Knowledge-Based Systems 182 (2019) 104795

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Efficient processing of top *k* group skyline queries☆

Zhibang Yang [a,b], Xu Zhou [b,*], Kenli Li [b], Guoqing Xiao [b], Yunjun Gao [c], Keqin Li [b]

[a] College of Computer Engineering and Applied Mathematics, Changsha University, Changsha 410003, Hunan, China
[b] College of Information Science and Engineering, Hunan University, Changsha 410082, Hunan, China
[c] College of Computer Science, Zhejiang University, Hangzhou 310027, Zhejiang China

ARTICLE INFO

Article history:
Received 22 June 2018
Received in revised form 21 March 2019
Accepted 3 June 2019
Available online 8 June 2019

Keywords:
Data management
Group skyline query
Top *k* query

ABSTRACT

For a given multi-dimensional data set, a group skyline query returns the optimal groups not dominated by any other group of equal size. The group skyline query is a powerful tool in many applications that call for optimal groups. However, it is common to return a large number of results which make users overwhelmed since it prevents them from making quick and rational decisions. To address this problem, we first identify and formulate a top *k* group skyline (T*k*GSky) query which returns *k* optimal groups dominating the highest number of points in the given data set. Next, new pruning strategies are presented to reduce the search space. Then, we propose efficient algorithms by exploiting novel techniques including a grouping strategy, a hybrid strategy, and a point-based replacement strategy, respectively. Finally, we also develop an approximate algorithm to further improve the T*k*GSky query performance. The performance of the proposed algorithms is studied by extensive experiments over synthetic and real datasets.

1. Introduction

1.1. Background

The skyline query reports the best points (skylines) that are not dominated by any other point in a given data set [1]. It is a powerful tool in many decision making applications [2,3] and receives growing attention in the database community. However, the skyline query and its variants mainly focus on analyzing individual points [4]. Accordingly, they cannot be utilized in many real-life applications that require to select a point group.

In [4], Liu et al. formulated the G-Skyline (GSky) query by generalizing the original definition of skyline to the group-based skyline (G-Skyline). Given two groups G and G' of equal size, 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'. A group is called G-Skyline if it is not dominated by any other group of the same size.

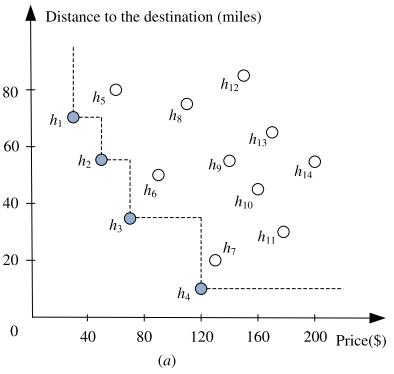
Example. The GSky query can apply in many real-life applications including a coach building a basketball team, an advertising company selecting billboards to serve its advertisements, and an organizer selecting some hotels for cooperation, etc. Consider an organizer of a conference needs to choose three hotels for cooperation. Fig. 1(a) depicts a hotel data set $H=\{h_1, h_2, \ldots, h_{14}\}$ that contains fourteen candidate hotels (data points). Each hotel is with two features (dimensions), distance to the destination and price. Without loss of the generality, small values of the two attributes are preferable. The hotels h_1, h_2, h_3, and h_4 which are not dominated by any other hotel are the skylines. As shown in Fig. 1(c), the GSky query presented in [4] returns the hotel groups $\{h_1, h_2, h_3\}$, $\{h_1, h_2, h_4\}$, $\{h_1, h_3, h_4\}$, and $\{h_2, h_3, h_4\}$ that only consist of skylines. Moreover, it could also get the G-Skylines, such as $\{h_3, h_4, h_6\}$, $\{h_1, h_4, h_7\}$, $\{h_4, h_7, h_{11}\}$, to name just a few, which contain non-skylines. Consider the G-Skyline $G=\{h_3, h_4, h_6\}$. The non-skyline h_6 and its parent h_3 are both included in the G-Skyline G. For the hotel groups not in Fig. 1(c), they are not G-Skylines because of being dominated by at least a hotel group of equal size. For instance, the hotel group $\{h_1, h_5, h_6\}$ is not a G-Skyline. This is since it is dominated by the hotel group $\{h_1, h_3, h_5\}$ for $h_3 \prec h_6$.

Compared to the group queries in [2,5,6], the GSky query proposed in [4] could get more comprehensive results. As well as the G-Skylines only consisting of skylines, it also reports the G-Skylines that include non-skyline points. However, the G-Skyline query may return a large number of results especially over a data set with a large cardinality, high dimensionality, or with a large

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2019.06.003.

* Corresponding author.
E-mail addresses: zbyang@hnu.edu.cn (Z. Yang), zhouxu@hnu.edu.cn (X. Zhou), lkl@hnu.edu.cn (K. Li), xiaoguoqing@hnu.edu.cn (G. Xiao), gaoyj@zju.edu.cn (Y. Gao), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.knosys.2019.06.003
0950-7051/



(a)

Hotel	Price	Distance
h_1	30	70
h_2	50	55
h_3	70	35
h_4	120	10
h_5	60	80
h_6	90	50
h_7	130	20
h_8	110	75
h_9	140	55
h_{10}	160	45
h_{11}	180	30
h_{12}	150	85
h_{13}	170	65
h_{14}	200	55

(b)

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

(c)

Fig. 1. The G-Skyline query example of a hotel set.

group size. In the experiments, over the anti-correlated datasets with k=3, d=3, and N=40 000, the number of the G-Skylines is up to several tens of millions.

A large number of G-Skylines could create emotional stress to users and act as powerful barriers to rational decisions as pointed out in [7]. To address this issue, Liu et al. [4] developed an interesting variant of the GSky query, namely PG-Skyline. It reports partial G-Skylines due to a pg-dominance operator. Given two groups G and G' of size l and a parameter p for $p \leq l$, G pg-dominates G' if for groups $G_p \subseteq G$ and $G'_p \subseteq G'$ with $|G_p| = |G'_p| = p$, $G_p \prec_g G'_p$. Although this approach is useful to reduce the GSky query results, it is difficult for users to specify the parameter p and it cannot report manageable size of results.

1.2. Our contributions

To resolve this problem, we investigate the GSky query with a size constraint and formulate a top k GSky (TkGSky) query. To improve the query performance, we first propose some pruning strategies to narrow the search space, and then present efficient algorithms based on new techniques such as the grouping strategy, the hybrid strategy, and the point-based replacement strategy.

In brief, our contributions are summarized as follows.

- We formulate the top k GSky (TkGSky) query to retrieve k G-Skylines with the highest number of dominated points.
- We exploit the properties of the TkGSky query and develop some prune strategies to reduce the search space.
- We propose three efficient algorithms, which are a fast algorithm (FA), a hybrid algorithm (HA), and a point-based replacement (PR) algorithm for the TkGSky query.
- We present an approximate algorithm to further improve the TkGSky query performance.
- We conduct an extensive experimental study to verify the effectiveness and efficiency of the proposed algorithms.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we introduce the TkGSky query and its properties. In Section 4, we design efficient exact and approximate algorithms for the TkGSky query. In Section 5, we evaluate performance of the proposed algorithms by extensive experiments. In Section 6, we conclude the paper and also expatiate the directions for future work.

2. Related work

In this section, we overview the closely related work about the TkGSky query.

2.1. Traditional skyline queries

The approaches for traditional skyline queries can be classified into two categories which are index-independent algorithms and index-based algorithms [8]. The OSP algorithm in [3] is considered the most efficient approach without an index. Among the index-based algorithms, the BBS algorithm in [1] based on R-tree is progressive and acknowledged to be I/O optimal. Recently, the skyline query receives growing attention, and many skyline query variants have been proposed. Gao et al. [8] researched the reverse k-skyband and ranked reverse skyline query. In [9–15], the skyline queries over distributed or parallel environments were investigated.

The traditional skyline query always returns a large number of query results. To resolve this problem, many approaches were proposed to identify k representative results with the highest dominant capacity or the maximum diversity. Lu et al. [16] concerned the case when the actual cardinality of skyline query results is less than the desired cardinality k. Gao et al. [17] introduced the most desirable skyline object (MDSO) query to return the most preferable k skyline objects. They presented a new ranking criterion that considers, for each skyline object s, the number of the objects dominated by s and their accumulated weights. Bai et al. [18] were concerned about calculating the k representative skylines over data streams. In addition, Magnani et al. [19] focused on the representative skyline queries in terms of both the significance and diversity of results.

2.2. Group skyline queries

Group skyline queries are very important in many applications that need to compute optimal groups of points. The existing group skyline query can be classified into the following two categories.

The first category uses an aggregate point to represent a group. The dominance relationship of groups depends on the traditional dominance relationship of the aggregate points. Chung et al. [5] extended the traditional skyline query and formulated a combinatorial skyline query, namely CSQ, which is to find the outstanding skyline combinations. Magnani et al. [20] introduced the aggregate skyline query that merges two basic database operators, skyline and group by. Zhang et al. [6] 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 the aggregate-based group dominance relationship. Jiang et al. [21] 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. In the group skyline query above, it is difficult to specify an appropriate

aggregate function. Moreover, it may overlook significant groups that contain non-skylines [4]. In [22], we formulated a top k favorite probabilistic products query which computes k favorite products due to the preferences of different customers. After that, in [23], we investigated a constrained optimal product combination problem under price promotion. In [24], Zeng et al. proposed a novel M-Skyline query model based on sunk cost. This query can offer backup recommendation.

