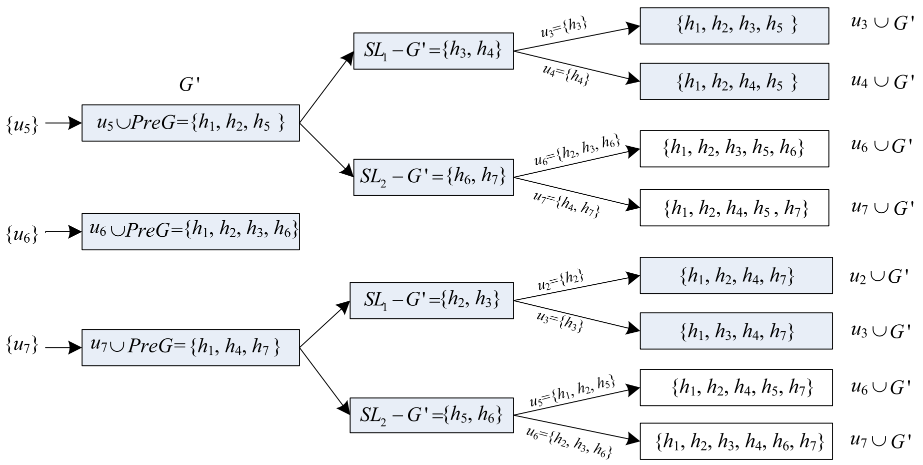


Fig. 6 Example of the LCQ algorithm

the first skyline layer SL_1 could be reported as a final result. Then, the hotels h_5, h_6, and h_7 within the 2nd skyline layer are considered. We get the unit group set $U_2 = \{u_5, u_6, u_7\} = \{\{h_1, h_2, h_5\}, \{h_2, h_3, h_6\}, \{h_4, h_7\}\}$. Based on Lemma 4.6, a FGSky query result contains at most $k - |PreG| = 4 - 1 = 3$ hotels within the 2nd skyline layer SL_2. Accordingly, a unit group set U', where each unit group contains at most three unit groups from U_2, is computed. Here, we have $U' = \{\{u_5\}, \{u_6\}, \{u_7\}, \{u_5, u_6\}, \{u_5, u_7\}, \{u_6, u_7\}, \{u_5, u_6, u_7\}\}$.

Figure 6 shows the steps of the LCQ algorithm when visiting the first three unit groups $\{u_5\}, \{u_6\}$, and $\{u_7\}$ of U'. By merging each unit group $u \in U'$ with the preference hotel set $PreG = \{h_1\}$, we have new candidate groups, $\{h_1, h_2, h_5\}$, $\{h_1, h_2, h_3, h_6\}$, $\{h_1, h_4, h_7\}$, $\{h_1, h_2, h_3, h_5, h_6\}$, $\{h_1, h_2, h_3, h_4, h_5, h_7\}$, $\{h_2, h_3, h_4, h_6, h_7\}$, and $\{h_1, h_2,$