The second category defines the dominance relationship between two groups due to the pair-wise dominance between points in these groups [25]. Liu et al. [4] first formulated the G-Skyline (GSky) query based on this new definition of group dominance. They presented a directed skyline graph (DSG) to capture the dominant relationship between points within the first l skyline layers for group size l. The DSG is efficient to speed up the GSky query. Yu et al. [25] 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 non-skyline points. Recently, Wang et al. [26] developed the minimum g-skyline support structure, called MDG, without redundant points. Compared to the traditional skyline computation, the G-skyline is much more complicated and expensive. To accelerate the G-Skyline query, Zhu et al. [27] developed parallel G-Skyline query algorithms for multicore processors.

2.3. Top k dominating queries

The top k dominating query returns k data objects with the highest number of objects dominated in a given data set. By combining the advantages of both the top k query and the skyline query, the top k dominating query has the nice properties that are the number of results is controllable, the result is scaling invariant, and no user-defined ranking function is required.

In recent years, the top k dominating query receives growing attention in database fields. Tiakas et al. [28] formulated the top k dominating query in metric space and proposed progressive algorithms which can report the query results in a progressive manner. Miao et al. [29] investigated the top k dominating queries over incomplete data where the data have some missing dimensional value(s). To improve the query performed on massive data, Han et al. [30] studied top k dominating query on massive data. They proposed the novel table-scan-based algorithm where the early termination approach is introduced.

The continuous top k dominating queries were researched in [31]. Santoso et al. [31] proposed a new indexing structure, the close dominance graph (CDG), to provide comprehensive information about the dominance relationship between points. He et al. [32] researched the why-not top k dominating query which helps users to know why their expected results not in the query results. Moreover, the top k dominating query over uncertain data has been researched in [33]. Zhan et al. [33] defined a new model for the top k dominating query on multi-dimensional uncertain data, and applied some simple statistics information of the objects to boost the query performance. Amagata et al. [34] researched the top k dominating query in distributed environments. As well as an exact algorithm, they also present approximate algorithms to get better query performance with sacrificing some accuracy of the results.

The frequently used symbols are summarized in Table 1.

3. The TkGSky query problem

In this section, we formulate the TkGSky query, and introduce the direct skyline graph proposed in [4].

Table 1
The summary of frequently used notations.

Notation	Definition
D	The dataset
N	The size of the dataset
G	The group of points
l	The group size
k	The number of returned results
SL_i	The ith skyline layer
$Score(G)$	Score of the group G
$Score^u(G)$	Upper bound of $Score(G)$
$p.layer$	The skyline layer that the point p belongs to
$ParSet(p)$	The parent set of p
$ChilSet(p)$	The child set of p
$DChilSet(p)$	The direct child set of p

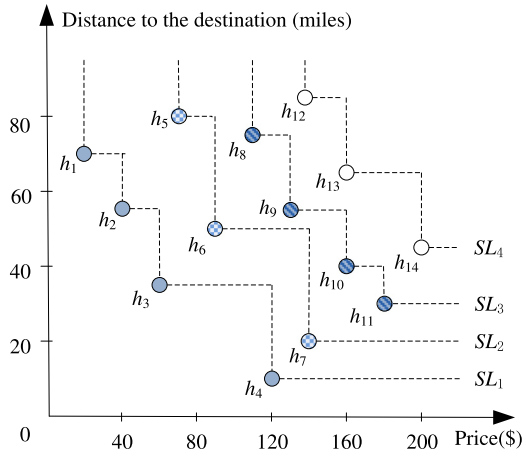


Fig. 2. The skyline layers of the hotel set in Fig. 1.

3.1. Problem definition

Given a d-dimensional dataset D, a point $p \in D$ is denoted as $<p[1], p[2], \ldots, p[d]>$. Here $p[i]$ is the ith dimensional value of p for $1 \le i \le d$. Without loss of the generality, small value is preferred in each dimension. A point p' dominates another point p, denoted as $p' \prec p$, if and only if for all i, $p'[i] \le p[i]$ and for at least one i, $p'[i] < p[i]$ for $1 \le i \le d$ [35]. The point not dominated by any other point is called skyline.

Definition 1 (*Skyline Layer*)**.** Given a dataset D and a parameter l, the first skyline layer SL_1 includes all the skylines of D, and the ith skyline layer SL_i consists of the skylines of $D - \bigcup_{j=1}^{i-1} SL_j$ for $1 < i \le l$.

In Fig. 1, the hotels could be organized as four skyline layers which are $SL_1 = \{h_1, h_2, h_3, h_4\}$, $SL_2 = \{h_5, h_6, h_7\}$, $SL_3 = \{h_8, h_9, h_{10}, h_{11}\}$, and $SL_4 = \{h_{12}, h_{13}, h_{14}\}$ as depicted in Fig. 2.

Definition 2 (*Group Dominance [4]*)**.** Given two l-point groups $G, G' \subseteq D$, G g-dominates G', denoted as $G \prec_g G'$, if and only if there are two permutations of G and G', $G = \{p_1, p_2, \ldots, p_l\}$ and $G' = \{p'_1, p'_2, \ldots, p'_l\}$, satisfying $p_i \preceq p'_i$ for $1 \le i \le l$ 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 (*Group Skyline Query, GSky*)**.** Given a dataset D and a group size l, the GSky query returns l-point groups $G \subseteq D$ that are not g-dominated by any other group of size l, formally,

$$\text{GSky}(D, l) = \{G \subseteq D \mid \nexists G' \subseteq D, G' \prec_g G, |G| = |G'| = l\}.$$

The G-Skyline is the group G not g-dominated by any other group of equal size.

For two hotel groups $\{h_1, h_2, h_3\}$ and $\{h_5, h_6, h_8\}$, we have $G \prec_g G'$ because $h_1 \prec h_5$, $h_2 \prec h_8$, and $h_3 \prec h_6$. Therefore, $\{h_5, h_6, h_8\}$ is not a G-Skyline. The group $\{h_1, h_2, h_3\}$ not dominated by any other group of equal size is a G-Skyline.

Definition 4 (*Top k Group Skyline Query, TkGSky*)**.** Given a dataset D and a parameter k, the TkGSky query returns k G-Skylines, G, such that they have the highest scores. The score of a group G is defined as

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

(Search result diversification has become important for improving user satisfactory, and we pick out only one of the G-Skylines with the same scores.)

Going back to the example in Fig. 1 with l=3 and k=2, for the hotel group $\{h_1, h_2, h_3\}$, its score is $Score(\{h_1, h_2, h_3\}) = |\{h_5, h_6, h_8, h_9, h_{10}, h_{12}, h_{13}, h_{14}\}| = 8$. After computing the scores of other hotel groups that are G-Skylines, we have the G-Skylines $\{h_1, h_3, h_4\}$, $\{h_2, h_3, h_4\}$, $\{h_1, h_2, h_4\}$, $\{h_3, h_4, h_6\}$, $\{h_3, h_4, h_7\}$ with top 2 highest scores. Because the hotel groups $\{h_1, h_3, h_4\}$ and $\{h_2, h_3, h_4\}$ are with a same score, we only choose one of them. Similarly, one of the hotel groups $\{h_1, h_2, h_4\}$, $\{h_3, h_4, h_6\}$, and $\{h_3, h_4, h_7\}$ is selected randomly and returned.

Lemma 3.1. *Given a G-Skyline $G \subseteq D$ with $G_1 = G \cap SL_1$, it holds that*

$$\begin{aligned} Score(G) &= Score(G_1) - |G - G_1| \\ &= |\cup_{p \in G_1} \{p' \in D - G, p \prec p'\}|. \end{aligned} \tag{1}$$

Proof. Due to Definition 4, we have

$$\begin{aligned} Score(G) &= |\cup_{p \in G} \{p' \in D - G, p \prec p'\}| \\ &= |\cup_{p \in G_1 \cup (G - G_1)} \{p' \in D - G, p \prec p'\}| \\ &= |\left(\cup_{p \in G_1} \{p' \in D - G, p \prec p'\}\right) \cup \left(\cup_{p \in G - G_1} \{p' \in D - G, p \prec p'\}\right)|. \end{aligned} \tag{2}$$

For any point $p' \in G - G_1$, there is at least one point $p \in G_1$ satisfying $p \prec p'$. For a given point p'' with $p' \prec p''$, we have $p \prec p''$ due to the transitivity of the dominance relationship. This means that the points dominated by p' are also dominated by p, and we have $\{p'' \in D, p' \prec p''\} \subset \{p'' \in D, p \prec p''\}$. Then, we have $\left(\cup_{p \in G - G_1} \{p' \in D - G, p \prec p'\}\right) \subset \left(\cup_{p \in G_1} \{p' \in D - G, p \prec p'\}\right)$, and this lemma holds in turn. □

Lemma 3.1 aforementioned can be utilized to reduce the cost of computing the scores of candidate groups in the TkGSky query. For instance, we have $Score(\{h_2, h_3, h_6\}) = Score(\{h_2, h_3\}) - |\{h_6\}| = 7$. due to Lemma 3.1.

3.2. The directed skyline graph (DSG)

In [4], it organizes the skyline layers by a directed skyline graph (DSG) where a node represents a point and an edge represents the dominance relationship between two different points. Besides, it also introduces that each G-Skyline of size l only contains the points within the first l skyline layers [4], and the DSG is built over the first l skyline layers. The structure of each node is [*layer index*, *point index*, *parent set*, *child set*]. Here the layer index is the skyline layer that the point lies on and the point index uniquely identifies the point. Additionally, the parent and child set is defined as follows.

Definition 5 (*Parent Set, ParSet*)**.** Given a point $p \in D$, the parent set of p includes all the points dominating p, formally,

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

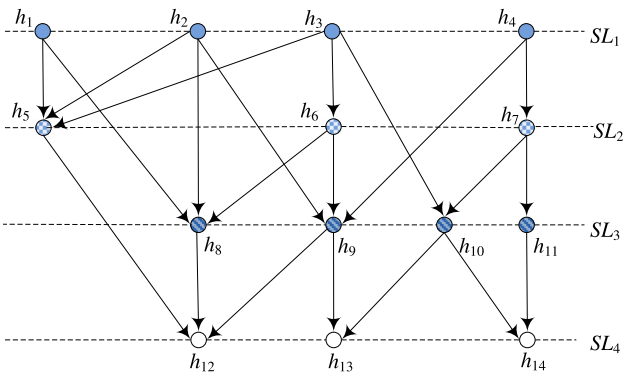


Fig. 3. The DSG of the hotel set in Fig. 1.

Moreover, for a point group $P' \subseteq D$, we have

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

Definition 6 (*Child Set, ChiSet*)**.** Given a point $p \in D$, the child set of p contains all the points dominated by p, and it can be defined as

$$\text{ChiSet}(p) = \{p' \in D | p \prec p'\}.$$

Moreover, for a point group $P' \subseteq D$, the child set is

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

The DSG in [4] is useful to accelerate the GSky query, and it can be reused very well in the GSky queries where the group sizes are no more than l. To process the GSky query with an arbitrary size, we create the DSG over the entire dataset. Consequently, when processing the DSG query with size larger than l, it is unnecessary to update the DSG by adding more skyline layers in advance. Additionally, it is helpful to compute the score of each G-Skyline. This is since all the children of the points are pre-computed and stored in the corresponding nodes.

Fig. 3 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 $h_1 \rightarrow h_5$ means that h_5 is dominated by h_1. In the node representing h_5, it stores its parent set $\{h_1, h_2, h_3\}$ and its child set $\{h_{12}\}$ as well as its layer index 2 and point index 5.

4. Approaches for the T*k*GSky query

In this section, we propose three algorithms, i.e., the fast algorithm (FA), hybrid algorithm (HA), and point-based replacement (PR) algorithm, for the TkGSky Query.

4.1. The fast algorithm

In this subsection, we propose the FA to process the TkGSky Query. In the FA, the grouping strategy and the upper bound pruning strategy are introduced to improve the query performance.

Due to Definition 4 and Lemma 3.1, for a given G-Skyline G, its score is closely related to the dominance capacity of the points $p \in SL_1 \cap G$. Inspired by this, we divide the G-Skylines G into groups according to $|SL_1 \cap G|$. Moreover, the following upper bound pruning strategy is utilized to identify and prune unqualified G-Skylines as soon as possible.

Definition 7 (*Upper Score of a Group G, Score^u(G)*)**.** For a given group G of size l, the upper score of G can be computed as

$$\text{Score}^u(G) = \sum_{p \in G \cap \text{SL}_1} |\text{ChiSet}(p)| - (l - |G \cap \text{SL}_1|).$$

For the G-Skyline $\{h_1, h_3, h_6\}$, we have

$Score^u(\{h_1, h_3, h_6\}) = |\text{ChiSet}(h_1)| + |\text{ChiSet}(h_3)| - (3 - |\{h_1, h_3\}|) = 9.$

Lemma 4.1. *For a given G-Skyline G, it can be pruned safely if Score^u(G) is less than the current kth largest score r.*

Proof. Since $Score(G) \le Score^u(G)$ and $Score^u(G) < r$, it holds that $Score(G) < r$. Hence G is not a result of the TkGSky query due to Definition 4, and this lemma holds. □

Algorithm 1 The Fast Algorithm (FA) for TkGSky

Input: A DSG DS over D, group size l, and the number of results k
Output: k G-Skylines with the highest scores

```
1: Compute the G-Skylines by the UWise+ algorithm [4]
2: Part the G-Skylines into l groups and initialize a set V_i to store the
   groups containing i points within SL_1 for 1≤i≤l
3: Initialize a max-heap H to store the G-Skylines with top k highest
   scores
4: Initialize the kth highest score r←0
5: for i←l to 1 do
6:    if V_i is not empty then
7:       for each G-Skyline G∈V_i do
8:          if Score^u(G)≥r then
9:             Score(G)←Score(G)−|G−G∩SL_1| //Lemma 3.1
10:            if Score(G)≥r then
11:               Update H and r using <G, Score(G)>
12: Return k G-Skylines with the highest scores from H
```

As illustrated in Algorithm 1, the FA first computes and divides the G-Skylines G due to $|G \cap SL_1|$ (Lines 1 and 2). In Line 1, the UWise+ algorithm [4] is applied to compute all the G-Skylines. Line 3 initializes a max-heap H to store the G-Skylines with top k highest scores. The kth highest score r is initialized to 0 in Line 4. Lines 5 to 11 are a for loop that computes the G-Skylines with top k highest scores. Here i varies from l to 1. This means the groups with more points within SL_1 are given priority to be processed. If V_i is not empty, Lines 7 to 11 are executed to compute the G-Skylines $G \in V_i$ whose scores exceed r. For each G-Skyline $G \in V_i$, Line 8 checks whether its upper bound of score is no less than r. If it returns "no", G is pruned due to Lemma 4.1, otherwise Line 9 computes the accurate score of G based on Lemma 3.1. Then, Lines 10 and 11 pick out the groups G with $Score(G) \ge r$ and update H and r using $<G, Score(G)>$. At last, we select k G-Skylines with different scores from H, and return them as the final results.

Example. Consider the example in Fig. 1 with $k=3$ and $l=2$. The FA first computes the hotel groups that are G-Skylines and part them into 3 groups due to $|G \cap SL_1|$. We have $V_1 = \{\{h_4, h_7, h_{11}\}\}$, $V_2 = \{\{h_1, h_3, h_6\}, \{h_2, h_3, h_6\}, \{h_3, h_4, h_6\}, \{h_1, h_4, h_7\}, \{h_2, h_4, h_7\}, \{h_3, h_4, h_7\}\}$, and $V_3 = \{\{h_1, h_2, h_3\}, \{h_1, h_2, h_4\}, \{h_1, h_3, h_4\}, \{h_2, h_3, h_4\}\}$. The hotel groups $G \in V_3$ are processed firstly. The hotel groups $\{h_1, h_2, h_4\}$, $\{h_1, h_3, h_4\}$, and $\{h_2, h_3, h_4\}$ are inserted into H since they have top 2 highest scores. The current 2nd highest score is 9. Next, the hotel groups $\{h_3, h_4, h_6\}$ and $\{h_3, h_4, h_7\}$ within V_2 are inserted into H. Then, the hotel group $\{h_4, h_7, h_{11}\} \in V_1$ is pruned directly because $Score^u(\{h_4, h_7, h_{11}\}) = Score(\{h_4\}) - (3 - |\{h_7, h_{11}\}|) = 5 < 9$. Finally, we get the hotel groups $\{h_1, h_2, h_4\}$, $\{h_1, h_3, h_4\}$, $\{h_2, h_3, h_4\}$, $\{h_3, h_4, h_6\}$, and $\{h_3, h_4, h_7\}$ which have top 2 highest scores. By selecting two G-Skylines with different scores, we get the final TkGSky query results.

Complexity. The FA computes the G-Skylines by applying the UWise+ algorithm in [4]. The UWise+ algorithm generates candidate groups incrementally. It first generates $\binom{\hbar}{1}$ unit group sets only consisting of one unit group of the point $p \in SL_1$. Here $\hbar$ is the total cardinality of the points within SL_1. Then, by combining these unit groups of size 1, new unit group sets including two unit groups are generated. The maximum size of these unit group sets is $\binom{\hbar}{2}$. Similarly, we have the maximum size of unit group sets of size i is $\binom{\hbar}{i}$ for $1 \le i \le l$. In conclusion, the time complexity of the UWise+ algorithm is

$$O\left(\sum_{i=1}^{l} \binom{\hbar}{i}\right) \le O(2^{\hbar}).$$

Next, the G-Skylines with top k highest scores are computed in Lines 2 to 11. Assume that the number of the G-Skylines is $\hbar_g$. It costs $O(\hbar_g)$ to part the G-Skylines into l groups. The time cost of creating the max-heap H is $O(\hat{\hbar_g})$ where $\hat{\hbar_g}$ is the number of first accessed points $p \in V_l$ whose upper scores exceed r. In worst case, the time cost of updating the max-heap H is $O(\log(\hbar_g - \hat{\hbar_g}))$. For $\hbar_g < \binom{\hbar}{l}$ and $\hat{\hbar_g} \le \hbar_g$, the cost of Lines 2 to 11 of the FA is

$$O\left(\hbar_g + \hat{\hbar_g} + \log(\hbar_g - \hat{\hbar_g})\right) < O\left(\binom{\hbar}{l}\right).$$

The time complexity of the FA is $O\left(2^{\hbar} + \binom{\hbar}{l}\right) = O(2^{\hbar})$.

4.2. The hybrid algorithm

The FA in Section 4.1 needs to compute all the G-Skylines first, then identifies the G-Skylines with top k highest scores. In this subsection, we propose the hybrid algorithm, called HA, which merely generates the G-Skylines that are likely to be the final TkGSky query results. Besides, we present a new algorithm, namely GSky, which can compute the G-Skylines directly and progressively.

The HA algorithm is developed based on the following properties and lemma.

Propety 1. For a given group $G \subseteq SL_1$ of size l, it can be returned as a G-Skyline directly.

Propety 2. For a given G-Skyline G of size l, G contains i points within SL_1 for $1 \le i \le l$.