$h_3, h_4, h_5, h_6, h_7\}$. The hotel group $\{h_1, h_2, h_3, h_6\}$ of size 4 can be reported as a final FGSky query result. However, the hotel groups $\{h_1, h_2, h_3, h_5, h_6\}$, $\{h_1, h_2, h_3, h_4, h_5, h_7\}$, $\{h_2, h_3, h_4, h_6, h_7\}$, $\{h_1, h_2, h_3, h_4, h_5, h_6, h_7\}$ are pruned directly because their sizes are larger than 4. Consider the hotel groups $\{h_1, h_2, h_5\}$ and $\{h_1, h_4, h_7\}$ of size less than 4. New hotel groups are computed by invoking the LCQ algorithm recursively. Through invoking LCQ(*LM*, 4, $\{h_1, h_2, h_5\}$), two new hotel groups $\{h_1, h_2, h_3, h_5\}$ and $\{h_1, h_2, h_4, h_5\}$ are generated by merging $\{h_1, h_2, h_5\}$ and hotel groups, $\{h_3\}$ and $\{h_4\}$, of size $k - |\{h_1, h_2, h_5\}| = 1$ over the hotel set $SL_1 - \{h_1, h_2, h_5\} = \{h_3, h_4\}$. The sizes of $\{h_1, h_2, h_3, h_5\}$ and $\{h_1, h_2, h_4, h_5\}$ are both $k = 4$, and they could be reported as final results. For the hotels $h_6, h_7 \in SL_2 - \{h_1, h_2, h_5\}$, we get a unit group set $U_2 = \{u_6, u_7\} = \{\{h_2, h_3, h_6\}, \{h_4, h_7\}\}$. Here, the unit of h_5 is ignored since h_5 has been contained in the hotel set $\{h_1, h_2, h_5\}$. Thereafter, the unit group set $U' = \{\{u_6\}, \{u_7\}\}$ of size at most $k - |PreG| = 1$ is computed. Through merging $\{h_1, h_2, h_5\}$ and each unit group set in U', respectively, we have two new hotel groups $\{h_1, h_2, h_3, h_5, h_6\}$ and $\{h_1, h_2, h_4, h_5, h_7\}$ with size $5>k$, and they could be pruned directly. Similarly, for the hotels within SL_3, we gain $U_3 = \{u_{11}\}$ and $U' = \{\{u_{11}\}\}$ where $u_{11} = \{h_3, h_4, h_7, h_{11}\}$. The hotel group $\{h_1, h_2, h_5\} \cup u_{11} = \{h_1, h_2, h_3, h_4, h_5, h_7, h_{11}\}$ with size larger than 4 could be pruned directly. Finally, based on the hotel group $\{h_1, h_4, h_7\}$, the hotel groups $\{h_1, h_2, h_4, h_7\}$ and $\{h_1, h_3, h_4, h_7\}$ are generated and returned as final FGSky query results.

Complexity Let h_i denote the cardinality of points within the i^{th} skyline layer for $1 \le i \le k$. The LCQ algorithm first generates groups G which are subsets of the point set $SL_1 - PreG$. The sizes of these groups G are all equal to $k' = k - |PreG|$. This costs $O\left(\binom{h_1}{k'}\right)$. The left of the LCQ algorithm is a for loop. In the for loop, it requires $O(|U_i| + |U|)$ to compute the unit group set U and the unit group sets U' where $|U_i| = h_i$ and $|U'| = \binom{h_i}{k'}$. For each unit group set U', assume that there are at most ∂_i candidate groups generated. Accordingly, based on the unit group set U, it costs $O\left(\binom{h_i}{k'} \times \partial_i\right)$ to generate new candidate groups. Therefore, the for loop costs

$$O\left(\sum_{i=2}^{k}\left(|U_i| + |U| + \binom{h_i}{k'} \times \partial_i\right)\right) = O\left(\sum_{i=2}^{k}\left(h_i + \binom{h_i}{k'} + \binom{h_i}{k'} \times \partial_i\right)\right).$$

In summary, the time complexity of the LCQ algorithm is

$$O\left(\binom{h_1}{k'} + \sum_{i=2}^{k}\left(h_i + \binom{h_i}{k'} + \binom{h_i}{k'} \times \partial_i\right)\right).$$

4.4 Extension

The FGSky query is more practical than the GSky query for two reasons: It can obtain results that the user expects, and it is also appropriate to the case where data are frequently updated. However, it may also face the problem of returning a prohibitively large number of results, which prevents users from making quick and rational decisions. To alleviate this problem, in this subsection, we discuss the FGSky query with a size constraint.

In the following, we present the Top l Flexible Group Skyline Query which aims to compute l G-Skylines that both satisfy user preferences and have the largest dominance ability.

Definition (Top l Flexible Group Skyline Query, TlFGSky): Given a data set D, two parameters k and l, and a user preference set $P \subseteq D$, the TlFGSky query returns l point groups $G \subseteq D$, such that G are not dominated by another group of size k, G contain all

Table 2 System parameters

Parameter	Values
Data set dimensionality(d)	**3**, 4, 5, 6
Group size (k)	3, 4, 5, **6**
Size of the unit set u_P ($\|u_P\|$)	1, **2**, 3, 4
Cardinality for Ind data sets (N)	**200000**, 400000, 600000, 800000
Cardinality for Ant data sets (N)	**20000**, 40000, 60000, 80000

points within P, and G have the highest scores. Here, the score of a group G is defined as

$$\text{Score}(G) = \left| \cup_{p \in G} \{p' \in P - G, p \prec p'\} \right|. \tag{1}$$

In case G-Skylines with the same dominance size are tie at rank l-th, only some of them are selected randomly and returned as the final TlFGSky query results.