Lemma 4.2. *For a G-Skyline G with $G_1 = G \cap SL_1$, G is not a final result of the TkGSky query if $Score(G_1) - (l - |G_1|) < r$. Here r is the current kth highest score.*

Proof. Due to Lemma 3.1, the score of the G-Skyline G is computed as

$$Score(G) = Score(G_1) - |G - G_1| = Score(G_1) - (l - |G_1|).$$

On the assumption that $Score(G_1) - (l - |G_1|) < r$, it holds that $Score(G) < r$. Therefore, G is not a final result due to Definition 4 and this lemma holds. □

As shown in Algorithm 2, the HA first initializes a max-heap H to store the G-Skylines with top k highest scores (Line 1). Line 2 initializes the kth highest score r to 0. Then, Line 3 generates the G-Skylines $G \subseteq SL_1$ of size l directly. Lines 4 to 8 pick out the G-Skylines with top k highest scores and add them to the heap H. The left part of the HA (Lines 9 to 14) is an iteration procedure to generate other G-Skylines whose scores may exceed r. In each iteration, it generates groups G_1 that consist of i points within the first skyline layer SL_1. Here i is reduced from $l-1$ to 1 by a step of 1. By this way, the G-Skylines that are likely to have large scores

Algorithm 2 Hybrid Algorithm (HA) for TkGSky

Input: A DSG DS over D, group size l, and the number of results k
Output: k G-Skylines with the highest scores
1: Initialize a max-heap $H=\emptyset$ to store the G-Skylines with top k highest scores
2: Initialize the kth highest score $r\leftarrow 0$
3: Generate G-Skylines $G\subseteq SL_1$ of size l //Property 1
4: **for** each G-Skyline G **do**
5: **if** $Score^u(G)\geq r$ **then**
6: $Score(G)\leftarrow Score(G)-|G-G\cap SL_1|$ //Lemma 3.1
7: **if** $Score(G)\geq r$ **then**
8: Update H and r using $<G, Score(G)>$
9: **for** $i\leftarrow l-1$ to 1 **do**
10: Generate groups $G_1\subseteq SL_1$ of size i //Property 2
11: Add the groups G_1 with $Score(G_1)-(l-i)\geq r$ into V_i due to Lemma 4.2 and sort them in non-increasing order of their scores
12: Generate G-Skylines G by invoking the GSky Generator algorithm with the inputs V_i, l, and DS
13: **for** each G-Skyline G **do**
14: Update H and r using $<G, Score(G)>$
15: Return k G-Skylines with the highest scores from H

are generated in priority. If $Score(G_1)-(l-i)\geq r$, the groups G_1 are inserted into V_i according to Lemma 4.2. Thereafter, the groups within V_i are sorted in non-increasing order of their scores. Line 12 invokes the GSky algorithm illustrated in Algorithm 3 to compute the G-Skylines G with taking into account the groups within V_i. Lines 13 and 14 update H and r using the new G-Skylines. At last, H contains all the final results of the TkGSky query.

To generate the G-Skylines G containing a specialized group G_1, we present the GSky algorithm based on the layered strategy.

Definition 8 (*Direct Child Set, DChiSet*)**.** Given an point p', p' is a direct child of p if there is an edge directly connecting the nodes of p and p' in the DSG. For a given point p, its direct child set DChiSet(p, i) contains all the direct children of p in the ith skyline layer SL_i. Besides, for a given group G, we have the direct child set

$$\text{DChiSet}(G, i)=\bigcup_{p\in G}\text{DChiSet}(p, i).$$

Taking into account the hotel h_4 in Fig. 3, we have DChiSet(h_4, 2)=$\{h_7\}$. For the hotel group $\{h_2, h_3\}$, it holds DChiSet($\{h_2, h_3\}$, 3) =$\{h_8, h_9, h_{10}\}$.

Definition 9 (*Maximum Layer*)**.** For a given group G, its maximum layer is denoted as

$$\text{max\_layer}(G)=\max_{p\in G} p.\text{layer}.$$

Here p.layer is the skyline layer that the point p belongs to.

For the hotel group $\{h_3, h_6\}$, $max\_layer(\{h_3, h_6\})=\max\{h_3.layer, h_6.layer\}=2$.

As depicted in Algorithm 3, the GSky algorithm takes the direct graph DS, the group size l, and the set V as the inputs. If V is not empty, Lines 2 to 11 are executed to generate G-Skylines based on each group within V. For each group $G'\in V$, Line 4 computes the direct child set of G' over the (max_layer+1)th skyline layer. Lines 5 to 11 take into account the groups $DC'\subseteq$DChiSet(G', max_layer +1) of size not more than $k-|G'|$. For the groups DC' satisfying ParSet(DC')$\subseteq G'$, new groups G'' are generated by merging it with G'. If the groups G'' contains less than l points, then it is added to V. The new groups G'' of size l are returned as G-Skylines. At last, Line 12 updates V by removing the groups G' from it.

Algorithm 3 G-Skyline (GSky) Generator Algorithm

Input: A DSG DS over D, group size l, and a group set V
Output: G-Skylines that contain some group within V
1: **while** V is not empty **do**
2: **for** each candidate group G' in V **do**
3: $max\_layer\leftarrow \max_{p\in G'} p.layer$
4: Compute $DChiSet(G', max\_layer+1)$
5: **for** each $DC'\subseteq DChiSet(G', max\_layer+1)$ with $|DC'|\leq k-|G'|$ **do**
6: **if** $ParSet(DC')\subseteq G'$ **then**
7: Generate a new group $G''\leftarrow G'\cup DC'$
8: **if** $|G''|<l$ **then**
9: Add G'' into V
10: **else**
11: Report G'' of size l as G-Skylines
12: $V\leftarrow V-\{G'\}$

Example. Reconsider the example in Fig. 1. The HA first generates the hotel groups $\{h_1, h_2, h_3\}$, $\{h_1, h_2, h_4\}$, $\{h_1, h_3, h_4\}$, and $\{h_2, h_3, h_4\}$ that only consist of points within SL_1. The hotel groups $\{h_1, h_2, h_4\}$, $\{h_1, h_3, h_4\}$ and $\{h_2, h_3, h_4\}$ with top 2 highest scores are inserted into the max-heap H. Now, the current 2th highest score $r=Score(\{h_1, h_2, h_3\})=9$. Next, the hotel group $\{h_3, h_4\}$ that contains two points within the first skyline layer SL_1 are generated and added to V_2. We have $V_2=\{\{h_3, h_4\}\}$. Then, by applying the GSky algorithm, we build the G-Skylines based on the group $\{h_3, h_4\}\in V_2$. For the group $\{h_3, h_4\}$, its direct child set in the 2nd skyline layer is $\{h_5, h_6, h_7\}$. Since $ParSet(h_6)\subseteq\{h_3, h_4\}$ and $ParSet(h_7)\subseteq\{h_3, h_4\}$, we can get two new G-Skylines $\{h_3, h_4\}\cup\{h_6\}$ $=\{h_3, h_4, h_6\}$ and $\{h_3, h_4\}\cup\{h_7\}=\{h_3, h_4, h_7\}$. Moreover, the hotel groups $\{h_1\}$, $\{h_2\}$, $\{h_3\}$, and $\{h_4\}$ that contain only one point within SL_1 are not taken into account. This is because the scores of the G-Skylines generated based on them are all less than $r=9$ due to Lemma 4.2. In this example, the FA needs to generate all the 11 G-Skylines while the HA in this section only generates 6 G-Skylines.

Complexity. In the HA, Lines 3 to 8 costs $O\binom{\hbar_1}{l}$ to generate the G-Skylines consisting of l points within the first skyline layer SL_1, and use them to create and update the max-heap H. Lines 9 to 14 are a for loop. Line 10 generates the groups containing i points within SL_1 and it costs $O\binom{\hbar_1}{i}$. Line 12 invokes the GSky algorithm to generate the G-Skylines based on the candidate groups within the set V_i. Taking into account each group within V_i for $1\leq i\leq l-1$, new candidate groups are generated by adding corresponding direct children. This costs $O\left(\binom{\hbar_1}{i}\times|DC'|\right)$. Suppose that the maximum size of candidate groups generated on a group $G'\in V_i$ is ∂_i. We have the time complexity of the GSky algorithm is $O\left(\binom{\hbar_1}{i}\times\partial_i\right)$. The cost of updating H in Line 14 is equal to $O\left(\binom{\hbar_1}{i}\times\partial_i h\right)$ where h is the maximum height of H. Therefore, the time complexity of the HA is

$$O\left(\binom{\hbar_1}{l}+\sum_{i=1}^{l-1}\left(\binom{\hbar_1}{i}+\binom{\hbar_1}{i}\times\partial_i+\binom{\hbar_1}{i}\times\partial_i\times h\right)\right)<O\left(h\partial 2^{\hbar_1}\right)$$

for $\partial=\max\partial_i$.

4.3. The point-based replacement algorithm

In this subsection, we develop the point-based replacement algorithm, called PR, for the TkGSky query. In the PR, the G-Skylines G only consisting of points within the first skyline layer are generated first. Then, based on these G-Skylines, other G-Skylines that are likely to be the final query results are generated by the following point-based replacement strategy.

Lemma 4.3. *Given a G-Skyline G of size l, a point $p \in G$ that dominates no other point within G, and a point p' with $ParSet(p') \subseteq G-\{p\}$, we can get a new G-Skyline G' by merging $G-\{p\}$ with $\{p'\}$.*

Proof. Since G is a G-Skyline of size l, for any point $\hat{p} \in G$, we have $ParSet(\hat{p}) \subseteq G$. Because $p \in G$ and p dominates no other point within G, $G-\{p\}$ is a G-Skyline of size $l-1$. For the point p' with $ParSet(p') \subseteq G-\{p\}$, it holds that $G'=(G-\{p\}) \cup \{p'\}$ is dominated by no other group of equal size and it is a G-Skyline of size l on basis of Definition 4. Therefore, this lemma holds. □

For the G-Skyline $G=\{h_1, h_3, h_4\}$ in Fig. 1, the hotel $h_1 \in G$ is not dominated by other hotel $h_3, h_4 \in G$ and the hotel h_7 is a child of h_4 satisfying $ParSet(h_7)=\{h_4\} \subseteq G-\{h_1\}$. According to Lemma 4.3, we have another hotel group $(G-\{h_1\}) \cup \{h_7\}=\{h_3, h_4, h_7\}$ which is a G-Skyline.

Lemma 4.4. *For G-Skylines $G \subseteq SL_1$ whose scores are no more than the current kth largest score r, it holds that the G-Skylines G' with $G' \cap SL_1 \subset G \cap SL_1$ are not final results of the TkGSky query and could be pruned safely.*

Proof. Since $G' \cap SL_1 \subset G \cap SL_1$, it holds $Score(G') < Score(G)$ due to Eq. (2). For $Score(G) \leq r$, we have $Score(G') < r$ and G' could be pruned safely due to Definition 4. Therefore, this lemma holds. □

Due to Lemma 4.4, for two given G-Skylines G and G' with $G \subseteq SL_1$ and $G' \cap SL_1 \subset G \cap SL_1$, G' could be pruned safely if $Score(G) < r$. This means that the new G-Skylines G' by replacing some point within G are not final results of TkGSky, and we only need to compute and store the G-Skylines $G \subseteq SL_1$ with top k highest scores.

In Fig. 1, $\{h_1, h_2, h_3\}$ is a G-Skyline. Since $Score(\{h_1, h_2, h_3\})=8 < r=9$, the G-Skylines $\{h_1, h_3, h_6\}$ and $\{h_2, h_3, h_6\}$ with $\{h_1, h_3, h_6\} \cap SL_1 \subseteq \{h_1, h_2, h_3\}$ and $\{h_2, h_3, h_6\} \cap SL_1 \subseteq \{h_1, h_2, h_3\}$ are not the final results of the TkGSky query with $k=2$ based on Lemma 4.4.

Algorithm 4 Point-based Replacement (PR) Algorithm for TkGSky

Input: A DSG DS over D, group size l, and the number of results k
Output: k G-Skylines with the highest scores

1: Sort the points $p \in SL_1$ in non-decreasing order of $Score(p)$
2: Initialize a max-heap $H=\emptyset$ to store the G-Skylines with top k highest scores and initialize the kth highest score $r \leftarrow 0$
3: Generate the G-Skylines $G \subseteq SL_1$ of size l //Property 1
4: **for** each G **do**
5: **if** $Score^u(G) \geq r$ **then**
6: $Score(G) \leftarrow Score(G) - |G - G \cap SL_1|$ //Lemma 3.1
7: **if** $Score(G) \geq r$ **then**
8: Update H and r using $<G, Score(G)>$
9: Initialize a max-heap $H' \leftarrow H$ and a set $V \leftarrow \emptyset$
10: **while** H' is not empty **do**
11: Remove top entry $<G, Score(G)>$ from H'
12: **for** each point $p \in G$ **do**
13: **if** $Score(G-\{p\}) \geq r$ and $p \neq p'$ for $p' \in G$ **then**
14: **if** $G-\{p\} \notin V$ **then**
15: Add p to a set R and insert $G-\{p\}$ into V
16: Sort the points $p \in R$ in non-ascending order of $Score(G-\{p\})$
17: **for** each $p \in R$ **do**
18: Add the point p' with $ParSet(p') \subseteq G-\{p\}$ to $G-\{p\}$ to form a new G-Skyline G'' //Lemma 4.4
19: **if** $Score(G'') \geq r$ **then**
20: Update H, H', and r using $<G'', Score(G'')>$
21: $R \leftarrow R-\{p\}$
22: Return k G-Skylines with the highest scores from H

As illustrated in Algorithm 4, the PR first sorts the points within the first skyline layer SL_1 in non-ascending order of their scores. Line 2 initializes a max-heap H to store the G-Skylines with the top k highest scores, and initializes the kth highest score r to 0. Line 3 generates the G-Skylines G of size l satisfying $G \subseteq SL_1$. Then, the G-Skylines G with top k highest scores are computed and stored in the max-heap H. Line 9 initializes a max-heap H' to the max-heap H. Lines 10 to 21 are an iteration procedure to generate other G-Skylines that may have top k highest scores. If H' is not empty, the top entry is removed and we get the G-Skyline G with the highest score. Next, Lines 12 to 15 compute each point $p \in G$ that can be replaced based on Lemma 4.3. If the score of $G-\{p\}$ exceeds r, there is not any point $p' \in G$ dominated by p, and the group $G-\{p\}$ is not in the set V, then the point p is added to the set R and the group $G-\{p\}$ is inserted to V. Here the set V is utilized to store the candidate groups $G-\{p\}$ that have been taken into account to reduce redundant computation. The group $G-\{p\} \in V$ can be overlooked because all the corresponding G-Skylines based on the group $G-\{p\}$ have been generated before. Line 16 sorts the points $p \in R$ in non-ascending order of $Score(G-\{p\})$. Lines 17 to 21 are executed to generate new G-Skylines G'' whose scores are no less than r. A new G-Skyline G'' is built by replacing one point $p \in G$ with a point p' whose parents are all contained in $G-\{p\}$ at a time. If DomSize(G'') exceed r, then G'' are added to the max-heap H'. Besides, H could be refreshed by $<G'', Score(G'')>$. At last, H contains all the G-Skylines with top k highest scores.

Example. In Fig. 1, the hotel groups $\{h_1, h_3, h_4\}$, $\{h_2, h_3, h_4\}$, and $\{h_1, h_2, h_4\}$ are generated and inserted into H since they have top 2 highest scores. These groups only consist of points within the first skyline layer SL_1. Here we have $r=Score(\{h_1, h_2, h_4\})=9$. The max-heap H' is initialized to H. Next, the hotel group $G=\{h_1, h_3, h_4\}$ which has the highest score is removed from H'. The hotel h_1 within $\{h_1, h_3, h_4\}$ is added to the set R since $Score(\{h_1, h_3, h_4\} - \{h_1\})=10 > r=9$, h_1 cannot dominate h_3 and h_4, and $\{h_1, h_3, h_4\} - \{h_1\}=\{h_3, h_4\}$ is not contained in V. For the hotel h_3, it is not added to the set R because the score of $\{h_1, h_2, h_3\} - \{h_3\}$ is less than $r=9$. Similarly, the hotel h_4 is not added to R, and we have $R=\{h_1\}$. By removing $h_1 \in R$, we have $G-\{h_1\}=\{h_3, h_4\}$. New G-Skylines $\{h_3, h_4, h_6\}$ and $\{h_3, h_4, h_7\}$ are gained by merging $\{h_3, h_4\}$ with $\{h_6\}$ and $\{h_7\}$, respectively. Here the parents of $\{h_6\}$ or $\{h_7\}$ are both contained in $G-\{h_1\}$. Now we have $V=\{\{h_3, h_4\}\}$. The G-Skyline $G=\{h_1, h_3, h_4\}$ or $\{h_1, h_2, h_4\}$ is not taken into account in this replacing process because $Score(G-\{p\})$ is less than $r=9$ for each point $p \in G$. After this replacing process, the new G-Skylines $\{h_3, h_4, h_6\}$ and $\{h_3, h_4, h_7\}$ are added to the heap H. Finally, the max-heap H contains the G-Skylines $\{h_1, h_3, h_4\}$, $\{h_2, h_3, h_4\}$, $\{h_1, h_2, h_4\}$, $\{h_3, h_4, h_6\}$, and $\{h_3, h_4, h_7\}$ which are with top 2 highest scores.

Complexity. The PR costs $O(h_1 \log h_1)$ to sort the points within SL_1 where $h_1=|SL_1|$. Lines 3 to 8 costs $O\binom{h_1}{l}$ to generate the G-Skylines consisting of l points within SL_1, and use them to create and update the max-heap H. Assume that the maximum size of H is N_H and for each group G from H, it generates at most β new G-Skylines by the point-based replacement strategy. The cost of updating H is $O(\beta N_H \log N_H)$. Consequently, the time complexity of PR is

$$O\left(h_1 \log h_1 + \binom{h_1}{l} + \beta N_H \log N_H\right) = O\left(2^{h_1} + \beta N_H \log N_H\right).$$

4.4. The approximate point-based replacement algorithm

From the experimental results, it is time-consuming when processing the TkGSky query over a data set with a large group size, high dimensionality, or large cardinality. This is similar to other group skyline queries and their variants. Consider accurate results are not required in many real-life applications. In

Table 2
System Parameters.

Parameters	Values
Dimensionality(d)	2, **3**, 4, 5
Group size (l)	2, 3, **4**, 5
Number of returned results (k)	20, 40, 60, 80, 100
Cardinality (N) for Cor	20,000, 40,000, 60,000, 80,000, **100,000**
Cardinality (N) for Ind and Ant	2000, 4000, 60,000, 80,000, 100,000

this subsection, we propose an approximate point-based replacement (APR) algorithm to process the TkGSky query. In the APR algorithm, we apply the α-allowance technique [36] to boost the TkGSky query performance. This is because the quality of approximate results for the α-allowance method is very high.

The pseudo codes of the APR and PR algorithms are similar. However, the APR algorithm is different from the PR algorithm in the following aspects. As well as the inputs of the PR algorithm, it takes a parameter α as an important input. Here $0<\alpha\leq 1$. The APR algorithm first sorts points $p\in SL_1$ in descending order of their scores. Here SL_1 contains all the points within the first skyline layer. Then, it selects top $\alpha\times 100$ percentage for $0<\alpha\leq 1$ of the points $p\in SL_1$ and stores these points in a new set SL_1'. In line 3 of the APR algorithm, only the points $p\in SL_1'$ are taken into account.