In Fig. 1, we consider the T1FGSky query with $l = 1$ and $k = 4$. Due to Equation (1), we have the score of the hotel group $\{h_1, h_2, h_3, h_4\}$ is $Score(\{h_1, h_2, h_3, h_4\}) = |\{h_5, h_6, h_7, h_8, h_9, h_{10}, h_{11}\}| = 7$. Similarly, we have Score$(\{h_1, h_2, h_3, h_5\}) = Score(\{h_1, h_2, h_3, h_6\}) = Score(\{h_3, h_4, h_7, h_{11}\}) = 6$. For other G-Skylines shown in Fig. 1b, their scores are all equal to 7. Finally, we will choose one G-Skyline with score 7 randomly and return it as the final result of the T1FGSky query.

5 Performance evaluation

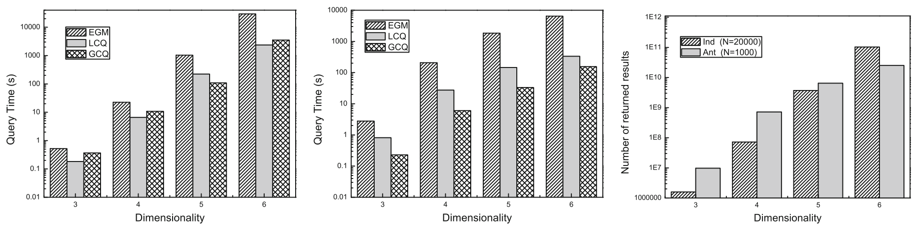
This section evaluates the performance of our algorithms in terms of query time (QT) and progressiveness. Here, QT represents the time between the submission of a query quest and the return of all the final query results. Similar to [29,30], the progressiveness is measured by QT as how many points reported by FGSky query during the query processing. In addition, we also show the number of the final FGSky query results (NR).

We have implemented all the algorithms for processing the FGSky query by C++. All the experiments were performed on a PC that is with the Intel(R) Core(TM) I5-3330S 2.7GHz CPU, 4GB main memory, and runs the Windows 7 operating system.

5.1 Performance on synthetic data sets

We generate the synthetic data sets by following the approach adapted in [1] and have completed a great number of experiments on the synthetic data sets with two popular distributions, i.e., independent (Ind) and anti-correlated (Ant). In the Ind data sets, each attribute value is generated independently on a uniform distribution, and for the Ant data sets, a point is good in one dimension but is bad in one or all of the other dimension(s) [11]. The points within the data sets are selected randomly to generate the user-specified point set P.

In each experiment, we vary one parameter at a time, while the other parameters are fixed to the default values, listed in bold. The parameters and their possible values are depicted in Table 2. The unit set u_P is computed as $\cup_{p \in P} u_p$, where $u_p = \{p\} \cup AncSet(p)$. Here, the values of all the parameters are set by referring to the related work [5,6,8,31].



(a) Ind **(b)** Ant **(c)** Number of results

Fig. 7 Experimental results versus dimensionality d

As shown in Table 2, we use different ranges of N for the Ind and Ant data sets. This is since it costs unacceptable QT when processing large Ant data sets.

5.1.1 Experimental results by varying dimensionality d

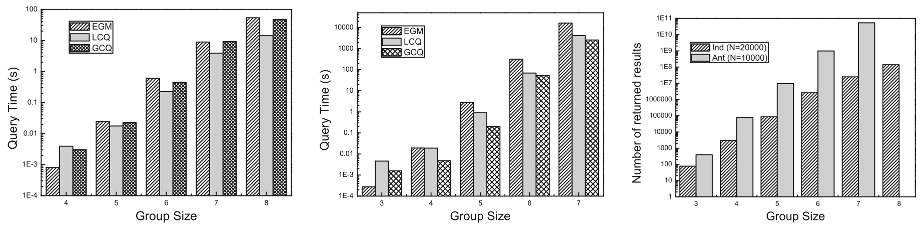
In the first part of experiments, we study the impact of the dimensionality d on the proposed algorithms when processing the data sets with $k = 6$, $|u_P| = 2$, and d is varied from 3 to 6 by a step of 1.