5. Performance evaluation

Our experiments use both synthetic and real-life datasets. A number of experiments have been completed on synthetic datasets with three popular distributions: Independent (Ind), Correlated (Cor), and Anti-correlated (Ant) following the closely related work in [4]. Furthermore, the real data set, NBA in [4], is also adopted. Specifically, three attributes including the numbers of points scored, rebounds, and assists are taken into account.

To the best of our knowledge, this is the first work for the TkGSky query. In this paper, we extend the UWise+ algorithm [4] for the G-Skyline query, and develop FA by integrating the group strategy and the upper bound pruning strategy. In addition, FA is taken as a baseline. In the following experiments, we evaluate the performance of the following algorithms.

- FA: The Fast Algorithm (FA) in Section 4.1.
- HA: The Hybrid Algorithm (HA) in Section 4.2.
- PR: The Point-based Replacement (PR) algorithm in Section 4.3.
- APR: The Approximate Point-based Replacement (APR) algorithm in Section 4.4.

The experiments were performed on a PC with Intel® Core™ I7-6700T 2.81 GHz CPU (contains 4 cores), 8GB main memory, and under the Microsoft Windows 7 operation system.

In order to compare the results in different scenarios, we tested one parameter at a time by varying all of its possible values while fixing the other three parameters to their default values, listed in bold. The four parameters with their possible values are listed in Table 2. The range of the cardinality of the Cor datasets is different from that of the Ind and Ant datasets which is similar to [26]. This is because the proposed algorithms over large Ind and Ant datasets need unacceptable query time. In the following experiments, the default values of N over the Ind and Ant datasets are set to 1000, and the default value of k is 10.

5.1. Performance on synthetic datasets

In this subsection, we evaluate the performance of the proposed algorithms in processing the TkGSky query. We research the effect of the parameters including the dimensionality d, the cardinality of the data set N, the group size l, and the number of returned results k.

5.1.1. Effect of dimensionality d

In the first set of experiments, we investigate the influence of the dimensionality d on the performance of the three algorithms. Fig. 4 shows the query time of the proposed algorithms with varying d from 2 to 6 by a step of 1 on Ind, Cor, and Ant, respectively.

The dimensionality d has a great impact on the performance of FA, HA, and PR. As d increases, the number of points within the first l=4 skyline layers grows sharply, and the size of G-Skylines grows in turn. Therefore, the query time of the three algorithms all increases with the growth of d. HA and PR require less time than FA. This is as expected because FA needs to compute all the G-Skylines while HA and PR only generate partial G-Skylines that are most likely to be the final results. Besides, PR performs best among the three algorithms.

There is also an interesting phenomena that over the Ind datasets, query time of HA and PR reduces as d varying from 5 to 6. The time cost of computing the G-Skylines increases as the dimensionality d grows [4]. However, comparing to the Ind data set with d=5, it may need much less time to compute the scores of the G-Skylines over the Ind data set with d=6. This is because the points within the G-Skylines may have less children, and the time cost to compute the scores of the G-Skylines is reduced.

5.1.2. Effect of Cardinality N

In the second set of experiments, we evaluate the impact of the cardinality N on the performance of the proposed algorithms.

For the Ind and Ant datasets, we vary N from 2000 to 10 000 by a step of 2000. Besides, for the Cor datasets, the cardinality N is changed from 20 000 to 100 000 by a step of 20 000. Similar to [26], we offer this different setting because the query time of the proposed algorithms over large Ind and Ant datasets is unacceptably long.

Fig. 5 depicts the query time of the proposed algorithms with the growth of N. Over the Ind, Cor, and Ant datasets, the query time of all the algorithms increases as the cardinality N grows. The HA and PR algorithms both outperform the FA algorithm. Again, PR has the best performance.

Over the Ind and Cor datasets, PR has a definite advantage with comparison to HA. However, over the Ant datasets, the query time of PR is close but also less than that of HA. The reason is that both HA and PR require most of the time to compute the G-Skylines $G\subseteq SL_1$ with top k highest scores, and the time to compute the left G-Skylines is too short.

5.1.3. Effect of group size l

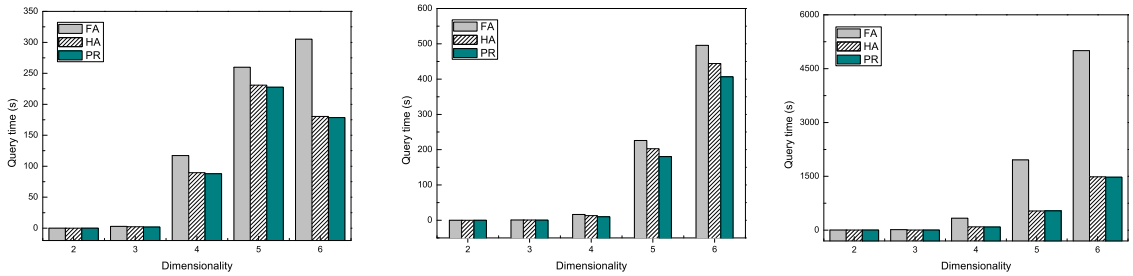
In the third set of experiments, we study the effect of the group size l on the performance of the proposed algorithms with l varying from 2 to 6 by a step of 1.

Fig. 6 shows the query time of FA, HA, and PR as the growth of l over the Ind, Cor, and Ant datasets, respectively. The group size l affects the performance of the three algorithms significantly. This is the same as the dimensionality d. The G-Skylines only contain the points within the first l skyline layers. As l increases, the size of candidate groups dramatically increases, the size of the G-Skylines becomes large with the increase of l, and the time cost to compute the scores of the G-Skylines increases in turn. Again, PR outperforms the other two algorithms in terms of query time. The Ant datasets need much more query time than the Ind and Cor datasets.

5.1.4. Effect of number of returned results k

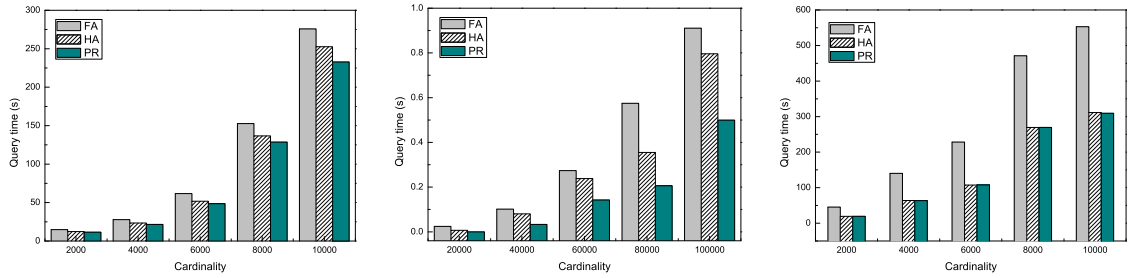
In the fourth set of experiments, we evaluate the performance of FA, HA, and PR by varying k from 20 to 100 by a step of 20.

Fig. 7 illustrates the query time of the proposed algorithms over the Ind, Cor, and Ant datasets, respectively. The three algorithms are not very sensitive to different k values. This is similar



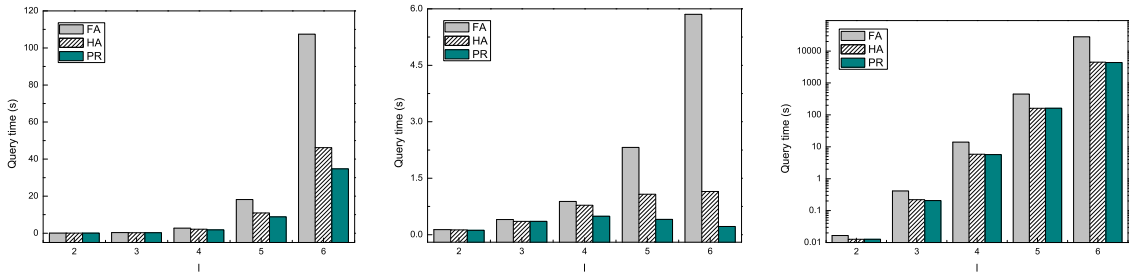
(a) $N{=}10^3, l{=}4, k{=}10$ (b) $N{=}10^5, l{=}4, k{=}10$ (c) $N{=}10^3, l{=}4, k{=}10$

Fig. 4. Query time vs. Dimensionality d.(a) Ind. (b) Cor. (c) Ant.



(a) $d{=}3, l{=}4, k{=}10$ (b) $d{=}3, l{=}4, k{=}10$ (c) $d{=}3, l{=}4, k{=}10$

Fig. 5. Query time vs. Cardinality N. (a) Ind. (b) Cor. (c) Ant.



(a) $d{=}3, N{=}10^3, k{=}10$ (b) $d{=}3, N{=}10^5, k{=}10$ (c) $d{=}3, N{=}10^3, k{=}10$

Fig. 6. Query time vs. Group size l. (a) Ind. (b) Cor. (c) Ant.

to the top k dominating query [37]. As k grows, the query time of the proposed algorithms over the Ind and Cor datasets increases slightly. But, over the Ant datasets, the query time grows significantly as the increase of k. With varying k, PR has the best performance among the three algorithms.

5.2. Performance of the APR algorithm

In this subsection, we evaluate the APR algorithm by comparing them with the PR algorithm in term of processing time and approximate ratio (AR). Here, the approximate ratio is the percentage of the approximate results returned by APR that also exist in the exact results. The larger approximate ratio, the better quality of the approximate results.

Fig. 8 and Tables 3 to 6 depict results of the APR algorithm by varying α from 0.9 to 0.5. With the reduction of α, the processing time of APR decreases. The is because only $\alpha \times 100$ percent of the points within SL_1 are considered, and partial candidate groups are generated. When α is set to no less than 0.5, APR can get good approximate ratio that is mostly close to 1.

Table 3
Approximate ratio of APR vs. Cardinality N.

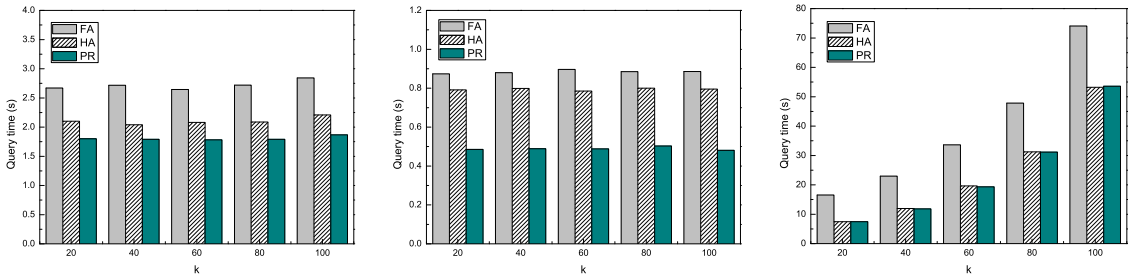
N	APR (0.9)	APR (0.8)	APR (0.7)	APR (0.6)	APR (0.5)
2000	1.00	1.00	1.00	1.00	0.90
4000	1.00	1.00	1.00	1.00	1.00
6000	1.00	1.00	1.00	1.00	0.40
8000	1.00	1.00	1.00	1.00	1.00
10,000	1.00	1.00	1.00	1.00	0.60

Table 4
Approximate ratio of APR vs. Dimensionality d.

d	APR (0.9)	APR (0.8)	APR (0.7)	APR (0.6)	APR (0.5)
2	1.00	0.70	0.40	0.40	0.40
3	1.00	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00

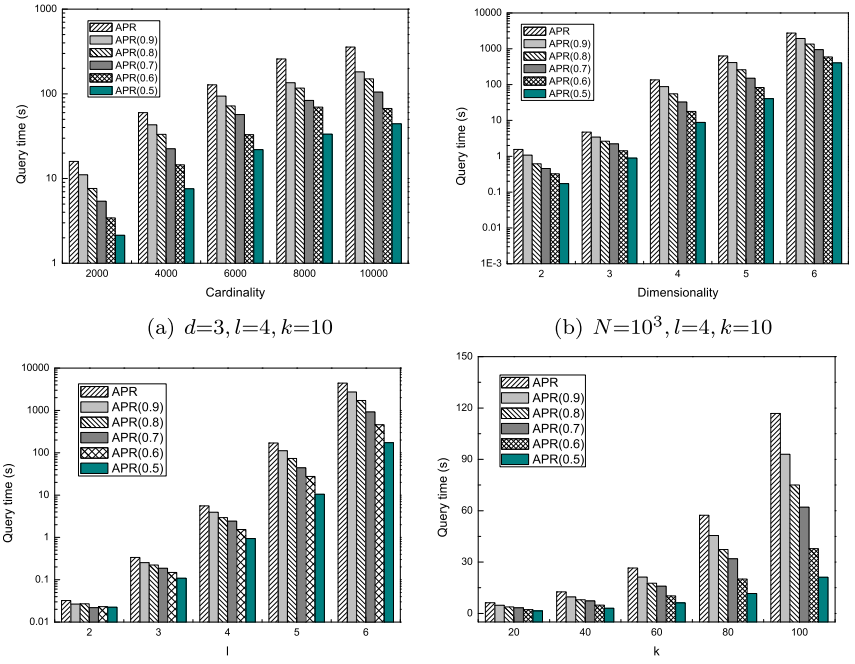
5.3. Performance on the NBA data set

In this subsection, we study the performance of the proposed algorithms over the real data set, NBA [4]. Basketball is a team



(a) d=3, N=10^3, l=4 (b) d=3, N=10^5, l=4 (c) d=3, N=10^3, l=4

Fig. 7. Query time vs. Number of returned results k. (a) Ind. (b) Cor. (c) Ant.



(a) d=3, l=4, k=10 (b) N=10^3, l=4, k=10

(c) N=10^3, d=3, k=10 (d) N=10^3, d=3, l=4

Fig. 8. Query time of the APR algorithm. (a) Cardinality N. (b) Dimensionality d. (c) Group size l (d) Number of returned results k.

Table 5
Approximate ratio of APR vs. Group size l.

l	APR (0.9)	APR (0.8)	APR (0.7)	APR (0.6)	APR (0.5)
2	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	0.80
4	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00

Table 6
Approximate ratio of APR vs. Number of returned results k.

k	APR (0.9)	APR (0.8)	APR (0.7)	APR (0.6)	APR (0.5)
20	1.00	1.00	1.00	1.00	1.00
40	1.00	1.00	1.00	1.00	1.00
60	1.00	1.00	1.00	1.00	1.00
80	1.00	1.00	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00

sport in which two teams, most commonly of five players each. Therefore, we set the group size l=5 in the experiments.

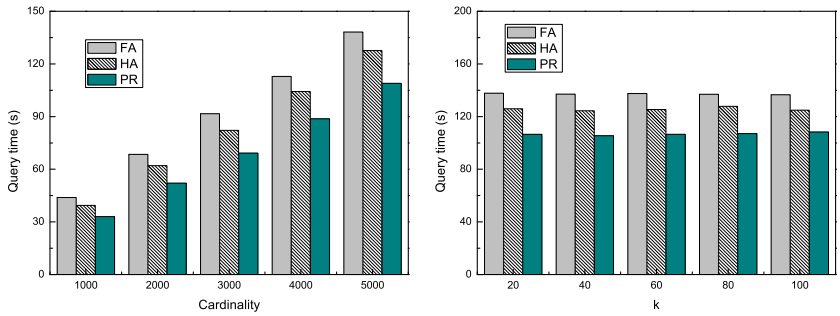
Fig. 9 illustrates the query time of the proposed exact algorithms with varying the cardinality N and the number of results k, respectively. Besides, the experimental results of the APR algorithm are shown in Fig. 10.

The experimental results on NBA are consistent with the ones obtained from the experiments on the synthetic datasets. From Fig. 9(a), the query time of all the algorithms increases as the cardinality N grows. The number of returned results k has not a significant influence on the performance of the exact algorithms as shown in Fig. 9(b). From Fig. 9, FA requires the most query time, and PR needs the minimal query time among the three exact algorithms.

Fig. 10 shows experimental results of the APR algorithm over NBA with the growth of the cardinality N. When N increases, the query time of APR with different α all increases. Again, k has a little effect on the performance of APR. Besides, the larger α makes the less query time of APR. In most cases, we can get all the results of the TkGSky query by the APR algorithm if α is larger than 0.5 (see Tables 7 and 8).

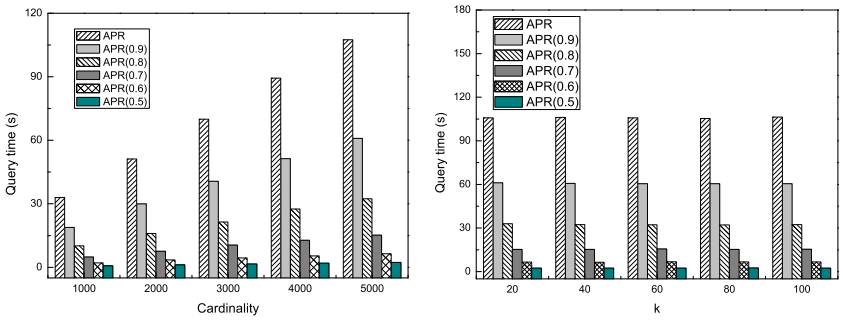
5.4. Summary

As analyzed above, over the Ind, Cor, and Ant datasets, the query time of the proposed algorithms all increases as the growth



(a) d=3, l=5, k=10 (b) N=5 ∗ 10^3, l=5, d=3

Fig. 9. Experimental results of the proposed exact algorithms over NBA. (a) Cardinality N. (b) Number of returned results k.



(a) d=3, l=5, k=10 (b) N=5 ∗ 10^3, l=5, d=3

Fig. 10. Experimental results of APR over NBA. (a) Cardinality N. (b) Number of returned results k.

Table 7
Approximate ratio vs. Cardinality N over NBA.

N	APR (0.9)	APR (0.8)	APR (0.7)	APR (0.6)	APR (0.5)
1000	1.00	1.00	1.00	1.00	1.00
2000	1.00	1.00	1.00	1.00	1.00
3000	1.00	1.00	1.00	0.80	0.40
4000	1.00	1.00	1.00	1.00	1.00
5000	1.00	1.00	1.00	0.90	0.50

Table 8
Approximate ratio vs. Number of returned results k over NBA.

k	APR (0.9)	APR (0.8)	APR (0.7)	APR (0.6)	APR (0.5)
20	1.00	1.00	1.00	1.00	1.00
40	1.00	1.00	1.00	1.00	1.00
60	1.00	1.00	1.00	1.00	1.00
80	1.00	1.00	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00

of the dimensionality d, the cardinality N, and the group size l. The three algorithms are not sensitive to the number of results k. The dimensionality d and the group size l have great impacts on the performance of the proposed algorithms. Among FA, HA, and PR, PR has the best performance. The PR algorithm always costs the least query time and is the most stable. In addition, all the algorithms need much more query time to process the Ant datasets compared with the Ind and Ant datasets.