As shown in Fig. 7, NR and QT increase with the growth of d. The dimensionality d impacts the performance of the EGM, LCQ, and GCQ algorithms obviously. As expected, the performance of the three algorithms all degrades when the dimensionality is increasing. The reason is that the sizes of the indexes, MDG and LMDG, grow in an exponential rate as d increases. Accordingly, much more candidate groups need to be created and checked. Both the LCQ and GCQ algorithms perform better than the EGM algorithm which demonstrates the effectiveness of the proposed pruning strategies, the layered strategy and grouping strategy. From Fig. 7a, b, the LCQ algorithm is better than the GCQ algorithm for processing Ind data sets, but, when processing the Ant data sets, the GCQ algorithm outperforms the LCQ algorithm. This interesting phenomenon is due to the data distribution. Furthermore, compared to the GCQ algorithm, the LCQ algorithm owns a poorer performance in most cases. This is because the LCQ algorithm invokes itself recursively to generate candidate groups which brings many repetitive but redundant computation cost.

5.1.2 Experimental results by varying group size k

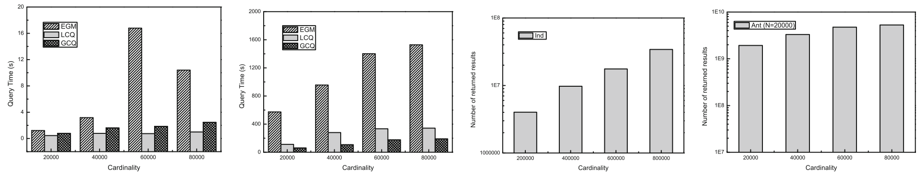
In the second part of experiments, we explore the impact of the group size k on the performance of the proposed algorithms. Specifically, k varies from 3 to 6 by a step of 1 with $d = 3$ and $|u_P| = 2$.

Figure 8 presents the experimental results about FGSky query as k grows. As show in the figure, the performance of the proposed algorithms is significantly affected by the group size k. As the increase of k, NR and QT of the three algorithms grow in turn. The reason is that in the FGSky query, the points of the first k skyline layers SL_i for $1 \leq i \leq k$ are considered. As k grows, cardinality of candidate groups consisting of points $p \in SL_i$ increases rapidly, and QT of the proposed algorithms grows in turn. Again, in most cases, the LCQ algorithm performs the best over Ind data sets, and the GCQ algorithm is the best to process the Ant data sets.



(a) Ind **(b)** Ant **(c)** Number of results

Fig. 8 Experimental results versus group size k



(a) Ind **(b)** Ant **(c)** Ind **(d)** Ant

Fig. 9 Experimental results versus cardinality N

5.1.3 Experimental results by varying $|u_P|$

In the third part of experiments, we inspect the performance impact of $|u_P|$ for the proposed algorithms by increasing $|u_P|$ from 1 to 5.

Table 3 illustrates NR and QT of the three algorithms with varying $|u_P|$. Different from the terms d and k, QT of the three algorithms reduces significantly as $|u_P|$ increases. This is as expected; for a given point set P, any FGSky query result of size k has contained all points in the unit set u_P. Accordingly, we only need to compute the other $k - |u_P|$ points from $MDG - u_P$ or $LMDG - u_P$. The larger $|u_P|$ is, the less candidate groups are computed and checked. In addition, NR reduces as $|u_P|$ raises. In most cases, the GCQ algorithm requires the least QT in comparison with the EGM and LCQ algorithms by varying $|u_P|$ from 1 to 4.

5.1.4 Experimental results by varying the cardinality N

In the fourth part of experiments, we examine the effect of the cardinality N on the performance of the three algorithms. Considering the query time, we varied N from 200000 to 800000 for Ind data sets, and for Ant data sets, N is changed from 20000 to 100000.

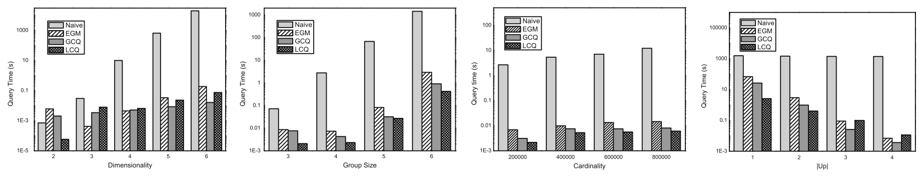
Figure 9 depicts NR and QT of the three algorithms for different values of N. The effect of cardinality N is not as important as that of the dimensionality d and the group size k. When N goes up, in most cases, QT of the proposed algorithms raises obviously, and NR grows also. For the Ind data sets, the LCQ algorithm owns the best scalability for the cardinality N. The GCQ algorithm behaves best when processing the Ant data sets.

5.1.5 Performance in comparison with the Naive algorithm

In this subsection, we add a baseline algorithm, namely Naive, in the experiments. The Naive algorithm generates all the G-Skylines by G-MDS [8], which is the state-of-the-art algorithm for the GSky query, and then picks out the ones that the user expects. We study the

Table 3 Experimental results versus $|u_P|$

$\lvert u_P \rvert$	Ind				Ant			
	NR	EGM(s)	LCQ(s)	GCQ(s)	NR	EGM(s)	LCQ(s)	GCQ(s)
1	29,651,855	9.6055	7.3334	0.7401	–	–	–	–
2	1,963,211	0.6348	0.4196	0.2281	1,941,996,248	572.2150	106.6010	49.4129
3	283,185	0.0299	0.0279	0.0707	16,587,947	5.8390	6.2308	0.3874
4	7575	0.0011	0.0028	0.0117	111,957	0.0373	0.2175	0.0068



(a) Dimensionality **(b)** Group Size **(c)** Cardinality **(d)** $|u_p|$

Fig. 10 Experimental results over Ind data sets

Table 4 Experimental results versus k over the Hou

k	NR	EGM(s)	LCQ(s)	GCQ(s)
3	287	0.0002	0.0042	0.0016
4	41,264	0.0106	0.0108	0.0026
5	4,086,756	1.2049	0.4293	0.1630
6	310,560,542	96.8637	25.0351	34.8211

Table 5 Experimental results versus $|u_P|$ over the Hou

$|u_P|$	NR	EGM(s)	LCQ(s)	GCQ(s)
1	286,107,869	79.4381	2.8683	10.9210
2	3,962,986	1.1575	0.4242	0.1287
3	41,838	0.00999	0.0214	0.0028
4	288	0.00039	0.0059	0.00223

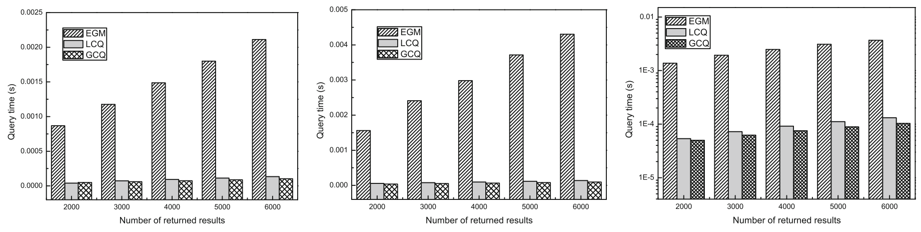
performance of our proposed algorithms in comparison with the Naive algorithm over Ind data sets. The experimental parameters are set due to the query time of the Naive algorithm.

As shown in Fig. 10, the three proposed algorithms all outperform the Naive algorithm with varying the dimensionality d ($N = 20000$, $k = 4$, $|u_P| = 2$), the group size k ($N = 200000$, $d = 3$, $|u_P| = 2$), the cardinality N ($d = 3$, $k = 4$, $|u_P| = 2$), and the size of u_P ($N = 200000, d = 3, k = 6$). In most cases, the proposed algorithms, EGM, GCQ, and LCQ, are three orders of magnitude faster than the Naive algorithm.