The TkGSky query may cost a long time especially for the datasets with large cardinality, high dimension d, or large group size l. This has two reasons: firstly, it is time-consuming to compute the score of each candidate group G. For a candidate group G, all the points dominated by any point $p \in G$ need to be identified; secondly, the number of the candidate groups grows sharply as the cardinality, dimension d, or group size l increases. A large number of candidate groups will also make the query time prohibitively long.

6. Conclusions

Group skyline query is a powerful tool for analyzing groups of different points. In [4], Liu et al. formulated the GSky query which can report much more comprehensive results than other group skyline queries. However, the GSky query may return numerous results which make users overwhelmed. In this paper, we investigate the GSky query with a size constraint. We formulate the TkGSky query that aims to retrieve manageable size of optimal groups. Besides, we develop three algorithms, where the upper bound pruning, hybrid strategy, and point-based replacement strategy are applied, for the TkGSky query. Experimental results demonstrate that the proposed algorithms are efficient and scalable. An interesting future work is to consider the TkGSky query in a parallel or distributed environments to get better scalability for data sets with large cardinality or high dimensionality. Besides, it is also significant to introduce the group rating prediction in [38] to the TkGSky query.

Acknowledgments

We are very grateful to the reviewers for their constructive comments which are helpful for us to improve the manuscript. The research was partially funded by the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), the National Natural Science Foundation of China (Grant Nos. 61772182, 61802032, 61806077, 61602170), and Science and Technology Plan of Changsha, China (K1705032).

References

[1] Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger, Progressive skyline computation in database systems, ACM Trans. Database Syst. 30 (1) (2005) 41–82.

[2] Hyeonseung Im, Sungwoo Park, Group skyline computation, Inform. Sci. 188 (2012) 151–169.
[3] Shiming Zhang, Nikos Mamoulis, David W. Cheung, Scalable skyline computation using object-based space partitioning, in: Proc. ACM SIGMOD Int. Conf. Manag. Data, ACM, 2009, pp. 483–494.
[4] Jinfei Liu, Li Xiong, Jian Pei, Jun Luo, Haoyu Zhang, Finding pareto optimal groups: Group-based skyline, Proc. VLDB Endow. 8 (13) (2015).
[5] Yu-Chi Chung, I-Fang Su, Chiang Lee, Efficient computation of combinatorial skyline queries, Inf. Syst. 38 (3) (2013) 369–387.
[6] Nan Zhang, Chengkai Li, Naeemul Hassan, Sundaresan Rajasekaran, Gautam Das, On skyline groups, IEEE Trans. Knowl. Data Eng. 26 (4) (2014) 942–956.
[7] Xun Zhao, Yanhong Wu, Weiwei Cui, Xinnan Du, Yuan Chen, Yong Wang, Dik Lun Lee, Huamin Qu, SkyLens: Visual analysis of skyline on multi-dimensional data, IEEE Trans. Vis. Comput. Graphics PP (99) (2017) 1–1.
[8] Yunjun Gao, Qing Liu, Baihua Zheng, Li Mou, Gang Chen, Qing Li, On processing reverse k-skyband and ranked reverse skyline queries, Inform. Sci. 293 (2015) 11–34.
[9] Kenneth S. Bøgh, Sean Chester, Ira Assent, SkyAlign: a portable, work-efficient skyline algorithm for multicore and GPU architectures, VLDB J. 25 (6) (2016) 1–25.
[10] Sean Chester, Ira Assent, Work-efficient parallel skyline computation for the GPU, VLDB Endow. (2015) 962–973.
[11] Jia Ling Koh, Chia Ching Chen, Chih Yu Chan, Arbee L.P. Chen, MapReduce skyline query processing with partitioning and distributed dominance tests, Inform. Sci. 375 (2017) 114–137.
[12] Jongwuk Lee, Hyeonseung Im, Gae Won You, Optimizing skyline queries over incomplete data, Inform. Sci. 361–362 (2016) 14–28.
[13] Yoonjae Park, Jun Ki Min, Kyuseok Shim, Efficient processing of skyline queries using MapReduce, IEEE Trans. Knowl. Data Eng. 29 (5) (2017) 1031–1044.
[14] Dimitrios Pertesis, Skyline query processing in SpatialHadoop, Inf. Syst. 54 (2015) 325–335.
[15] Xu Zhou, Kenli Li, Yantao Zhou, Keqin Li, Adaptive processing for distributed skyline queries over uncertain data, IEEE Trans. Knowl. Data Eng. (2016) 371–384.
[16] Hua Lu, Christian S. Jensen, Zhenjie Zhang, Flexible and efficient resolution of skyline query size constraints, IEEE Trans. Knowl. Data Eng. 23 (7) (2011) 991–1005.
[17] Yunjun Gao, Qing Liu, Lu Chen, Gang Chen, Qing Li, Efficient algorithms for finding the most desirable skyline objects, Knowl.-Based Syst. 89 (C) (2015) 250–264.
[18] Mei Bai, Junchang Xin, Guoren Wang, Luming Zhang, Roger Zimmermann, Ye Yuan, Xindong Wu, Discovering the k representative skyline over a sliding window, IEEE Trans. Knowl. Data Eng. 28 (8) (2016) 2041–2056.
[19] Matteo Magnani, Ira Assent, Michael L. Mortensen, Taking the Big Picture: representative skylines based on significance and diversity, VLDB J. 23 (5) (2014) 795–815.
[20] Matteo Magnani, Ira Assent, From stars to galaxies: skyline queries on aggregate data, in: Proc. Int. Conf. Extending Database Technol., ACM, 2013, pp. 477–488.
[21] Tao Jiang, Bin Zhang, Dan Lin, Yunjun Gao, Qing Li, Incremental evaluation of top-k combinatorial metric skyline query, Knowl.-Based Syst. 74 (2015) 89–105.
[22] Xu Zhou, Kenli Li, Guoqing Xiao, Yantao Zhou, Keqin Li, Top k favorite probabilistic products queries, IEEE Trans. Knowl. Data Eng. (2016) 2808–2821.
[23] Xu Zhou, Kenli Li, Zhibang Yang, Keqin Li, Finding optimal skyline product combinations under price promotion, IEEE Trans. Knowl. Data Eng. 31 (1) (2019) 138–151.
[24] Yifu Zeng, Chen Guo, Li Kenli, Zhou Yantao, Zhou Xu, Li Keqin, M-Skyline: Taking sunk cost and alternative recommendation in consideration for skyline query on uncertain data, Knowl.-Based Syst. 163 (2019) 204–213.
[25] Wenhui Yu, Zheng Qin, Jinfei Liu, Li Xiong, Xu Chen, Huidi Zhang, Fast algorithms for pareto optimal group-based skyline, in: Proc. Int. Conf. on Information and Knowledge Management, 2017, pp. 417–426.
[26] Changping Wang, Chaokun Wang, Gaoyang Guo, Xiaojun Ye, Philip Yu, Efficient computation of g-skyline groups, IEEE Trans. Knowl. Data Eng. 30 (4) (2018) 674–688.
[27] Haoyang Zhu, Peidong Zhu, Xiaoyong Li, Qiang Liu, Peng Xun, Parallelization of groupbased skyline computation for multicore processors, Concurrency Comput. Pract. Exp. 29 (3) (2017) e4195.
[28] Eleftherios Tiakas, George Valkanas, Apostolos N. Papadopoulos, Yannis Manolopoulos, Dimitrios Gunopulos, Processing top-k dominating queries in metric spaces, ACM Trans. Database Syst. 40 (4) (2016) 1–38.
[29] Xiaoye Miao, Yunjun Gao, Baihua Zheng, Gang Chen, Huiyong Cui, Top-k dominating queries on incomplete data, IEEE Trans. Knowl. Data Eng. 28 (1) (2016) 252–266.
[30] Xixian Han, Jianzhong Li, Hong Gao, Efficient top-k dominating computation on massive data, IEEE Trans. Knowl. Data Eng. PP (99) (2017) 1–1.
[31] Bagus Jati Santoso, Ge Ming Chiu, Close dominance graph: An efficient framework for answering continuous top-k dominating queries, IEEE Trans. Knowl. Data Eng. 26 (8) (2014) 1853–1865.
[32] Zhian He, Eric Lo, Answering why-not questions on top-k queries, in: IEEE International Conference on Data Engineering, 2012, pp. 750–761.
[33] Liming Zhan, Ying Zhang, Wenjie Zhang, Xuemin Lin, Identifying top k dominating objects over uncertain data, in: Database Systems for Advanced Applications - International Conference, 2014, pp. 388–405.
[34] Daichi Amagata, Takahiro Hara, Makoto Onizuka, Space filling approach for distributed processing of top-k dominating queries, IEEE Trans. Knowl. Data Eng. PP (99) (2018) 1–1.
[35] Stephan Börzsönyi, Donald Kossmann, Konrad Stocker, The skyline operator, in: Proc. Int. Conf. Data Eng., 2001, pp. 421–430.
[36] Yunjun Gao, Lu Chen, Xinhan Li, Bin Yao, Gang Chen, Efficient k-closest pair queries in general metric spaces, VLDB J. 24 (3) (2015) 415–439.
[37] Xiang Lian, Lei Chen, Top-k dominating queries in uncertain databases, Inform. Sci. 226 (3) (2013) 23–46.
[38] Jiang Wenjun, Wu Jie, Wang Guojun, Zheng Huanyang, Forming opinions via trusted friends: Time-evolving rating prediction using fluid dynamics, IEEE Trans. Comput. 65 (4) (2016) 1211–1224.