5.2 Performance on the real data sets

Furthermore, the proposed algorithms are also evaluated based on household (Hou) which is a real data set that is also utilized in [30]. The Hou contains tuples of the percentage of an American families annual income. Its cardinality is 127,000. We takes into account the attributes including the expenditures of gas, electricity, water, and heating. In this subsection, we report the experimental results on the Hou data set.

Tables 4 and 5 show QT of the three algorithms on the Hou data set. The results over the real data set are consistent with the ones of the experiments on the Ind and Ant data sets. As k increases, NR and QT of the proposed algorithms grow clearly. Most of the time, the GCQ algorithm requires the least QT with changing k from 3 to 6. In addition, NR and QT of the proposed algorithms reduce when varying $|u_P|$ from 1 to 4. With the growth of $|u_P|$, the GCQ algorithm needs the least QT.



(a) Ind **(b)** Ant **(c)** Hou

Fig. 11 Progressiveness comparison

5.3 Progressiveness performance

In this set of experiments, we analyze the progressiveness of the EGM, GCQ, and LCQ algorithms through evaluating QT by changing the number of reported FGSky query results (NR). The accumulated QT of the algorithms is reported as NR grows. For two algorithms, when returning the same number of query results the one needs less QT is considered having better progressiveness. As depicted in Fig. 11, when returning the same number of the FGSky query results, the GCQ algorithm always needs the minimal QT. This indicates the GCQ algorithm has the best progressiveness among the proposed algorithms.

6 Conclusions

In this paper, we investigate the FGSky query. Particularly, the FGSky query reports the expected G-Skylines which can satisfy the user preferences. To process the FGSky query, effective algorithms which can output the final query results progressively are proposed. In addition, we demonstrate the effectiveness, efficiency, and progressiveness of the proposed algorithms by extensive experiments.

As shown in the experimental results, the FGSky query requires too long query time over high-dimensional data sets which will influence its practicality. To address this problem, one approach is to apply dimensionality reduction techniques to covert the date sets to subspaces with lower dimensionality as mentioned in [32]. In addition, we could design approximate algorithms to get good query performance. It is a significant and challenge work to process the FGSky query over high dimensionality. We will take it as our further work.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable and helpful comments on improving the manuscript. The research was supported by the NSFC (Grant Nos. 61802032, 61772182, 61602170), the Key Program of NSFC (Grant No. 61432005), the International (Regional) Cooperation and Exchange Program of NSFC (Grant No. 61661146006), the Emergency Special Project of NSFC (Grant No. 61751204), the Key Area Research Program of Hunan (2019GK2091), and the Hunan Province Key Laboratory of Industrial Internet Technology and Security (2019TP1011). Xu Zhou is the corresponding author of this paper.

References

1. Borzsony S, Kossmann D, Stocker K (2001) The skyline operator. In: Proceedings of the international conference on data engineering, pp 421–430

2. Chung Y-C, Su I-F, Lee C (2013) Efficient computation of combinatorial skyline queries. Inf Syst 38(3):369–387
3. Im H, Park S (2012) Group skyline computation. Inf Sci 188:151–169
4. Magnani M, Assent I (2013) From stars to galaxies: skyline queries on aggregate data. In: Proceedings of the international conference on extending database technology. ACM, pp 477–488
5. Zhang N, Li C, Hassan N, Rajasekaran S, Das G (2014) On skyline groups. IEEE Trans Knowl Data Eng 26(4):942–956
6. Liu J, Xiong L, Pei J, Luo J, Zhang H (2015) Finding pareto optimal groups: group-based skyline. In: Proceedings of the international conference on very large data bases, vol 8, no 13
7. Xu Z, Li K, Yang Z, Xiao G, Li K (2018) Progressive approaches for pareto optimal groups computation. IEEE Trans Knowl Data Eng 99:1
8. Wang C, Wang C, Guo G, Ye X, Yu PS (2018) Efficient computation of g-skyline groups. IEEE Trans Knowl Data Eng 30(4):674–688
9. He Z, Lo E (2012) Answering why-not questions on top *k* queries. In: Proceedings of the IEEE international conference on data engineering (ICDE), pp 750–761
10. Islam MS, Zhou R, Liu C (2013) On answering why-not questions in reverse skyline queries. In: Proceedings of the IEEE international conference on data engineering (ICDE), pp 973–984
11. Gao Y, Liu Q, Chen G, Zheng B, Zhou L (2015) Answering why-not questions on reverse top *k* queries. In: Proceedings of the international conference on very large data bases (VLDB), vol 8(7), pp 738–749
12. Chen L, Lin X, Hu H, Jensen CS, Xu J (2015) Answering why-not questions on spatial keyword top *k* queries. In: Proceedings of the IEEE international conference on data engineering (ICDE), pp 279–290
13. Liu Q, Gao Y, Zheng B, Zhou L (2016) Answering why-not and why questions on reverse top *k* queries. VLDB J 25(6):867–892
14. Islam MS, Liu C, Li J (2016) Efficient answering of why-not questions in similar graph matching. In: Proceedings of the IEEE international conference on data engineering (ICDE), pp 2672–2686
15. Chen L, Xu J, Jensen CS, Li Y (2016) Yask: a why-not question answering engine for spatial keyword query services. In: Proceedings of the international conference on very large data bases (VLDB), vol 9(13), pp 1501–1504
16. Tran QT, Chan CY (2010) How to conquer why-not questions. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
17. Wan Q, Wong RC-W, Ilyas IF, Özsu MT, Peng Y (2009) Creating competitive products. Proc VLDB Endow 2(1):898–909
18. Su I-F, Chung Y-C, Lee C (2010) Top-k combinatorial skyline queries. In: Database systems for advanced applications. Springer, pp 79–93
19. Zhou X, Li K, Yang Z, Xiao G, Li K (2018) Progressive approaches for pareto optimal groups computation. IEEE Trans Knowl Data Eng 31(3):521–534
20. Zhu H, Zhu P, Li X, Liu Q, Xun P (2017) Parallelization of group-based skyline computation for multi-core processors. Concurr Comput Pract Exp 29(3):e4195
21. Zhu H, Zhu P, Li X, Liu Q (2017) Top *k* skyline groups queries. In: Proceedings of the ninth international conference on extending database technology (EDBT)
22. Zhu H, Li X, Liu Q, Xu Z (2019) Top-k dominating queries on skyline groups. IEEE Trans Knowl Data Eng 32:1431–1444
23. Yu W, Liu J, Pei J, Xiong L, Chen X, Qin Z (2019) Efficient contour computation of group-based skyline. IEEE Trans Knowl Data Eng 32:1317–1332
24. Miao X, Gao Y, Guo S, Chen G (2018) On efficiently answering why-not range-based skyline queries in road networks. IEEE Trans Knowl Data Eng 30(99):1
25. He Z, Lo E (2014) Answering why-not questions on top *k* queries. IEEE Trans Knowl Data Eng 26(6):1300–1315
26. Chen L, Gao Y, Wang K, Jensen CS, Chen G (2016) Answering why-not questions on metric probabilistic range queries. In: Proceedings of the IEEE international conference on data engineering (ICDE), pp 767–778
27. Chen L, Li Y, Xu J, Jensen CS (2017) Direction-aware why-not spatial keyword top-k queries. In: Proceedings of the IEEE international conference on data engineering (ICDE), pp 107–110
28. Chen L, Li Y, Xu J, Jensen CS (2018) Towards why-not spatial keyword top *k* queries: a direction-aware approach. IEEE Trans Knowl Data Eng 30(99):796–809
29. Ding X, Jin H (2012) Efficient and progressive algorithms for distributed skyline queries over uncertain data. IEEE Trans Knowl Data Eng 24(8):1448–1462
30. Zhou X, Li K, Zhou Y, Li K (2016) Adaptive processing for distributed skyline queries over uncertain data. IEEE Trans Knowl Data Eng 28:371–384
31. Ren W, Lian X, Ghazinour K (2019) Skyline queries over incomplete data streams. VLDB J 28:961–985

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